A Topologically Aware Resource Management System

SHARIQ MOBEEN

KTH Information and Communication Technology

Degree project in
Communication Systems
Second level, 30.0 HEC
Stockholm, Sweden

A Topologically Aware Resource Management System

Shariq Mobeen

mobeen@kth.se

2014-01-15

Master's Thesis

MSV-RBS Department at Ericsson

Examiner: Professor Gerald Q. Maguire Jr.
Industrial Supervisor: Magnus Kronqvist

School of Information and Communication Technology (ICT)
KTH Royal Institute of Technology
Stockholm, Sweden

Abstract

As companies fight for market share whoever is able to bring products to market faster has an advantage over their competitors. Therefore it is absolutely essential to constantly evaluate and optimize processes to achieve shorter time-to-market for products.

These optimizations have to be carried out in all parts of a company. This thesis describes one such attempt made by a Swedish telecommunication vendor focused on enabling a resource management system to gain a greater understanding of the resources available during testing. This system manages all of the hardware utilized by the users, software testers, within one particular part of the organization and aids users by automatically converting the information stored in its database into a configuration file that will later be used in the testing framework's execution environment. Unfortunately, the current version of this resource management system lacks semantic understanding of the information necessary to automatically generate the configuration file, leaving a rather large part of the configuration file to be manually entered by the testers, a rather time-consuming task. The inability to completely automate the process means that the testing process is slower, more error prone, and increases the work needed for a new engineer to become a productive software tester.

In order for the resource management system to automatically generate the configuration file it needs to know not only *which* resources it is managing, but must also how these resources are *interconnected*, i.e. the topology of the resources. For this reason this thesis describes how to make the resource management system topologically aware, thus making verification of the System under Test (SUT) more efficient and mitigating the problems mentioned above. This thesis does *not* deal with the intricate details of how to automatically extract the topology, as this is inherently domain specific and thus difficult to generalize. Rather, this thesis focused on how to allow users to custom-build their desired topology by defining a set of rules that restrict how resources can be interconnected.

The goal of providing functionality for storing and retrieving topological information from database has been successfully achieved, and the resulting code has been integrated into the existing resource management system. However, the functionality has not yet been delivered because of a limitation in our front controller that stops us from providing an efficient web interface to our tool. After delivery the implemented solution is expected to remove most manual work related to test configuration and therefore also reduce the learning curve for new engineers.

Keywords: Topology, Resource Management System, SUT, Testing

Sammanfattning

När företag slåss om marknadsandelar har de som kan leverera produkter till marknaden snabbare en fördel över sina konkurrenter. Det är därför av högsta vikt att kontinuerligt utvärdera och optimera processer för att produkten snabbare skall nå marknaden..

Dessa optimeringar måste utföras inom samtliga områden inom ett företag. Denna uppsats beskriver ett sådant försök av ett svenskt telekombolag att stärka ett resurshanteringssystem för att uppnå en högre förståelse för de resurser den hanterar. Detta system hanterar samtlig hårdvara för användare (mjukvarutestare) inom en del av organisationen. Det hjälper användarna att automatiskt konvertera informationen i sin databas till en konfigurationsfil som används i testramverkets exekveringsmiljö. Tyvärr saknar den nuvarande versionen den semantiska förståelsen av dess data för att kunna automatiskt generera konfigurationsfilen, vilket tvingar användaren att manuellt ägna sig åt denna tidskrävande uppgift. Oförmågan att inte kunna automatisera fullt ut innebär att den övergripande testprocessen är långsammare, mer felbenägen och ökar tiden det tar för en ny ingenjör att komma igång och bli en produktiv mjukvarutestare.

För att resurshanteringssystemet ska kunna generera konfigurationsfilen automatiskt krävs inte bara kunskap om vilka resurser den hanterar utan också hur dessa är sammankopplade. Det vill säga hur de topologiskt relaterar till varandra. Den här uppsatsen beskriver därför hur ett resurshanteringssystem kan bli topologiskt medvetet och därigenom åstadkomma en mer effektiv testning av produkten och därmed överkomma de tidigare nämnda problemen. Denna uppsats inte gå in på detaljer om hur man extraherar den topologiska strukturen av resurser eftersom detta i sin natur är domänspecifikt och därigenom svårt att generalisera. Fokus istället ligga på hur man kan tillåta användare att bygga önskad topologi genom att definiera regler för hur olika resurser kan sammankopplas.

Målet vi satte upp med att kunna lagra och inhämta topologisk information från en databas har med framgång integrerats i det existerande resurshanteringssystemet. Ändringen är dock ännu inte fullt ut levererad på grund av en begränsning i vår nuvarande front controller som hindrar oss från att på ett effektivt sätt koppla samman vårt nya verktyg med ett webbgränsnitt. Efter leverans förväntas den implementerade lösningen eliminera större delen av det manuella arbete som tidigare krävts i samband med konfiguration av testmiljön, och därigenom även minska inlärningskurvan för nya ingenjörer.

Nyckelord: Topologi, Resurshanteringssystem, SUT, Testning

Acknowledgements

First of all, Thanks to ALLAH, the most merciful and most beneficent for giving me wisdom and allowing me to complete this thesis.

I would like to express my gratitude to **Professor Gerald Q. Maguire Jr.** for his assistance & supervision. His invaluable guidance helped us in all the difficulties we faced in the progress of this thesis project. I would like to give special thanks to my supervisor **Magnus Kronqvist** for helping me to develop my technical skills and for his continuous guidance and encouragement throughout this thesis process.

I wish to express my special thanks to Ahmed Kamal Mirza (Late), Muhammad Muaaz, and Waqas Liaqat Ali for their moral support and guidance during my study period. I also wish to express my gratitude to Abdul Rahim and Muhammad Fahad for their valuable recommendations of how to improve this thesis.

Last but not least, I want to acknowledge & express gratefulness to my family who showed great patience and endured with me throughout all.

Table of Contents

List of Figures

List of Tables

List of acronyms and abbreviations

CPP	Connectivity Packet Platform
DID	Device Identifier
DSR	Design Science Research
DU I&V	Digital Unit Integration and Verification
FT	Function Test
JCAT	Java Common Automated
JCT	Joint Common Test
LAB	Laboratory
MIS	Management Information System
MSV	Maintenance and System Verification
NCI	Node Common Infrastructure
png	portable network graphics
RBS	Radio Base Station
RMS	Resource Management System
ST	System Test
STP	System Test Plant
SUT	System Under Test
SW	Software
TF	Topological Framework
TEC	Test Environment Configuration
XML	extensible markup language

1 Introduction

This chapter presents a general introduction and gives the basic background knowledge required to understand this Master's thesis. Next it describes the motivation for this thesis project. Then, it summarizes the expected contributions in the field of testing that should result from this thesis project. Finally, the chapter concludes with a description of the structure of rest of thesis.

1.1 Problem definition

In the Maintenance and System Verification (MSV) - Radio Base Station (RBS) System verification department, each RBS is tested at the system level. During system verification, the RBS is tested both using *regression testing* (to ensure that it meets legacy requirements) and *feature testing* (to lead market trends by providing novel features).

Testing of a lot of different types of Test Environment Configurations (TECs) in conjunction with different RBS nodes requires a large amount of effort and time. Time and effort are both critical attributes in the quality of testing of the system under test (SUT). If more time is spent testing, then better quality of testing can be ensured (assuming that the additional testing time is spent testing a larger number of test cases which cover more of the system's functionality). However, the time required to input the TECs into a test environment is generally ignored. This leads to increased efforts on the part of the verification engineer, with time spent unnecessary on preparation/debugging of the testing environment which may even lead to an unreliable pass/failure verdict in the testing results.

Another problem is that currently the focus on testing is from the laboratory (LAB) administration's point of view. The LAB administration needs to keep track of the equipment which is owned and ensure effective utilization of this equipment (for power consumption saving and because this equipment is expensive). Bodega is the database providing the basis for the Resource Management System (RMS) used by the LAB administration. Although this RMS has addressed the problems of tracking and equipment utilization very adequately, there are still major opportunities for efficiency enhancements that this thesis project hopes to contribute to. Not all but many of the pieces of equipment need more structured and detailed information to assist not only the LAB administration to work effectively. Additionally, this data can also provide greater insight for the test teams enabling them to increase the automation of their test environment.

1.2 Context of Study

This thesis project is being carried out at Ericsson AB, Kista, Sweden in the MSV - RBS System verification department. In this Master's thesis the word "department" always refers to the "MSV – RBS System verification department". This department consists of two test teams: microCPP (Connectivity Packet Platform) and Node Common Infrastructure (NCI) system test (NCI-ST) team (NCI-1 and NCI-2). To be more specific this thesis project is carried out in the NCI I-2 test team. The NCI -test teams perform Black-Box testing of software (SW) (see section: 2.1.2 for details). The NCI team consists of a NCI function-test (NCI-FT) (see section: 2.1.1 for details) team and a NCI-Design team. Figure 1-1 shows a model of the interactions between these teams.

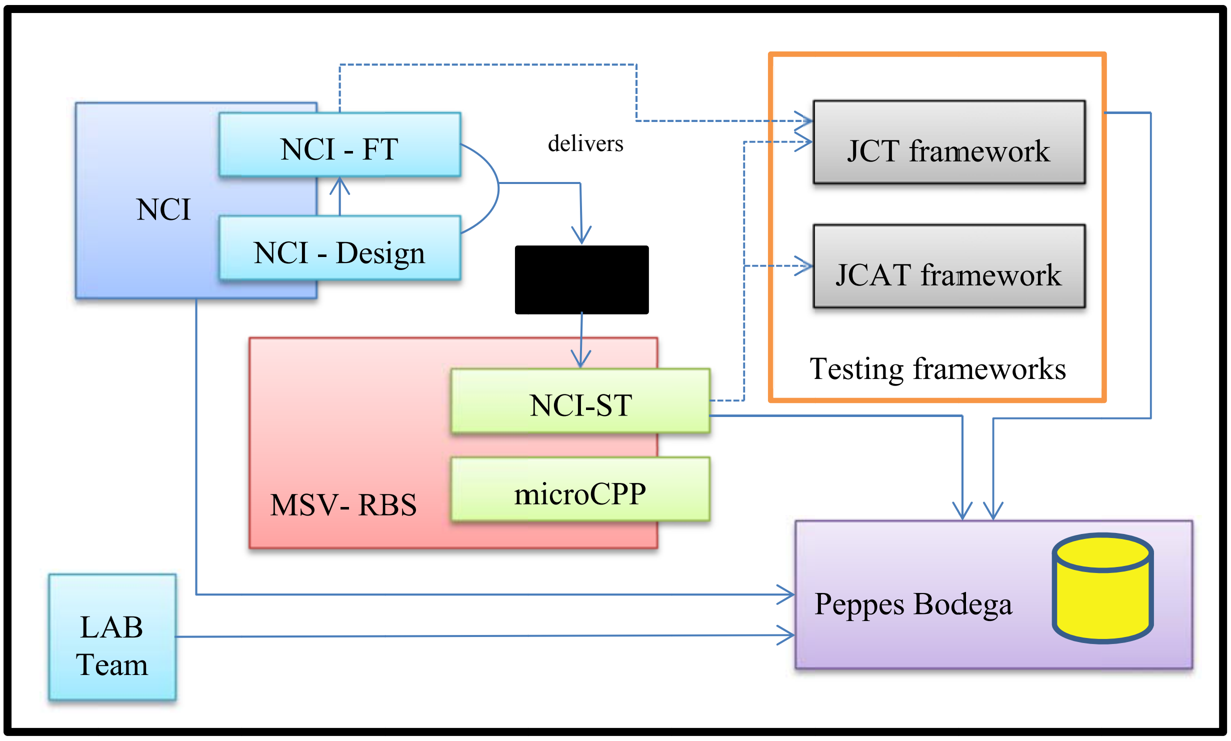


Figure 1-1: Work flow and Team interaction model

In the context of this Master’s Thesis project we will not focus on the hierarchy of teams in the department, but rather only on those details relevant to this Master’s Thesis project.

NCI can be considered as two cross-functional teams: NCI-FT and NCI-Design (see section 2.1 for details). NCI-Design develops certain features and they are initially tested at a function-test level by NCI-FT. The NCI-FT team uses Joint Common Test (JCT) as a test framework for testing implemented features. After testing at a functional level, these features are then delivered as a black-box to the NCI-ST team.

The NCI-ST team is currently using JCT as a test framework for testing implemented features, but on the side has started porting the test environment from JCT to Java Common Automated (JCAT). Both the JCT and JCAT test frameworks use the PeppesBodega-RMS (see Chapter 4 for details) for retrieving information about the SUT. PeppesBodega-RMS[*] is an internal tool used within the MSV department and has not been made publically available.

The LAB Team is responsible for the physical installation of devices in the lab. This team also uses the PeppesBodega RMS to store information about the devices. The information stored in the PeppesBodega RMS is then used by NCI-FT and NCI-ST teams for testing purposes either directly (to access device information) or indirectly (to execute test cases via JCT/JCAT framew).

[*] There is no relationship between Pepesbodega (http://www.pepesbodega.se/) and PeppesBodega-RMS. It is just a coincidence that their names are similar.

1.3 Motivation

As discussed earlier the time and effort spent during the testing process plays a key role in determining the quality of testing of the SUT. The ability to perform testing using different test equipment depends on both the location & availability of the test equipment & SUT; and the topology of the interconnection of this equipment that is needed to perform the desired tests on the SUT. The motivation for this thesis project is to provide a hassle-free environment to the MSV – RBS System verification department. Also, the results should ease the LAB setup process, both from an administrative point of view and for verification purposes.

The above discussion establishes the need for a topology aware RMS system which will contain comprehensive details about the SUT as administered by the LAB team.

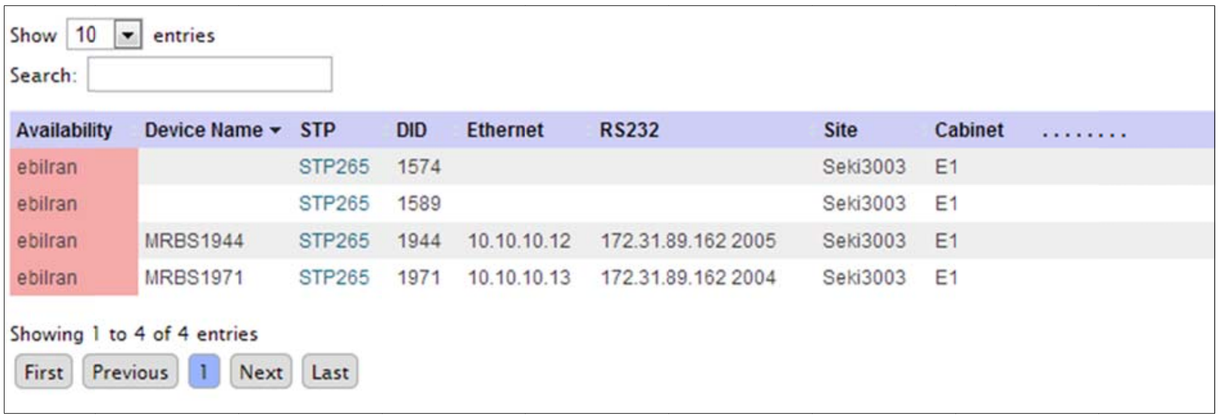
1.4 Objectives and Challenges

The main goal of this Master's Thesis project is to design, develop, and evaluate a topology aware RMS system that will provide efficient TECs and efficiently use & administer the LAB's equipment. The specific goals of this Master's thesis are:

- Design and implement a topological RMS that outputs a TEC for the SUT in a textual format (in the scope of this Master's thesis this format will be: JCT-config (Erlang) or JCAT-config (extensible markup language (XML)) and in a graphical format, such as portable network graphics (png);
- To provide a detailed view of all of the relevant LAB equipment for administrative purposes ; and
- To evaluate the tool that is produced.

In order to achieve these goals, the biggest challenge is expected to be gathering the requirements from all of the departments using the existing PeppesBodega-RMS. The remaining challenges include collecting data about the LAB equipment and their interconnections in order to specify their topology.

Figure 1-2 shows the current picture of devices booked in a System Test Plant (STP). The devices have two interfaces for performing action i.e. IP (referred to Ethernet in Figure 1-2) and Serial connection (referred to RS232 in Figure 1-2) connection. An IP connection has an IP address associated to it and used for sending commands to the device. The serial connection is a TCP connections and physical realized over an Ethernet connection to a switch with multiple ports; this interface is used for debugging purposes. The STP information shown in Figure 1-2 lacks any information about how the four devices (identified by Device Identifiers - DIDs) are connected to each other. This topological information is critical for the verification engineers for their test oriented tasks. For example: "*Checking the delay measurement*" for the connection between device-A and device-B requires information such as (1) type of connection between them, (2) length of interconnection (cable connecting the devices), and other delay calculation information which depends on topological information.


Show 10 entries

Search:

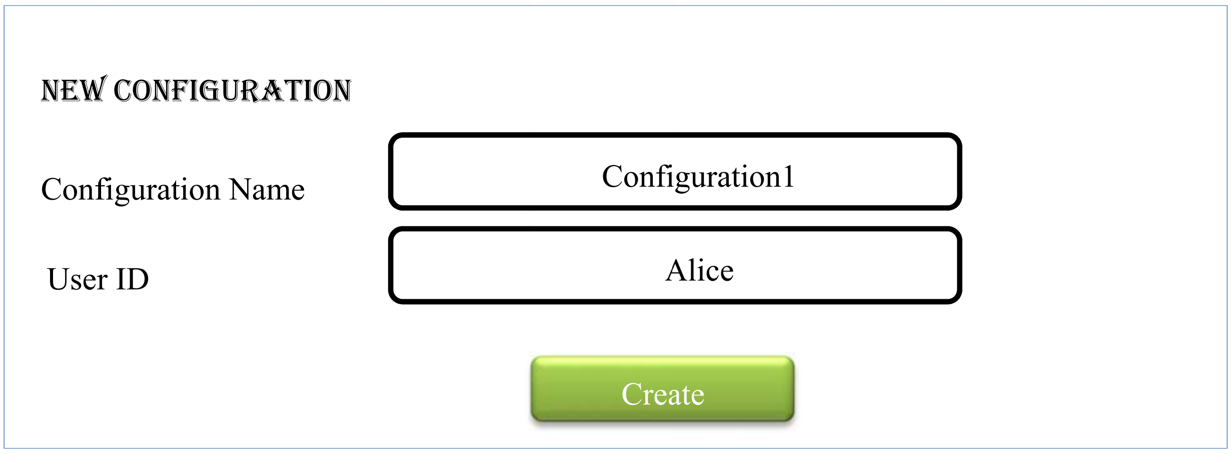
Availability	Device Name	STP	DID	Ethernet	RS232	Site	Cabinet	....
ebilran		STP265	1574			Seki3003	E1	
ebilran		STP265	1589			Seki3003	E1	
ebilran	MRBS1944	STP265	1944	10.10.10.12	172.31.89.162 2005	Seki3003	E1	
ebilran	MRBS1971	STP265	1971	10.10.10.13	172.31.89.162 2004	Seki3003	E1	

Showing 1 to 4 of 4 entries

First Previous 1 Next Last

Figure 1-2: STP view in PeppesBodega-RMS

As discussed above this topological information is very important when it comes to testing of interconnections or to setup connections between the devices. In context of this Master's thesis project *'topology'* and *'configuration'* will be used alternatively, as every test configuration corresponds to a specific interconnection topology of a selected set of devices. The aim of the project is to realize a user interface that will enable a user to create a test configuration by selecting the devices that are to be interconnected with the SUT (and each other). Figure 1-3 shows a proposed interface for creating an empty topology. In Figure 1-3 a new configuration '*Configuration1'* is created for user *'Alice'*.


NEW CONFIGURATION

Configuration Name: Configuration1

User ID: Alice

Figure 1-3: Create a new empty topological configuration

After creation of a new configuration, the devices included together with the SUT should be included from the main database in order to carry out a particular test oriented task. This interface should be capable of including a complete STP or a single standalone device from an existing STP. For example, consider the case where STP-1 contains device-1, device-2, and device-3; and STP-2 contains device-4, device-5, and device-6; then it should be possible to utilize all devices (device-1, device-2, and device-3) from STP-1 and any single device from STP-2 (device-4, device-5, or device-6). Figure 1-4 shows that all of the devices in STP-1 and device-4 (from STP-2) are being added to the '*Configuration-1'* for user *'Alice'* (in the configuration created in Figure 1-3).

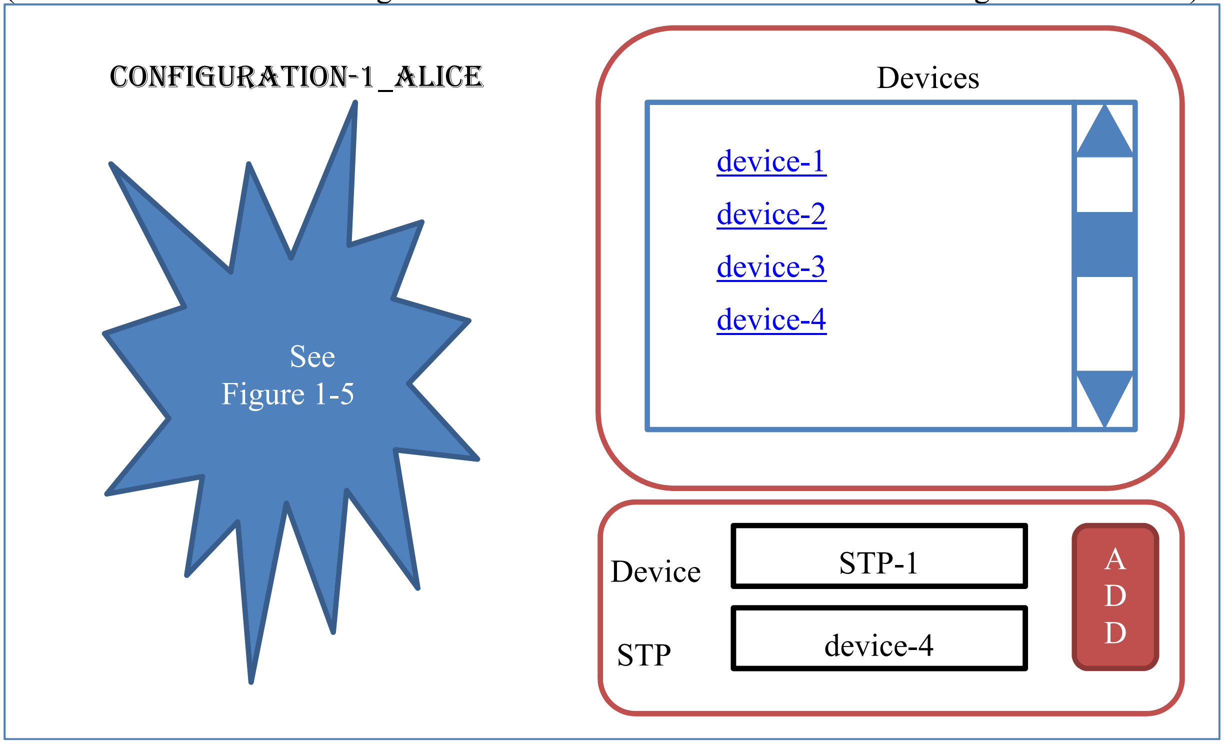


Figure 1-4: Populating a topological configuration

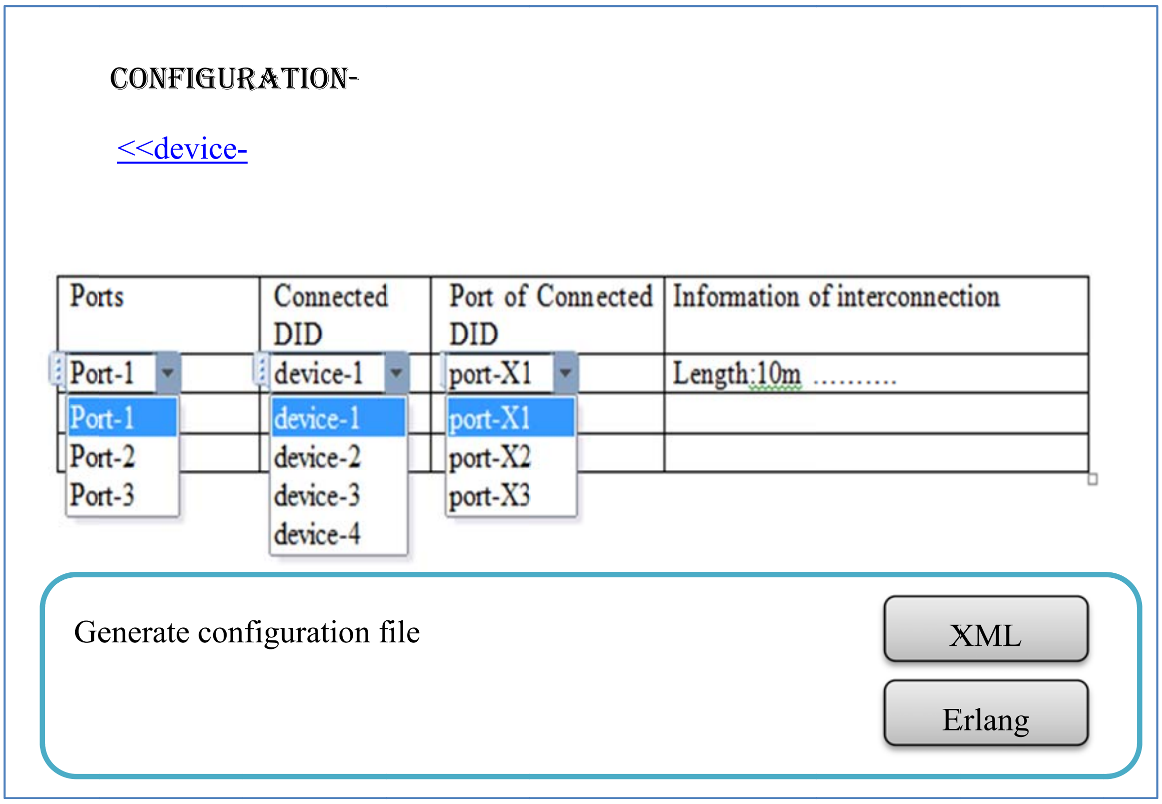
Figure 1-5 shows the formation of a configuration. The aim of this interface is to provide a user with the ability to define the interconnections between any two components. This basically involves the following five important items of information:

1. DID of first device
2. Port of first device that needs to be connected
3. DID of second device
4. Port of second device which is connected to first device
5. Information about this interconnection

It is also important to note that if device-1's port-1 is connected to device-2's port-1, then the information about this interconnection should be the same when seen from device-2's perspective. For this purpose the functionality of undirected graphs will suffice for the objectives of this Master's thesis (See details in Chapter 5).

As mentioned in section1.2 there are two frameworks being used for test automation: JCT (written in Erlang[*]) and JCAT (written in Java). JCT requires input in Erlang (cfg) format, whereas JCAT needs input in XML format. To cater to the needs of all of the teams it is important to provide the user with the ability to generate either an XML or Erlang configuration file.

[*] For further information about the Erlang programming language see: http://www.erlang.org/


CONFIGURATION-

<<device-

Ports	Connected DID	Port of Connected DID	Information of interconnection
Port-1	device-1	port-X1	Length:10m ....

Figure 1-5: Generation of a configuration file

1.5 Target Audience

This results of this thesis project will assist the verification engineers within the MSV – RBS System verification department (here after the "target" department), function test (FT) teams, and LAB administration team. The thesis should also provide additional insights for test architects (for example, enabling them to design better and more complete tests). Other verification departments might utilize the research, findings, and salient features of this project to assist them in efficiently performing testing and for administrative purposes.

1.6 Contributions

This Master's thesis contributes to an existing PeppesBodega RMS by modifying following the behavioral aspects of this RMS:

- Restructured the PeppesBodega RMS:
  - Store/retrieve topological information
  - Provide hassle-free input for generation of TECs
  - Enable efficient utilization of used/unused resources
  - Align TEC information for all teams using this RMS

This master' thesis also contributes to facilitating understanding of the basic concepts related to testing and to build a foundation for new verification engineers by providing domain specific knowledge for the target department.

The idiom "Use a picture. It's worth a thousand words" explains the importance of visualization in understanding complex concepts. There are two basic means for explaining any phenomenon, namely textual and graphical representation. Tools for analysis can either generate textual details or graphical visualizations. Unfortunately in many cases presenting information in only a textual format makes it quite difficult to digest the concept, while in other cases a graphical representation is insufficient for understanding the idea behind the concept. Moreover, having only a graphical representation may not provide the input necessary for other processes (in our case the need for a TEC). For these reasons this thesis project provided will utilize textual representation and emphasize the need for graphical representation.

1.7 Outline

The thesis is organized into six chapters. Following this introduction chapter, Chapter 2 provides the basic concepts and terminology of testing (providing a foundation for rest of the thesis), then reviews some of the relevant literature within the domain of RMS in general and a topology based RMS in particular. Architectural details of the existing PeppesBodega-RMS are given in Chapter 3. Chapter 4 explains the methodology adopted to achieve the goals stated in Section 1.4. Chapter 5 explains the design and gives implementation details of the proposed topology based RMS (here after referred to as the PeppesBodega-TF (Topological Framework)). Chapter 6 describes the testing (functional and non-functional) and evaluation of a prototype of the PeppesBodega-TF. Chapter 7 concludes the thesis and suggests future work.

2 Basic Concepts and Background Study

This chapter provides a comprehensive background concerning the two major subjects of this thesis: testing and resource management systems (RMSs). Testing has been further divided into two subsections, with Section 2.1 introducing the concepts that are critically important for readers who have no or limited background in testing; whereas section 2.2 focuses on explaining the terminology of the testing domain. Section 2.3 provides information to give the reader a basic understanding of approaches to better utilize resources when testing as well as how to reduce the overhead in doing so. Section 2.5 introduces the basics of a network topology resource management system. Finally, section 2.6 introduces the basics of formal grammars.

2.1 Basics of Testing

If you ask ten different verification engineers, it is quite possible that all of them define and interpret "testing" in their own way. However, with respect to software all of the definitions will highlight the belief that testing is used to find bugs in the software in order to assure better software quality. A concise and one of the most accurate definitions is provided by B.B. Agarwal, et al.:

> *"Testing is the major quality-control measure used during software development. Its basic function is to detect errors in the software. Thus, the goal of testing is to uncover requirement, design, and coding errors in the program" [1].*

Testing is generally performed in two organizational structures[2]:

1. An independent development team and a separate independent test team; or
2. A team consisting of testers and developers, i.e., a cross functional team.

The concept of cross-functional teams has been gaining attention within a large number of organizations over the past several decades[2]. We will not compare which organizational structure is better, as this is outside of the scope of this Master's thesis. However, for the purposes of this thesis project we need to be able to support both organizational structures.

2.1.1 Levels of Testing

Software testing depends upon the scope (or levels [3, 4]) and the time plan usually follows the categorized order (see Figure 2-1):

1. Unit Testing,
2. Integration Testing,
3. System Testing, and
4. Acceptance Testing.

Each of these levels of testing will be described further detail in the following paragraphs. These descriptions will follow the chronologic order generally used for testing; hence it will start with unit testing and end with acceptance testing.

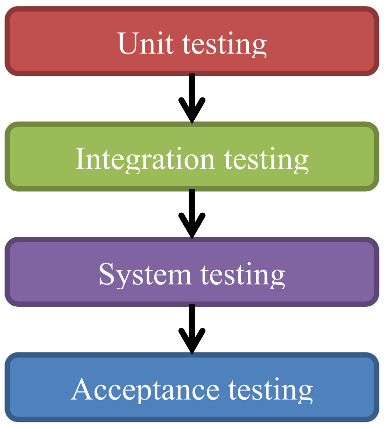


Figure 2-1: Testing levels [5, 6]

2.1.1.1 Unit Testing

Unit testing (component testing or function testing) is used to test isolated parts of modules, units, or modules of software. This type of testing is usually performed by the developer of each particular software unit.

2.1.1.2 Integration Testing

After unit testing, the isolated units that are intended to work together are grouped together and verification is performed on these groups (as built from these units). This is done to verify that the communication within a particular group of units works as expected.

2.1.1.3 System Testing

System testing is done after integration of a complete system has been performed. A system test is the first stage in which the system is tested against specific system requirements. Verification performed in this stage that does *not* require knowledge of the software's design is referred to as black box testing. *This thesis project is mainly being carried out within a department which focuses on this level of testing.*

2.1.1.4 Acceptance testing

Acceptance testing is an important test phase because this level of testing will analyze if the quality of the software of the whole system is sufficient to deliver the system to customers.

2.1.2 Types of testing techniques

Testing can be performed using either of following two techniques[5]:

1. Black-box testing or
2. White-box testing

In black box testing input (as a stimulus) is given to a system, and then the result is matched with the expected (required) output. If the result matches the expected output, then the test is said to be passed – otherwise the test is said to be failed. In white-box testing we are concerned with both the test result and whether the software worked properly or not. In the case of white-box testing we have access to the source code for the software and can analyze the test results in terms of execution paths through the code. In the target department, system testing is performed using black-box testing.

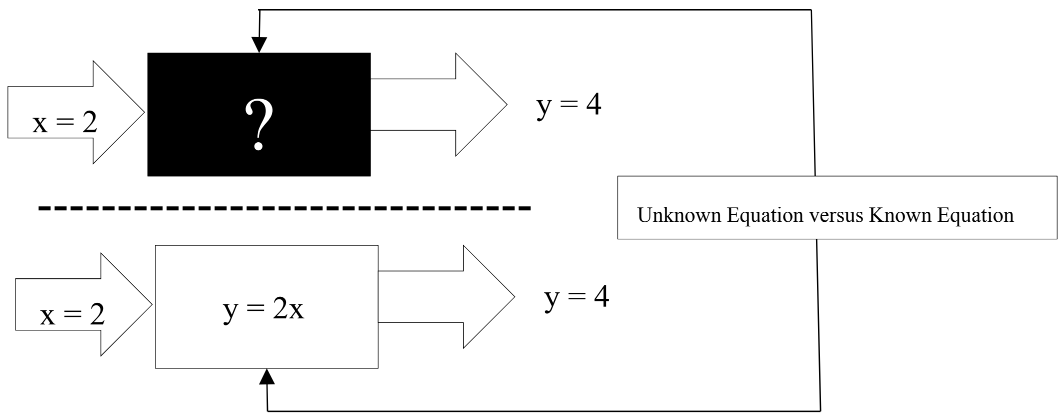


Figure 2-2: Black-box testing vs. White-box testing[5]

2.2 Taxonomy of Test development

This section will provide the reader with the testing terminologies generally used in automated test environments[6, 7]. We will describe desirable properties of the commonly used terminologies and their interpretation in the target department (if not explicitly specified otherwise). The most important terms are given along with a brief description in Table 2-1.

Table 2-1: Testing Terminology

Test Oracle	The test oracle provides a verdict for the failure or passing of the software application under test[8]. In the target department this is referred to as a "Requirement specification" by the system-design team and as a "Test Specification" by the test team.
Test case	A test case is the basic unit of an automated test suite. A particular test case is designed to produce an output which will be checked by a test oracle (requirement specification). Some of the desirable properties of test cases are: • Must have a single test oracle, • Created using a modular approach, • Designed in a structured way to facilitate its easy maintenance, and • Be well documented.
Test Suite	Test cases are combined together into a test suite targeting a common area of interest.
Test Specification	See explanation of Test Oracle.
Test Scope	The test scope defines the amount of testing performed to reach a test verdict which satisfies the pass criteria.
Test Coverage	For a given test case or test suite the degree of test coverage specifies the fraction of the complete set of features of a given SUT that are tested.
Legacy testing	A "regression test" is performed in order to see if a newly added feature causes errors in previously working release(s).

2.3 Resource Management System (RMS)

Resource management has always been an important part of any large organization and even small groups of people working together may need to perform resource management in order to efficiently achieve a goal of common interest. A number of software packages have been developed to automate and assist in resource allocation (one of the main objectives for creating and using a RMS)[9]. Horikiri et al. provided a generalized definition of an RMS:

> *"A resource management system, of the type wherein processes are applied to real resources, which are resources previously input into a computer system that performs information processing, to obtain new resources, includes a plurality of context maintaining units that respectively establish a correspondence with attributes".[10]*

In Section 1.4 we noted that a focal point of this Master's thesis project is the storage and retrieval of topological information concerning a SUT to or from a RMS. In terms of this topology formation RMSs can be categorized into two distinct classes: Self-forming (intelligent) systems and Lazy (unintelligent) systems.

To the best of my knowledge after carrying out a detailed literature review, these two classes have not been classified using these specific terms in any scientific literature. However, from a research perspective these categories have been prevalent under different names (see the discussion in Sections 2.3.1 and 2.3.2). The reason for this categorization in the context of this Master's thesis is provided in the following discussion of these systems.

2.3.1 Self-forming systems

Systems that consist entirely of intelligent[*] devices are called 'self-forming systems'. A significant body of research has been devoted to gathering resource information via different communication methods within a network of such devices[11–15]. All of these methods have their own advantages and disadvantages in terms of performance, capacity, completeness, etc. For example, Migas et al. attempted to finding route information, topology information, etc. with the help of static and mobile agents that crawl the network to obtain the information necessary for reconfiguration of an *ad hoc* network [14]. Although this method automatically obtains information available within the network, it injects extra traffic within the network. This extra traffic may not be problematic when bandwidth is not a scarce resource, but the method still has capacity implications.

As the devices registered in the PeppesBodega RMS are not intelligent, we will not further study this approach. However, in the future this approach might be relevant if the devices that are being registered are intelligent.

2.3.2 Lazy systems

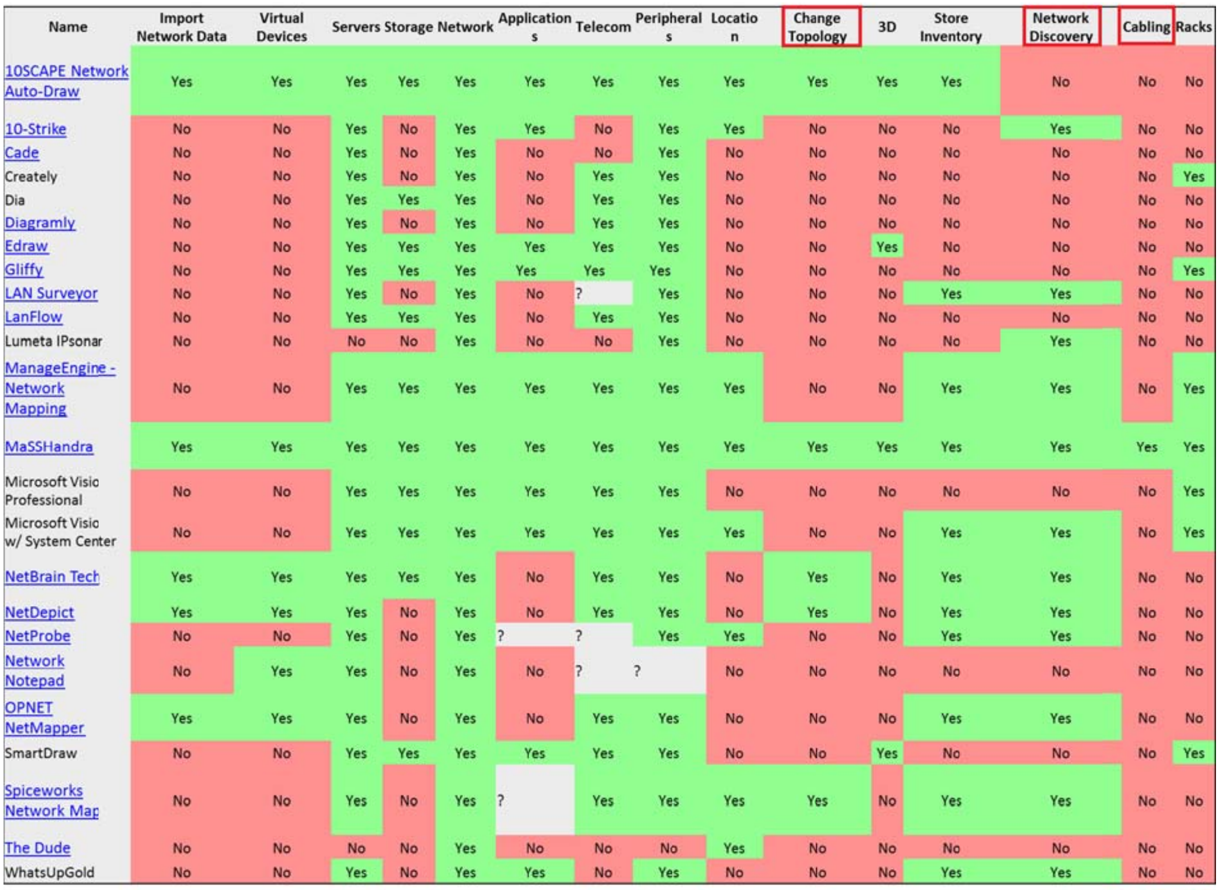
Systems that consist of at least one non-intelligent device are called "Lazy systems". The reason why they are called *lazy* is that they do not automatically provide any configuration information (for example, they do not provide any information concerning their

[*] **Intelligent devices** are devices that are able to sense their environment and can contribute to achieve a specific goal. For example, gathering information about the topology of a system.

interconnection topology). This domain is applicable to the devices considered in this Master's thesis project.

2.3.3 Network visualization software

A number of programs for network visualization have been developed, each targeting specific needs, such as for designing and generating different types of networks[16]. These programs can either be used by manually entering information to create diagrams (for lazy systems - section 2.3.2) or utilize automatic/semi-automatic (self-forming systems - section 2.3.1) approaches for creating diagrams by sensing their own environment. Figure 2-3 shows a detailed overview of some examples of such software.



Name	Import Network Data	Virtual Devices	Servers	Storage	Network	Application s	Telecom	Peripheral s	Locatio n	Change Topology	3D	Store Inventory	Network Discovery	Cabling	Racks
10SCAPE Network Auto-Draw	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No
10-Strike	No	No	Yes	No	Yes	Yes	No	Yes	Yes	No	No	No	Yes	No	No
Cade	No	No	Yes	No	Yes	No	No	Yes	No	No	No	No	No	No	No
Creately	No	No	Yes	No	Yes	No	Yes	Yes	No	No	No	No	No	No	Yes
Dia	No	No	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No	No
Diagramly	No	No	Yes	No	Yes	No	Yes	Yes	No	No	No	No	No	No	No
Edraw	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No	No
Gliffy	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	Yes
LAN Surveyor	No	No	Yes	No	Yes	No	?	Yes	No	No	No	Yes	Yes	No	No
LanFlow	No	No	Yes	Yes	Yes	No	Yes	Yes	No	No	No	No	No	No	No
Lumeta IPsonar	No	No	No	No	Yes	No	No	Yes	No	No	No	No	Yes	No	No
ManageEngine - Network Mapping	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes
MaSSHandra	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Microsoft Visio Professional	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	No	No	No	No	Yes
Microsoft Visio w/ System Center	No	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes	No	Yes
NetBrain Tech	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	No	No
NetDepict	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	No	No
NetProbe	No	No	Yes	No	Yes	?	?	Yes	Yes	No	No	Yes	Yes	No	No
Network Notepad	No	Yes	Yes	No	Yes	No	?	?	No	No	No	No	No	No	No
OPNET NetMapper	Yes	Yes	Yes	No	Yes	No	Yes	Yes	No	No	No	Yes	Yes	No	No
SmartDraw	No	No	Yes	Yes	Yes	Yes	Yes	Yes	No	No	Yes	No	No	No	Yes
Spiceworks Network Map	No	No	Yes	No	Yes	?	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No
The Dude	No	No	No	No	Yes	No	No	No	Yes	No	No	No	No	No	No
WhatsUpGold	No	No	Yes	No	Yes	Yes	No	Yes	No	No	No	Yes	Yes	No	No

Figure 2-3: Network Visualization Software Adopted from [16]

In Figure 2-3 the cells marked with red rectangles are applicable to a tool which meets the goals (as stated in section 1.4) of this Master's thesis. *Network Discovery*, in the scope of this Master's thesis refers to application marked "*No*" in this table. The column marked "*Change Topology*" refers to providing the users of the application with the option to change the topology of the SUT. The column marked "*Cabling*" is also important in the domain of topological information, in the target department this concerns information about cable connections (in terms of their length, delay, and type).

2.4 Common Public Radio Interface (CPRI™)

In order to facilitate the interconnection of radio equipment a number of vendors (including Ericsson, Huawei, NEC, NSN and Alcatel-Lucent) have defined a common scheme for specifying the internal interfaces associated with a radio base station.

The CPRI specification is defined for the communication between Radio Equipment control (REC) and Radio Equipment (RE). The specification of CPRI covers layer 1 and 2 of OSI model. The layer 1 (physical layer) supports communication both over electrical (local radio units) and optical cable (remote radio units). Layer 2 (Data link layer) supports flexibility and scalability. This standardization has provided a platform to cross-use the products from different vendors. We will not provide details of this protocol as this is not intrinsic ally of interest for this Master's thesis. The readers who are interested can find detailed information about its history, specification and ongoing activities at http://www.cpri.info/index.html. The CPRI specification is relevant in this thesis only because we will refer to CPRI ports (see for example section 5.2.1.3)

2.5 Network topology management system

The schematic description of a network of nodes and their interconnections is referred to as a "network topology"[17]. A network topology can be categorized into two sub-categories based on a geometrical view: a physical topology and a logical topology.

A physical topology refers to the physical placement of nodes and their interconnections (via cables, fiber, etc.). In computer networks, physical topology refers to the physical layout, i.e., the locations of the computer and the cabling between the computers.

In contrast, a logical (signaling) topology refers to the path followed by a signal from one node to another node in the network. The logical topology mostly is the same as the physical topology. However, in some cases software can introduce differences between the logical and physical topology. In other cases the hardware within the nodes are responsible for the mismatch between the logical and physical topologies. This (difference between the logical and physical topologies) should *not* be taken as a fault in the network because logical topologies are typically generated for a specific purpose.

Figure 2-4 presents a scenario where the logical topology is different from physical topology. In the physical topology (shown on the left side of Figure 2-4) the computers are connected to a central hub. The physical topology only indicates that the data packet will be sent to all the other computers. However, if we want to know if "the data packet will be sent simultaneously out all of the ports" or "will travel around a ring and be consecutively sent to all the ports", we have to look inside the hub (i.e., we must know **how** the input and output ports are interconnected). This information which tells us the signaling path, in this case the path a data packet will traverse, is referred as a logical topology.

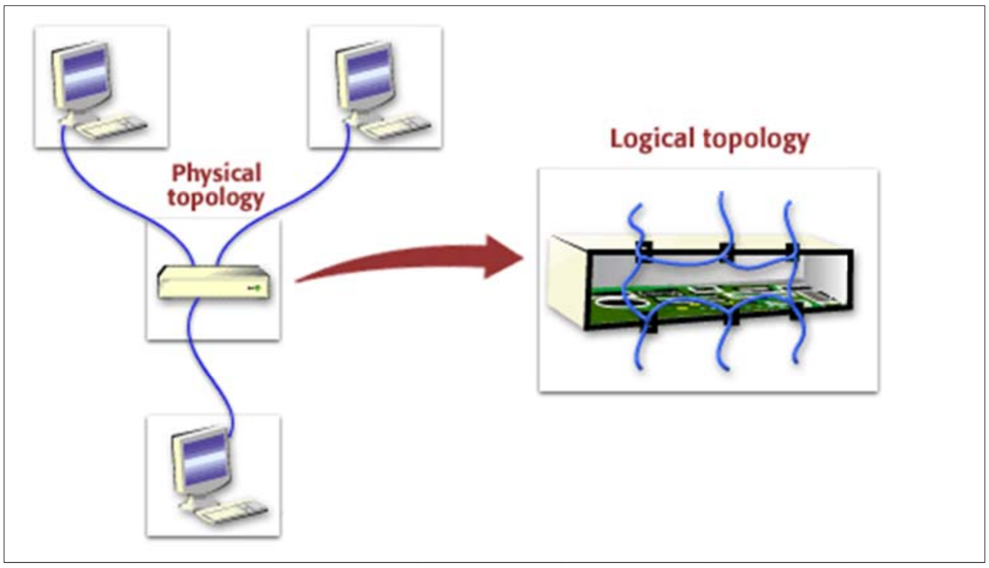


Figure 2-4: Logical versus Physical topology [18]

2.5.1 Applications of Topological Resource Management Systems

Topology resource management systems have been utilized in different fields, each with their own domain requirements. The following two subsections describe two application scenarios, where we see gathering of information about the topology of the resources in the network in order to achieve a specific goal.

2.5.1.1 Conservation of energy in a sensor network

Schurger et al. [19] utilized the concept of topological resource management to conserve energy in a sensor network. In their approach, the sensor nodes' activity is tuned to maximize sleep mode and the sensor nodes only enter wake mode when data needs to be forwarded to neighboring nodes. Topological information is needed to plan the wakeup and sleep transitions of the sensor nodes.

2.5.1.2 Efficient utilization of resources to increase network lifetime

Pan et al. [20] also utilized information about a system's topology to increase a network's operating lifetime in the case of battery powered wireless sensor network nodes. In their approach, a two-tiered wireless sensor network consists of many base stations and many sensor nodes. The sensor nodes were physically placed in clusters around base stations. The responsibility of each sensor node is to capture, encode, and transmit sensor data (from the specific area where it resides) in a raw format to an application node. The application node then creates a local view of the area and sends this to the base station in a composite bit-stream format. Passing topology information to the base stations helps the base stations to better manage the network; thus increasing the network's operating lifetime.

In both of the above scenarios, we see that both of the approaches to gathering information about the topology of resources in the network can be used to achieve a system specific goal.

2.5.2 Network topology management system through a database of managed network resources including logical topologies

Kulkarni et al. [21] presented a RMS for computer networks along with specific methods for maintaining complex relationships in this network of computer elements. This architecture used a simple database to store node information, type information, and view data. The views are specific to the context of each node's information. For example, adding or removing a parent of a child node will change the views of both nodes.

This system was specifically designed for computer networks for management and control purposes. The system was designed to provide the capability for visualization of computer networks. As mentioned above, there are two types of topologies (logical and physical). This system was capable of meeting the needs for maintenance of both physical and logical topologies in the database by applying a new data model. Physical and logical databases were stored in separate Management Information Server (MIS) databases. Users of this system were restricted to accessing the data through the database containing the physical topology. Consistency is maintained via a consistency application that is present in both (physical and logical) databases.

2.5.2.1 Salient Features of a Network RMS

Over the past few decades there has been an enormous expansion in computer networks and at the same time these networks are becoming more and more complex. The invention by Kulkarni et al. still holds an important position as it was not specific to a particular computer network. Their invention is still applicable for most network domains because it generalizes the services that a RMS could provide. The following are the essential operations that network administrators (i.e., management users) usually require:

Monitor	Monitoring the network is important for calculating performance and capacity metrics. How efficiently resources within the system are utilized can be evaluated. Unusual behavior of resources can also be detected by statistical methods appropriate for the specific domain of the system under consideration.
Manage	Monitoring of resources creates an opportunity for administrators to find an alternative way to utilize these resources in order to increase the system's efficiency.
Control	The control part of the RMS ensures that the managed resources follow a specified behavior.

2.5.2.2 Components of a topological managed RMS

The work of Kulkarni et al. introduced important components in the topological managed RMS:

1. Plurality of nodes in the network
2. Plurality of interconnection among these nodes
3. A management system consisting of managed network resources stored in a database
4. Database of managed network resources including:
    a. Definition of "Network nodes"
    b. "Node types" associated with network nodes
    c. "Node view" associated with network nodes

5. Plurality of network administrators

2.5.2.3 Specifications of a topological managed RMS

Apart from inheriting salient features of a network RMS (see section 2.5.2.1), the work of Kulkarni et al. also provided following features that result from cross-communication among the components (see section 2.5.2.2):

1. Network administrators should have the option to modify the database with information about the managed network;
2. Network administrators should be able to visualize the "Node view" as extracted from the database of the managed network; and
3. The "Node view" should be updated when node attributes change. For example, if a parent is added to the attribute of a node, then this parent relationship can be used to create a new "Node view".

2.6 Grammar

In any programming language, we have a set of rules to write expressions in the language, which will then be translated by the compiler into machine language instructions to perform the requested operations.

The set of rules in formal language theory is known as a 'grammar'. A grammar G can be defined as tuple of four items: G = {N, T, P, S} where N = Finite set of non-terminals, T = Finite set of terminals, P = Finite set of production rules, and S = a start symbol.

Chomsky [22] described a hierarchy of grammars with four classes of formal grammars:

1) Recursively enumerable grammar (Type-0),
2) Context-sensitive grammar (Type-1),
3) Context-free grammar (Type-2), and
4) Regular grammar (Type-3).

Type-1, Type-2, and Type-3 grammars are differentiated by the way the production rules are setup for them. An explanation of these production rules are provided in their respective subsections below. We will not go into the details of these grammars, as it requires quite a detailed explanation of all the concepts involved. However Chomsky[22] and Natarajan [23] explain automata theory and relevant topics in detailed.

2.6.1 Recursively enumerable grammar (Type-0)

In this grammar, there exists a Turing machine, for which production rules are defined such that the machine will enumerate all possible words from the alphabets of the language.

2.6.2 Context-sensitive grammar (Type-1)

A context-sensitive grammar or type-1 grammar has production rules of form:

$$\alpha A\beta \rightarrow \alpha\gamma\beta$$

Where A is a non-terminal and α, β, γ are the strings from the set of terminals and non-terminals. Also the following restrictions are made:

α ⟶ empty or non-empty

β ⟶ empty or non-empty

γ ⟶ must be non-empty

2.6.3 Context-free grammar (Type-2)

Context-free grammars form the theoretical foundation for most programming languages, even though the syntax is restricted to context-sensitive name resolution for declaration and scope of code. Type-2 grammars have production rules of form:

$$A \rightarrow \gamma$$

Where A is a non-terminal and γ is string from the set of terminal and non-terminals. γ can be empty or non-empty. A parser is often utilized for the subset of type-2 grammars for easy parsing. LL parser[24] is an example of a parser which utilizes the subset of context-free grammars.

We have utilized this approach in the implementation of the solution proposed in this Master's thesis. The reason for choosing this approach is that we needed a flexible approach that can be easy maintained in the future despite increase in the number and types of devices in the LAB.

2.6.4 Regular grammar (Type-3)

Regular grammar can be implemented in two ways: right-regular or left-regular grammars.

Right-regular In this case the regular grammar is restricted to having a single non-terminal on left side and the right side consists only of a single terminal; which can be followed by single non-terminal in the case of a right-regular grammar.

Left-regular In this case the regular grammar has a single terminal on right side and can possibility be preceded by a single non-terminal.

This grammar classification is extensively applied to regular expressions. Regular expressions are frequently used for specifying searching patterns and defining the lexical structure of programming languages – both are applications of regular grammars.

3 Methodology

The scientific research methodology used in this thesis project is based on a set of analytical techniques and perspectives (or logically formulated steps) to investigate a given phenomenon, to acquire new knowledge, and to correct & integrate existing knowledge.

The goal of this Master's thesis project is to design, develop, and evaluate a topology aware RMS system that will address the need to efficiently generate a TEC and enable efficient administration of the LAB's equipment. Therefore, this thesis has adopted the Design Science Research (DSR) approach, because DSR involves designing novel artifacts to analyze and understand the behavior of given aspects of information systems[25].

In addition to positivist and interpretive perspectives, DSR is considered is yet another set of analytical techniques or perspectives for performing research in information systems[26]. Henver et al. describe the process of DSR as "Design science ... creates and evaluates IT artifacts intended to solve identified organizational problems"[27]. Wherein the *artifacts* are defined as "innovations that define the ideas, practices, technical capabilities, and products through which the analysis, design, implementation, management, and use of information systems can be effectively and efficiently accomplished"[27].

We will create new artifacts to extend the PeppesBodega RMS in order to achieve the goals stated in Section 1.4. The underlying process of this Master'sthesis is derived from the general methodology of DSR as shown in Figure 3-1.

The process (shown in Figure 3-1) begins with an *Awareness of problem*. The knowledge about the problem space is acquired during this phase, and the scope of the problem area is delimited during this phase. The *suggestion/solution* phase follows immediately after the *Proposal* (i.e., an output of the Awareness of problem phase). A *tentative design* is formulated after building knowledge concerning the problem space. The *tentative design* is implemented during a development phase. However, the techniques for implementation vary depending on the design specification of the artifact(s) to be constructed. Once constructed, the artifact is evaluated during the *Evaluation* phase according to the *Proposal*; deviations from expectations (both quantitative/qualitative) are carefully noted and tentatively explained. The *Conclusion* phase is considered as the finale of a specific research effort. The results obtained, knowledge gained, and the facts learned during whole process are the outputs of this phase.

In following subsections, we will give an overview of the methodology to be utilized in solving the problem stated earlier in this thesis (in the scope of design science research).

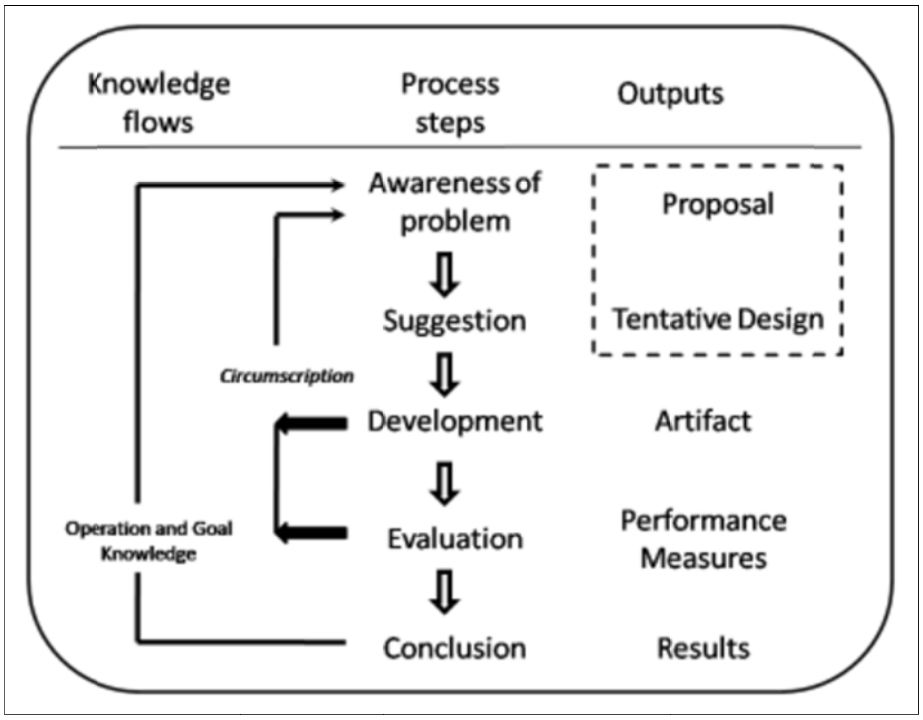


Figure 3-1: Research methodology of Design Science Research (Adapted from [27])

3.1 Awareness of problem

"Problem space is defined by the environment for which the phenomenon of interest lies" [27].

In this Master's thesis, our environment of interest is a topological resource management system. As discussed in section 1.1, there is a long lead time before new verification engineers can start actual testing and there can also be a mismatch of the group of devices utilized for testing (here after referred to as a STP) causing frequent interruptions in testing in order to reconfigure the testing environment before continuing testing. From the LAB administration's view, not all but most of the equipment needs more structured and detailed information in order to enable efficient utilization of this equipment and to enable efficient testing.

3.2 Suggestion / solution

"The objective of design-science research is to develop technology-based solutions to important and relevant business problems"[27].

The solution to the problem that this Master's thesis project will address is the design, development, and evaluation of a topology aware RMS system that will address the needs for efficiently generating TECs and will enable well managed administration of the LAB's equipment. Apart from this functional requirement, we have important non-functional requirements, for example to gather requirements from all the departments using the existing PeppesBodega RMS. Other non-functional requirements include collecting data concerning the LAB's equipment and their potential interconnections in order to manually extract topology information[*]. The evaluation (see section 3.4) is part of our non-functional requirements.

In this Master's thesis project, the current web framework (PeppesBodega RMS) will be extended to provide topology information in textual and graphical representations. This information will be presented in XML [28] for the textual output format and as png [29] for the graphical output format. The textual information in XML format will be used by two test frameworks JCT and JCAT as input for their TECs. The visual representation will serve two purposes: (1) so that an experienced user can easily debug configuration problems and (2) to lower the learning curve for new verification engineers.

3.3 Development

We followed an iterative-waterfall model (as shown in Figure 3-2) to implement the suggested tentative design. The functional and non-functional requirements of the problem space (as explained in section 3.1) are elaborated during the requirements elicitation phase. Next the design specifications (as explained in section 3.2) are proposed and implemented to fulfill these requirements. As a result, a functional prototype is developed which is an extension of the prototype developed in previous phases(s). Finally, tests (see Chapter 6 for details) are performed to examine if there is any gap between the specifications and this prototype. This is an iterative model, thus in every iteration the development underwent a similar cycle of requirement elicitation, modification of the design, implementation of this design, and testing of the prototype.

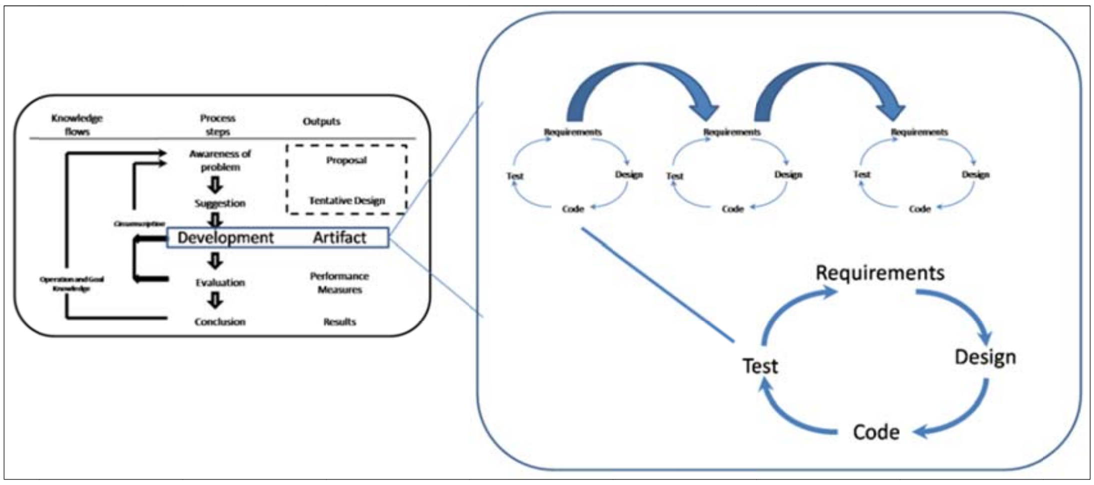


Figure 3-2: The development model followed to construct the artifact(s)

3.4 Evaluation

The evaluation of the artifacts developed as constructed during development phase is performed during *Evaluation* phase. These artifacts are evaluated on the basis of following

[*] As noted in section 2.3.2 the current equipment does not directly provide information about the device and its potential interconnections, hence this information must be manually extracted and entered into the RMS.

criteria: reliability and comparative analysis. Chapter 6 of this report provides the details of the evaluation of the artifacts constructed during this Master's thesis project.

3.5 Conclusion

The *conclusion* phase is the finale of this (Master's thesis project's) research effort. Chapter 6 and Chapter 7 presents the results obtained, the knowledge acquired, and the facts learned during this Master's thesis project.

4 “PeppesBodega” RMS

As discussed in Section 2.3, a RMS has an important position in the effective utilization of resources in terms of both cost and time. When it comes to system level testing, it is nearly impossible to avoid using a RMS to track information for the different SUTs used by different verification engineers.

The target department felt there was a need for a new RMS system, but planned the transition to this new RMS in two distinct phases:

“Old-PB-System”	This RMS was developed as a result of the initial needs of the LAB team to manage device information, which was previously done using a piece of paper or a simple text file (containing device information). This manual method resulted in a non-trivial task to perform on a daily basis.
“PeppesBodega”	This new RMS replaced the “Old-PB-System” and was introduced as a requirement by Micheal_Thomsson[*]. He provided a detailed description of the requirements for the PeppesBodega RMS.

4.1 Old-PB-System

The “Old-PB-System” was a simple RMS system that consisted of:

- a single HTML file that realized a graphical interface for its users,
- a text file that served as a database, and
- two Linux commands (sed[†] and awk[‡]) were used to perform data manipulation of the information about devices in database.

Figure 4-1 shows the architecture of the Old-PB-System.

[*] **Micheal D. Thomsson**, Project Manager for DU I&V Department at Ericsson AB

[†] **sed**, a stream editor, http://www.gnu.org/software/sed/manual/sed.html

[‡] **awk**, pattern scanner/processor, http://www.gnu.org/software/gawk/manual/gawk.html

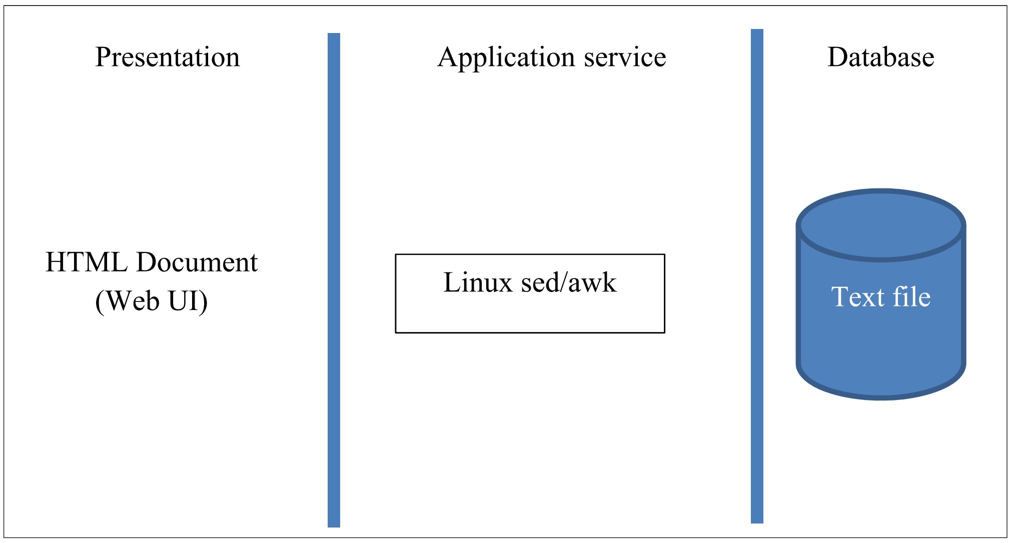


Figure 4-1: Old-PB-System Architecture

4.1.1 Information Bank

In this design a single HTML file was used to store the device data. The device data was limited to the following information: Name of the device, IP address of the device, and Serial number of the device.

To load the data into the HTML page the standard Linux sed and awk commands were used. This design supported only two needs of the LAB team: (1) tracking all the devices in the lab and (2) manual allocation of IP address to each device from an IP address pool.

4.1.2 Limitations

The design of the Old-PB-System had quite a lot of problems and included the following limitations:

- *Insufficient information for design and test teams*

  Verification engineers and software developers were unable to see the detailed information about the devices in order to see if they suited task specific requirements.

- *No time-limitation for booking*

  Once a device was booked, it was booked forever. Even if the person who made the booking would be away for a long period of time, no-one else could utilize this device. Due to this a series of problem occurred. Initially there was an increase in the number of unusable devices, and then new devices were ordered even though an existing device could have been reused. From the LAB's team point of view, this was a complete disaster from both space and cost perspectives.

- *Slow response time*

  The load time for several hundred devices on the webpage took approximately 40 seconds. According to Jakob Nielsen [*], a web usability consultant, no more than 10 seconds is acceptable for a website's visitor to retain his/her interest[30].

4.2 PeppesBodega-RMS

PeppesBodega-RMS is the current RMS. It is used extensively by both the design and test teams at the target department. It is a robust, scalable, and easily maintainable solution which addressed the limitations of the previous system (details of how it did this will be explained in subsequent subsections). Currently this RMS is being extensively used by the LAB team (see section 1.2) for administrative purposes and also by the design and test teams for task oriented purposes.

4.2.1 Development

PeppesBodega-RMS was developed in Erlang[†]. OTP[‡] was generally used in development of its web-framework. beard[32] is one of the essential components that was created in order to provide dynamic content selection via the web-interfaces.

4.2.2 Stake-holders

PeppesBodega-RMS provides services to three different graphical regions: Sweden, China, and Croatia. All three regions have their own local administrators and users.

4.2.3 Access rights

Users of PeppesBodega-RMS can be divided into two user groups based upon the actions performed on certain content in the database:

- *Administrator group*

  This group is further divided into two subgroups

Local Administrator	A local administrator has the relevant permissions to modify the information concerning devices/STPs local to a particular region. For example, the local administrator for Sweden can only modify device/STP information for the LAB's devices located in Sweden
Super Administrator	A super administrator has the relevant permissions to modify information about devices/STP globally (i.e., in any region). A super administrator can modify the view content for all devices/STPs. These changes

[*] **Jakob Nielsen**, http://www.nngroup.com/people/jakob-nielsen/

[†] **Erlang,** http://www.erlang.org

[‡] **OTP,** "OTP is set of Erlang libraries and design principles providing middle-ware to develop these systems".[31]

depend upon a change request from local administrators. A moderator is responsible for maintenance of the PeppesBodega-RMS which includes troubleshooting of malfunctioning features (see section 4.2.5 for details), updating the framework with updates of the development tools (see section 4.2.1 for details) being used, development of new requested features, etc.

- *User group*

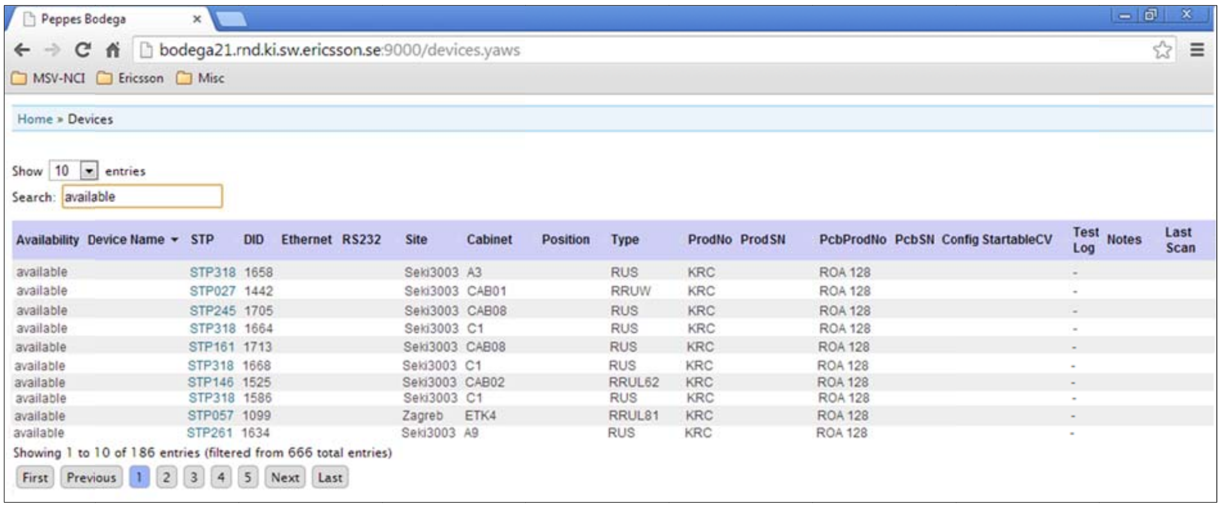
  Members of this group have no permissions to modification any device/STP. The only actions they can perform are to book or cancel their booking of a STP.

4.2.4 Web based views

These are the two major views that display information via the web interface: device view and STP view. These are further explained in the following paragraphs.

4.2.4.1 Device view

A 'device' is an atomic item in the PeppesBodega-RMS database. The device view is used to display a list of devices and their associated information. In the device view, the free-text search box is used to search for devices which match a given string. By default only the first ten entries are shown, however from a drop-down option the user is able to see all the devices that match the search criteria. Figure 4-2 shows a snapshot of the device view.



Show 10 entries
Search: available

Availability	Device Name	STP	DID	Ethernet	RS232	Site	Cabinet	Position	Type	ProdNo	ProdSN	PcbProdNo	PcbSN	Config	StartableCV	Test Log	Notes	Last Scan
available		STP318	1658			Seki3003	A3		RUS	KRC		ROA 128				-		
available		STP027	1442			Seki3003	CAB01		RRUW	KRC		ROA 128				-		
available		STP245	1705			Seki3003	CAB08		RUS	KRC		ROA 128				-		
available		STP318	1664			Seki3003	C1		RUS	KRC		ROA 128				-		
available		STP161	1713			Seki3003	CAB08		RUS	KRC		ROA 128				-		
available		STP318	1668			Seki3003	C1		RUS	KRC		ROA 128				-		
available		STP146	1525			Seki3003	CAB02		RRUL62	KRC		ROA 128				-		
available		STP318	1586			Seki3003	C1		RUS	KRC		ROA 128				-		
available		STP057	1099			Zagreb	ETK4		RRUL81	KRC		ROA 128				-		
available		STP261	1634			Seki3003	A9		RUS	KRC		ROA 128				-		

Showing 1 to 10 of 186 entries (filtered from 666 total entries)

First Previous 1 2 3 4 5 Next Last

Figure 4-2: Device view in PeppesBodega-RMS

In Figure 4-2, the search criteria 'available' will display all the devices that are currently *not* booked by any user. Search criteria can be used to find available devices of a particular type. Additionally, different search criteria can be combined with a space character in between them and each of different search criteria will have AND* boolean logic.

* AND, http://www.exploratorium.edu/lc/search/boolean.html , Last visited : 2013-12-23

4.2.4.2 STP view

One or more devices are grouped together to form an ‘STP’. The resulting STP is bookable by any user. The concept of a STP was introduced so that device(s) can be booked by a particular user for a specific purpose. An STP consists of the following four important components:

- List of devices,
- Booking purpose,
- Booking time (period), and
- Booking user.

The ‘List of devices’ refers to a subset of all the devices that will be reserved for the ‘Booking User’ who has booked the STP for a specific ‘Booking purpose’ for a specified ‘Booking time’ (period). The default booking time (period) is two weeks. The ‘Booking time’ (period) ensures that the ‘Booking User’ has control of these devices for the specified time frame. After this time period expires the status of these devices automatically returns to the “Available” state.

4.2.5 Salient Features of PeppesBodega-RMS

PeppesBodega-RMS is being used extensively in the NCI and MSV-RBS-NCI departments (See section 1.2) for administration of the devices maintained by the LAB team. These departments have both common and domain specific tasks which need to be performed. For example, NCI uses the information about devices to perform function tests, whereas MSV-RBS-NCI utilizes the same information to perform system tests. However, the configuration files needed by NCI and MSV-RBS-NCI are different and contain domain specific information. The PeppesBodega-RMS serves as the backbone of the testing processes in these departments.

The PeppesBodega-RMS has the following salient features (each of which will be described further in the following paragraphs):

1. Device tracking,
2. Concept of STP,
3. Device booking,
4. IP address pool management,
5. Storage Inventory,
6. Database backup,
7. Email subscription,
8. Naming Convention,
9. Lab-Scan System, and
10. Requirement-Request portal.

4.2.5.1 Device tracking

Within the PeppesBodega-RMS, each device is uniquely identified by a unique device identifier (DID).

4.2.5.2 Concept of STP

A STP is the smallest bookable unit for the PeppesBodega-RMS. A STP must contain at least one device. For a device to be bookable by a user it must be part of a STP and then this STP can be booked. It is important to note that the set of devices in a STP is typically very stable, since these devices are used together to conduct a specific type of test.

4.2.5.3 Device booking

As mentioned above devices cannot be booked until the device is part of an STP. Device booking is done in terms of a STP for the following reasons:

- The SUT for a specific purpose usually consists of more than one device, i.e. it is easier to book a single STP for testing a SUT rather than separately booking several devices. Consider a SUT which requires device$_1$, device$_2$, and device$_3$, it is more convenient to book one STP rather than separately booking these three devices.
- Managing a STP is far easier than managing individual devices. If you need another device to test the SUT, then you simply add the device to the STP. Once the testing task is finished, then all of the devices in the STP can be made available by simply unbooking the STP. For a similar test the STP can be rebooked by another person.
- Each STP has associated with it a set of comments that can be used to identify the purpose for which this STP can be used.

4.2.5.4 IP address pool management

In a large network, such as Ericsson's internal network, efficient utilization of IP address space[*] is important. Due to the consumption of a large number of IP addresses by the many devices within the LAB, it was very important to automatically track the IP addresses allocated to devices and free up unused IP addresses.

Public IP address space is being used for the LAB's devices rather utilizing a private IP address space because it is preferred to keep the LAB environment as close to the customers' environment as possible; for example, for vulnerability tests. Additionally, when the SUT is installed it will generally be in an environment with statically assigned IP addresses, rather than a networked environment with a dynamic host configuration protocol server; hence it is better to test the devices with such a configuration.

4.2.5.5 Storage Inventory

Devices in a storage area are part of a reserved STP (specifically STP-003). This STP is non-bookable and it is not visible to members of the 'User Group', rather it is only visible to the 'Administrator Group'. Within the 'Administrator group', the local administrators have their own reserved STPs and these STPs do not overlap with each other.

4.2.5.6 Database backup

A backup of the database is taken at regular intervals. Thus if the database is corrupted or unavailable for any reason, the database can be recovered from the backup. Additionally, there are cases when it is necessary to know what device information was available on a particular date in the past.

The database backup is stored only on a single disk, at least from the moderator's perspective. But internally within the network team, the separate replicas are maintained and data is safely preserved even in the case of a failure of a primary disk.

The shortcoming of this database backup approach is that there is no mechanism which can cater for the scenario when information was entered in the database since the last backup

[*] Specifically addresses from the IPv4 address space.

and then the database is corrupted. In this case the information lost cannot be recovered by any known means in the currently implemented system, i.e., there is no journaling or log file to keep track of changes to the database.

4.2.5.7 Email subscription

An email subscription is available to a 'Booking user'. This user will be sent email twice before a booking expires for a particular STP. The first email is sent three days prior to the expiration of the booking period and a second email is sent on the last day of booking period. In this way the booking user is notified that if no actions are performed, the booking will expire. Before the booking expires the user can rebook the STP for a new time period.

4.2.5.8 Naming Convention

For effective communication to take place among different departments concerning a particular product, the naming convention should be consistent. The PeppesBodega-RMS defines naming conventions for all the devices. This information is used during the process of selecting and installing software on a particular type of device. A department can search in the device view to find information about all of the devices of a specific type, and then if necessary a suitable non-booked device can be selected for inclusion in the STP.

4.2.5.9 Lab-Scan System

The PeppesBodega-RMS saves information in two phases. In the first phase information is saved in the database and in the second phase a text file (specifically the TEC) is generated. This text file is currently used for two purposes:

- To create an installation script for a particular device and
- In the test environment (JCT and JCAT) this file is used to automate the generation of test instructions for this particular device.

4.2.5.10 Requirement-Request portal

Change requests and new functionality requests are handled via a web portal. All the stakeholders (see section 4.2.2) can submit requirements. The impact of these potential changes analyzed by a moderator who considers the estimated implementation details (technical architecture needs and deliverable artifacts) to meet the requirements of the proposed change. Then an estimate of the time that will be required to make this change is calculated and provided to the department responsible for the RMS to prioritize their implementation of these requirements.

4.2.6 Architectural design of PeppesBodega-RMS

This section describes the overall architecture design of PeppesBodega-RMS that is relevant for the proposed extension of it in this Master's thesis. The following are the main components of the PeppesBodega-RMS architecture (as shown in Figure 4-3): (1) Command line interface (CLI), (2) Web User Interface (UI), (3) Core Application, and (4) Database.

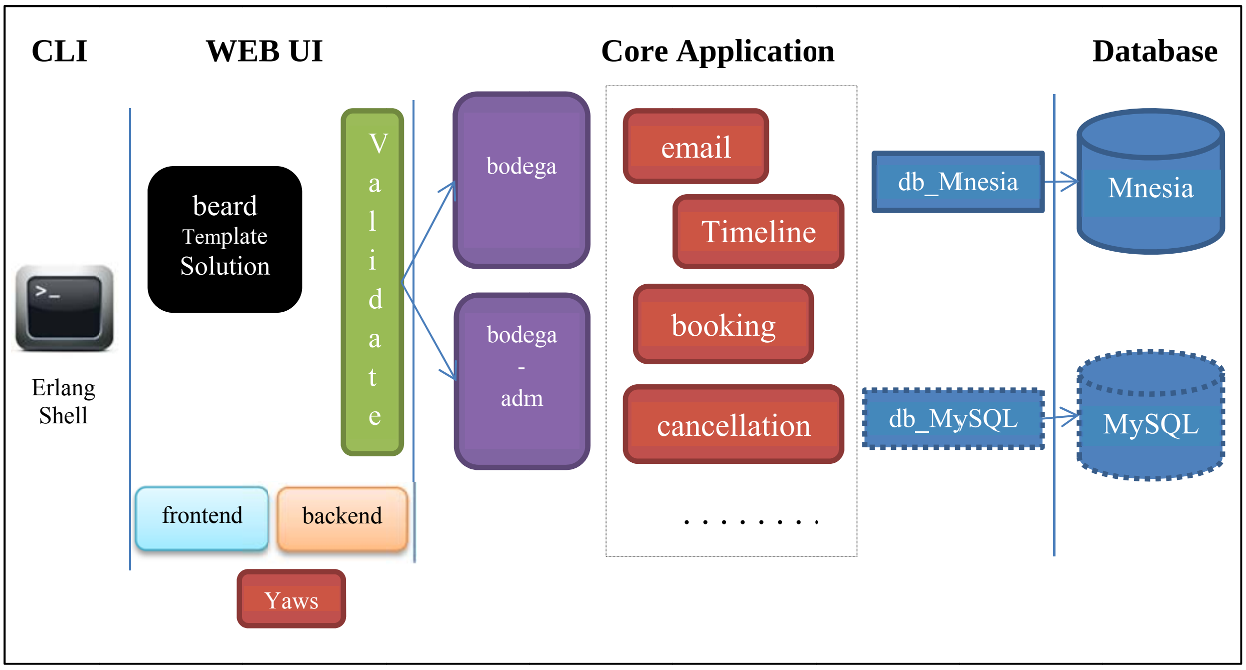


Figure 4-3: Architectural Design of PeppesBodega-RMS

4.2.6.1 CLI

The functionality implemented in the PeppesBodega-RMS (see section 4.2.5) can be used via a command line interface (CLI) by the moderator for debugging purposes. Other than the moderator, no-one else can use the CLI. The CLI is accessed via an Erlang shell, where all the functionality is available via the two core applications: bodega and bodega-adm (See section 4.2.6.3). As the CLI is not easy to interact with; a tool with an interactive graphical user interface (GUI) would be more convenient to use than a simple CLI; and that was the motivation behind users using a WEB UI in PeppesBodega-RMS rather than using a CLI.

4.2.6.2 Web UI

PeppesBodega-RMS is currently running on Yaws (Yet another webserver) webserver[*]. As yaws supports the dynamic-content web application, we needed a template solution to handle web requests. At first mustache [33] was used as a template solution, but due to the fact that this resulted in a slow response times, a similar template solution with high performance, called beard [32] was developed and is now used for creating webpages with dynamic-content selection. The beard template solution works with two input files: (1) a Template file and (2) a View-logic file.

The template file is used to provide the structure of the content (in HTML [34] format) on the webpage and does not contain any embedded logic. The view-logic file contains the functions used in the templates for data input via templates. The result is that the logic is isolated via the template, which has the advantage that the template, if needed, can be reused independent of natural language boundaries, i.e., the same view logic can be used with a template in English, Swedish, Chinese, etc. The framework under discussion has used a

[*] For details about YAWS see http://hyber.org/ .

separate directory structure for both view-logic and template files. Also the naming convention corresponding to view-logic and template is kept the same with two different extensions used in order to distinguish them from each other.

As discussed above PeppesBodega-RMS is using *beard* for its dynamic content selection and as the requirements increased so did the pattern catalogues (see section 4.2.6.3). There was a need for a front-controller to make the design sufficiently efficient that we can shift between different patterns by changing only a single entry point. The front-controller has been further divided into front-end and back-end parts. The front-end and back-end parts together provide the interface for different functional and non-function requirements (see section 4.2.5); some of the most important requirements were listed in section 4.2.6.3.

The input to the core application is firstly validated by the web UI module. This restriction of having the actual functionality and validation functionality in a separate place makes it easier to locate the bug in the event of a problem.

4.2.6.3 Core Application

As discussed in section 4.2.3, there are two access-rights groups, i.e. administrator group and user group. Based upon the functionality of both the groups and their ability to change the state of PeppesBodega-RMS, two APIs were developed:

1. bodega.erl
2. bodega_adm.erl

In general, states are very important to consider when it comes to efficient design techniques. QuickCheck [35] is an efficient property-based testing tool, which provides the user with the state of the SUT when a fault occurs. This in-turn is only possible if the implementation and the structure the code design utilized as few side-effects as possible and where side-effects are used they should be limited to only a few specific areas; although the use of functional programming languages (such as Erlang) restricts the usage of side-effects. PeppesBodega-RMS was carefully designed such that the use of side-effects is limited to only certain specific places.

The API ‘bodega.erl’ is responsible for the all of the actions performed by the user group, whereas the ‘bodega_adm.erl’ API provides the functionality for all actions that can be performed by the administrator group. The allowed state-changes available for members of the user group are the booking and unbooking of devices, while the allowed state-changes for the administrator group include all the possible state-changes in PeppesBodega-RMS (see section 4.2.5)

4.2.6.4 Database

A state-change in PeppesBodega-RMS is handled in two different ways (either state-change-independent or state-change-dependent), depending upon what has been changed and where the changes have occurred.

A state-change-independent state-change that is done locally is written directly to the database. This type of state-change is not an input to any test oriented task and is only used to administer the device usage information, such as booking and unbooking. Once this action has been performed, the new data is written to the database.

A state-change-dependent state-change is also done locally on the system but is *not* written directly to the database. The state-change remains local to the system, until a ‘lab-scan’ is performed and then the state-change is pushed to the database. This type of state-change is used in test oriented tasks, such as installation of devices, test case execution, etc. The reason for this dependency is to ensure that the contents of the database are readily available to the user when performing a test oriented task; this is achieved by storing the database in a user-readable format each time a lab-scan is performed.

Initially the core application was developed to support MySQL; later it was decided to shift to *Mnesia* [36] as all the RMS development was in Erlang, hence it was logical to use an Erlang based database in order to be consistent with the development tools. Today there is still support for MySQL, but *Mnesia* is the only database (actively) used in this framework as it caters for all the needs for PeppesBodega-RMS. In Mnesia the backend database storage can be either Ets[*] or Dets[†]. Ets tables resides in Erlang runtime system where as Dets are stored on disk. This reflects lower read/write time of Ets as compared to Dets. The selection of usage of Ets and Dets is done as a property[‡] provided during the creation of table in Mnesia database. In PeppesBodega-RMS Dets are being used.

[*] Ets, http://www.erlang.org/doc/man/ets.html, Last accessed : 2013-12-28

[†] Dets, http://www.erlang.org/doc/man/dets.html, Last accessed : 2013-12-28

[‡] property, {disc_only_copies, nodes()} refers to dets, {ram_copies, nodes()} refer to both ets whereas {disc_copies, nodes()} refers both to ets and dets. http://www.erlang.org/doc/man/mnesia.html, Last accessed : 2013-12-28

5 PeppesBodega Extended RMS's Topological Framework

PeppesBodega Extended RMS's topological framework is the artifact constructed (in this Master's thesis) to solve the problem stated in section1.1. DSR was adopted as a methodology throughout this Master's thesis project to create PeppesBodega Extended RMS - topological framework (TF). To simplify the naming we will refer to this artifact as PeppesBodega-TF in the remainder of this thesis.

5.1 Existing design

Chapter 4 explained the design transitions (section 4.1 and section 4.2) of PeppesBodega-RMS; it also stated different reasons for why a new solution was implemented to extend the existing PeppesBodega-RMS

PeppesBodega-RMS has been catering very well to the needs of all the involved departments when it comes to the functions expected of an ordinary RMS system, e.g. device tracking, efficient use of IP-pool and storage inventory etc. But the requirements for a topological RMS emerged, not only to contribute to even more structured information being available about each of the devices, but also to increase the efficiency of test automation.

5.2 Design Transition

PeppesBodega-TF should facilitate the activities of the different teams (see section 1.2) by providing more and better structured information about the devices and hence provide greater control over test automation. PeppesBodega-TF is expected to successfully fulfill all the objectives (stated in section 1.4) and to solve all the problems described earlier in this thesis.

The following subsections describe the step-by-step processes that this Master's thesis project employed to implement the desired solution.

5.2.1 Development tools and design

The development of PeppesBodega-TF has utilized the same development tools and followed the same design principle as PeppesBodega-RMS (see section 4.2). The reason for these being the same is that the artifact created in this Master's thesis project (PeppesBodega-TF) is not a standalone web-framework; but rather it is an extension of the existing PeppesBodega-RMS. So we found it fairly logical and convenient to use the same development tools and design principles. However, there are some new concepts introduced. Each of these new concepts will be discussed in following subsections.

5.2.1.1 Device Definition

In order to explain the following subsection, first we need to define the devices in the LAB and how they can be connected to each other. As the system verification domain is very broad in the MSV-NCI department, it is difficult to explain all possible configurations in the context of this topological RMS. Before explanation of any configurations, the relevant background definitions are needed. The configuration explanation will be developed during the explanation in each relevant subsection (See section 4.2.6.3).

5.2.1.1.1 Devices

As mentioned in section 4.2.5.1, each device is uniquely identified by a unique device identifier (DID) to keep track of the devices for LAB administration purposes. As of today these DIDs are *not* used inside a TEC; but there exists a plan for automating the device information for the convenience of verification engineers and to avoid conflicts with mismatches of information due to human error.

The devices (identified by DIDs) are associated with product identifiers (PIDs). A device associated with a PID guarantees that the device has certain specific functional features. More than one device can have same PID. PIDs (in Ericsson AB) have the same meaning as models of a particular appliance (in real life).

5.2.1.1.2 Interconnection

Each device has some connection points[*] with which other devices can connect to it; these connection points are referred as interconnections. We have logically grouped together the devices that have some common functionality (although having different PIDs) and having common interconnections. This concept of logically combining different devices on a ‘device type’ level will be discussed in section 5.2.1.4.

5.2.1.2 Grammar selection

PeppesBodega-RMS was built following quite good programming practices, of which the most noteworthy was avoiding hardcoded values in the source code. The Erlang config files are used as input to different processes in order to avoid hardcoded values being written in multiple places.

For the PeppesBodega-TF it was decided *not* to use the Erlang config files; but rather we opted for a computer grammar based solution. The reason for selecting a computer grammar was that we needed a scalable solution for definition of devices and their possible interconnections; and some of the inherent features of the selected grammar (e.g. allowed structure, composition of expressions, etc.) were helpful during the implementation.

To specify the definition of devices and their possible interconnections, we have selected a context-free grammar. As context-free grammars are extensively used for type definitions in programming languages, a similar approach was adopted to provide the foundation blocks used in the proposed solution.

As the SUT being used in the target department is being tested internally (see section 1.2) the grammar is domain specific. This Master’s thesis proposes a special grammar translation that can also be used externally.

5.2.1.3 Grammar definition

After selecting the grammar, the definition of the grammar was done in BNF[37–39] (in Erlang format). The BNF in its standard format and its interpretation in Erlang format (as relevant to this Master’s thesis domain) can be found in Appendix A.

[*] These connection points include both physical connectors and logic connectors (for example, an IP address, protocol, and port number).

We will explain three grammar expressions (with different levels); the rest of the grammar also follows the same principle. See Equation 5-1, Equation 5-2, Equation 5-3, Equation 5-4, and Equation 5-5 as references that will be used in expression definitions.

$$\{du, [' KDU\ 127\ 161/1\ R1A/4']\}$$

Equation 5-1 : Erlang grammar - logical binding

$$\{du, ['\mathrm{KDU\ 137\ 930/1\ P1B}']\}$$

Equation 5-2 : Erlang grammar - logical binding

$$\{' KDU\ 127\ 161/1\ R1A/4', [cpriport, cpriport, cpriport, cpriport, idlport, gpsoption]\}$$

Equation 5-3 : Erlang grammar - Interconnections

$$\{'\mathrm{KDU\ 137\ 930/1\ P1B}', [cpriport, cpriport, cpriport, cpriport, idlport, gpsoption]\}$$

Equation 5-4 : Erlang grammar declaration

$$\{cpriport, [du, radio, cpric]\}$$

Equation 5-5: Erlang grammar – Valid Interconnection options

Table 5-1: Levels of Grammar definition

Logical binding	As mentioned in section 5.2.1.1.2, we have logically grouped the devices with different PIDs. Referring to Equation 5-1 and Equation 5-2 both PIDs 'KDU 127 161/1 R1A/4'[*] and ''KDU 137 930/1 P1B'* have 'du' as their binding type[†]. The reason for this is that when we later define what interconnections can be connected to which device, i.e. 'valid interconnection option', it will be easier if we can just mention a binding type, in this case 'du', instead of giving each individual PID.
Interconnections	Equation 5-3 provides the details for connection points for a '$KDU\ 127\ 161/1\ R1A/4$' PID. To define a PID's interconnection we can specify multiple 'cpriport' but to uniquely identify each connection port, we concatenate each of 'cpriport' [‡] with a letter from the English alphabet. (For details see section 5.2.1.4.)
Valid Interconnection options	This level of definition for our grammar indicates what 'logical binding' a connection point can connect to. Equation 5-5 defines that a cpriport interconnection can only connect to a du, radio or cpriport (logical bindings).

[*] This is an Ericsson radio base station (RBS) 6000 Digital Unit example product.

[†] "du" can be interpreted as "Device Unit".

[‡] "cpriport" means Common Public Radio Interface (CPRI™) port - see Section 2.4.

5.2.1.4 Grammar translation

After defining grammar, it was equally important to translate the grammar into such a format that can be structured into objects, i.e., Erlang records. In Erlang a type definition is very strict, so it was important to have getter and setter functions for all the objects created. The translation of the grammar was divided into two parts:

1. Conversion of grammar into records and
2. Interfaces for records.

For the first part a parser was designed that performs the translation into records. This parser only translates PIDs with their interconnection into Erlang records (See Equation 5-3). As a device can have multiple 'cpriport' it is important to uniquely distinguish them, for this we concatenate a letter from the English alphabet (A-Z). Equation 5-6 shows a conversion of the grammar (PID with its interconnections) defined in Equation 5-3 into a Erlang record.

$$-\mathbf{record}(\boldsymbol{KDU}\ \mathbf{127}\ \mathbf{161/1}\ \boldsymbol{R1A/4}, \{\mathbf{cpriportA}, \mathbf{cpriportB}, \mathbf{cpriportC}, \mathbf{cpriportD}, \mathbf{idlport}, \mathbf{gpsoption}\}).$$

Equation 5-6: Erlang grammar - record definition of a PID

Then second phase defines the getter and setter for interconnections that a specific PID can use to connect to other devices. The restriction for getting any field (interconnection) for a record (PID) is that records are atoms[*] which cannot be passed as a variable to get or set any field in record. So getters and setter functions for all interconnection were mandatory to define at compile time. Equation 5-7 shows a getter function generated for interconnection 'cpriportA' of PID 'KDU 127 161/1 R1A/4'. Equation 5-8 shows the setter function generated for interconnection 'cpriportA' of PID 'KDU 127 161/1 R1A/4'. The getter and setter functions are automatically generated for all the interconnections of all PIDs defined in the grammar.

$$'\boldsymbol{KDU}\ \mathbf{127}\ \mathbf{161/1}\ \boldsymbol{R1A/4}'(\boldsymbol{Record}, \boldsymbol{cpriportA}) \rightarrow \boldsymbol{Record}\#'\boldsymbol{KDU}\ \mathbf{127}\ \mathbf{161/1}\ \boldsymbol{R1A/4}'.\boldsymbol{cpriportA}$$

Equation 5-7: Erlang grammar – interconnection getter function

$$'\mathbf{KDU\ 127\ 161/1\ R1A/4}'(\mathbf{Record}, \mathbf{cpriportA}, \mathbf{Value}) \rightarrow \mathbf{Record}\#'\mathbf{KDU\ 127\ 161/1\ R1A/4}'\{\mathbf{cpriportA} = \mathbf{Value}\}$$

Equation 5-8: Erlang grammar – interconnection setter function

Then the last phase defines the last level of a connection, i.e. what the interconnection can be connected to. For example 'cpriport' on device-A can be connected to device-B via a matching 'cpriport'.

The recursive definition of a device and its interconnection acts as an input to the records generation module. However, the definitions need to undergo a completeness test in order to generate the corresponding records (See section 6.3.1 for more details). For example, if the 'cpriport' is *not* defined and it is being used in the interconnection of a PID, then the parser will exit generate an error.

[*] atoms, http://www.erlang.org/doc/reference_manual/data_types.html

As the grammar and its parser is foundation component of the PeppesBodega-TF, so its testing needs to be performed to verdict its proper functional behavior. We have developed integrated testing in its design and also performed extensive testing with QuickCheck; see section 6.3.2.2 for details.

5.2.2 Integration with existing architecture

This section describes how the newly developed PeppesBodega-TF was integrated into the PeppesBodega-RMS architecture (see Section 4.2.6). We have integrated PeppesBodega-TF into the existing PeppesBodega-RMS so that users do *not* need to switch between two different tools.

Figure 5-1 shows the extended and newly developed subsystems in the context of the current PeppesBodega-RMS architecture. The PeppesBodega-TF architecture, in accordance with our design goal of it being an extension of PeppesBodega-RMS's design, has following four components: (1) CLI, (2) Web UI, (3) Core Application, and (4) Database.

Components (1, 2, and 4) were changed mainly due to the extension of the core application (component 3). Each of the components 1, 2, and 4, will be analyzed in a corresponding subsection with regard to the modifications made in it due to the extension of component 3.

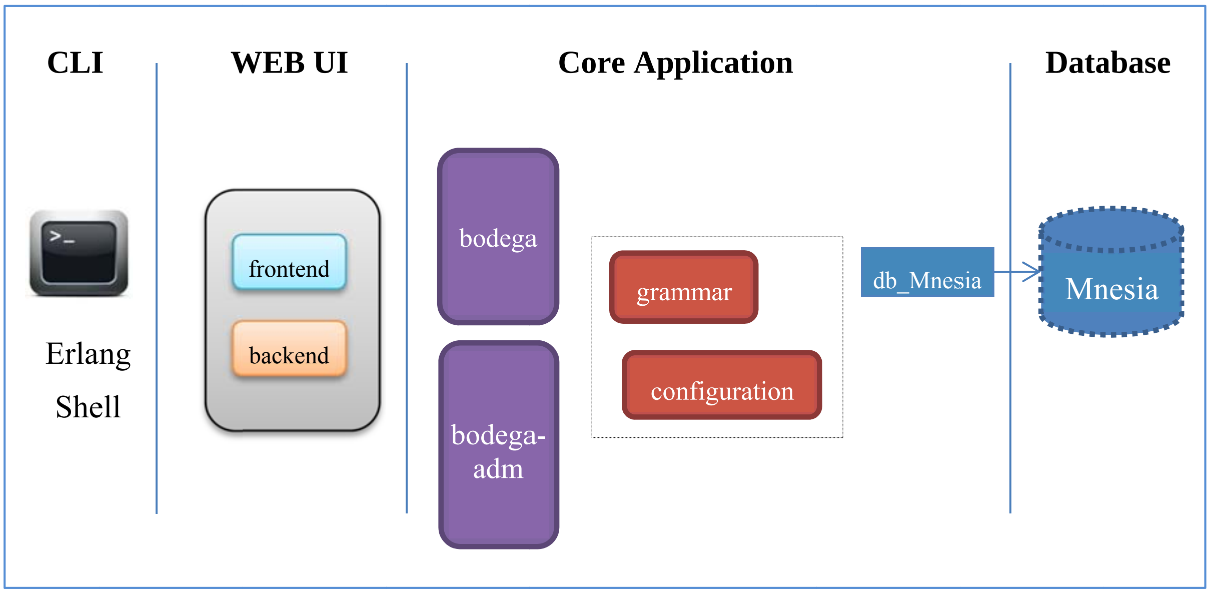


Figure 5-1: Architectural design of PeppesBodega-TF

5.2.2.1 CLI

As mentioned in Section 4.2.6.1, the CLI can be used for accessing the functionality provided in the core application for the moderator's debugging purposes. During the implementation phase of PeppesBodega-TF, the CLI was extensively used to test the newly implemented functionality. Only the working functionality was made usable via the web UI. The reason for not checking the functionality via the web UI was that if the application crashed, we had only limited opportunities to see the reason for the crash. Furthermore, using the CLI required less effort to display the desired debugging information in the event of a problem. At the end of the implementation phase, the non-functional testing was also done to see if the implemented solution contains any bottlenecks when PeppesBodega-TF will be actually deployed with bigger database than of today; see section 6.3.3 for more details.

It was also observed that there were some limiting factors regarding the web framework and its interface with the dynamic connections in the topology configuration. The following subsections will provide details of those limitations that hindered development of the web UI.

5.2.2.2 Web UI

As mentioned in Section 4.2.6.2, the web UI requires following four components: (1) Yaws, (2) Frontend and backend controllers, (3) beard compiler, and (4) Validation checks.

Yaws is still being used as a web server as we saw no reason to change to another webserver as no issues are identified that hindered the development of the Web UI. The frontend and backend controllers were modified to handle the interfaces for the creation and management of configurations; details internal to this extension are explained in Section 5.2.2.3. The beard compiler has been used without modifications. However, validation checks have been extensively implemented based upon the new functionality added to the core application.

The following four frontend views were created to interface to the new functionality provided by the core application.

- *Topology creation view*
  Figure 5-2 shows frontend view for creation of a topology. "Signum" refers to a user-id of a particular user. The backend actions of 'Topology creation view' checks for a valid topology name (i.e. a name consisting only of alphanumeric characters) and valid signum (i.e. consisting only of alphabetic characters) and then on successful validation the browser is redirected to the 'Topology signum view', while an unsuccessful validation redirects the browser to an error page.

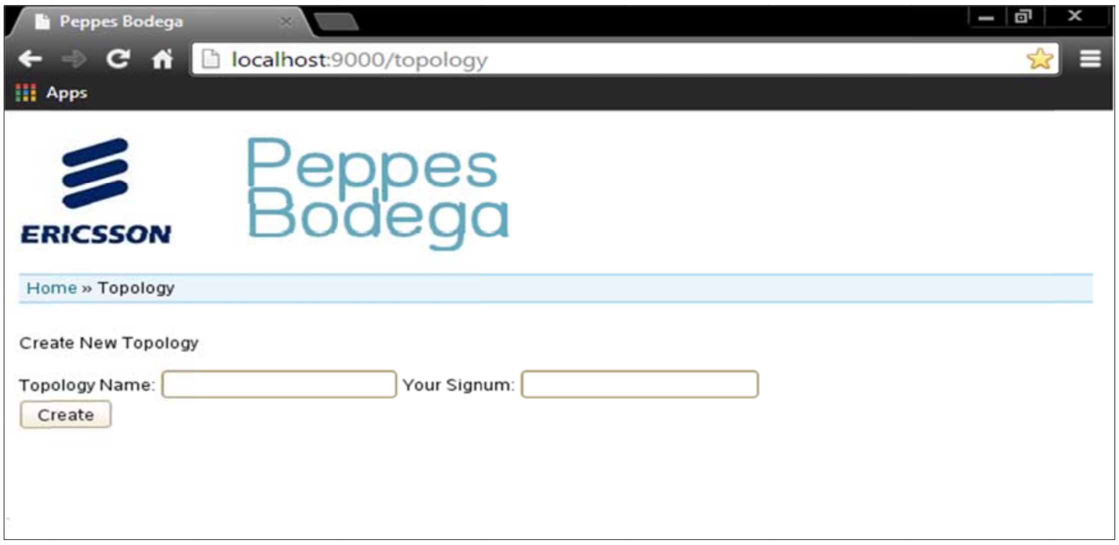
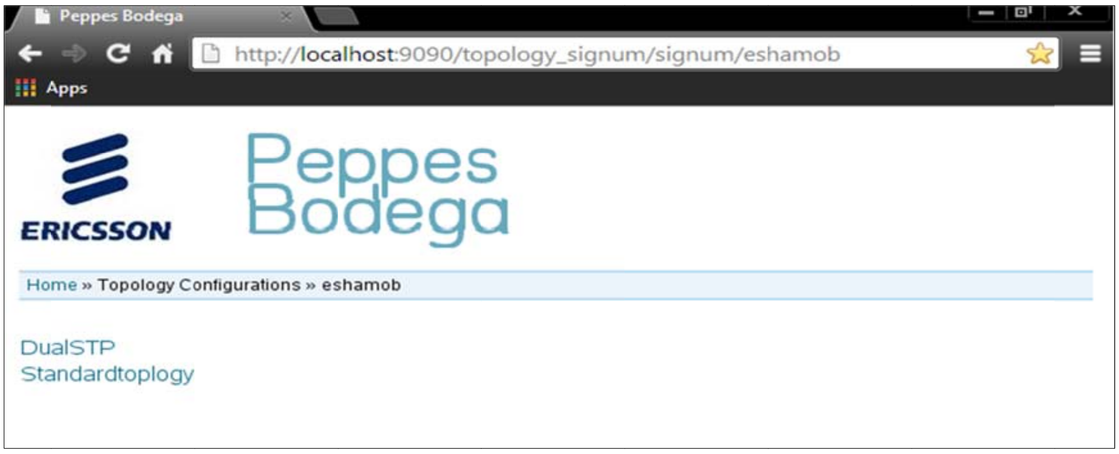


Figure 5-2: Create configuration

- *Topology signum view*
  In this view all the topologies are displayed that have been created by a particular signum. In *Figure 5-3* two topologies 'DualSTP' and 'Standardtopology' have been

created by signum 'eshamob'. The topologies listed in this view are hyperlinks to the management page of the selected topology.


Peppes Bodega

http://localhost:9090/topology_signum/signum/eshamob

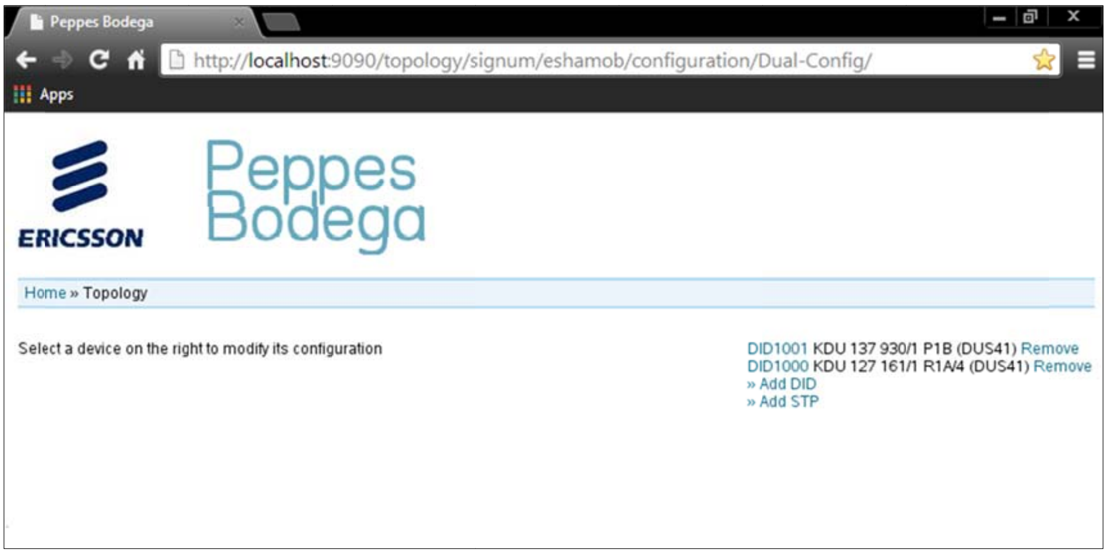
Home » Topology Configurations » eshamob

DualSTP
Standardtoplogy

Figure 5-3: Topology signum view

- *Topology management view*
  After selection of a particular topology for management, this view provides two options (See *Figure 5-4*):

Add STP/Devices	The 'Add STP' hyperlink provides the option to add all the devices in a given STP. While the 'Add DID' hyperlink provides the option of adding a device with a particular DID.
Edit Configuration	After successfully adding the devices needed to form the desired topology, the topological management view is then modified to include hyperlinks to manage each of the included DIDs.


Peppes Bodega

http://localhost:9090/topology/signum/eshamob/configuration/Dual-Config/

Home » Topology

Select a device on the right to modify its configuration

DID1001 KDU 137 930/1 P1B (DUS41) Remove
DID1000 KDU 127 161/1 R1A/4 (DUS41) Remove
» Add DID
» Add STP

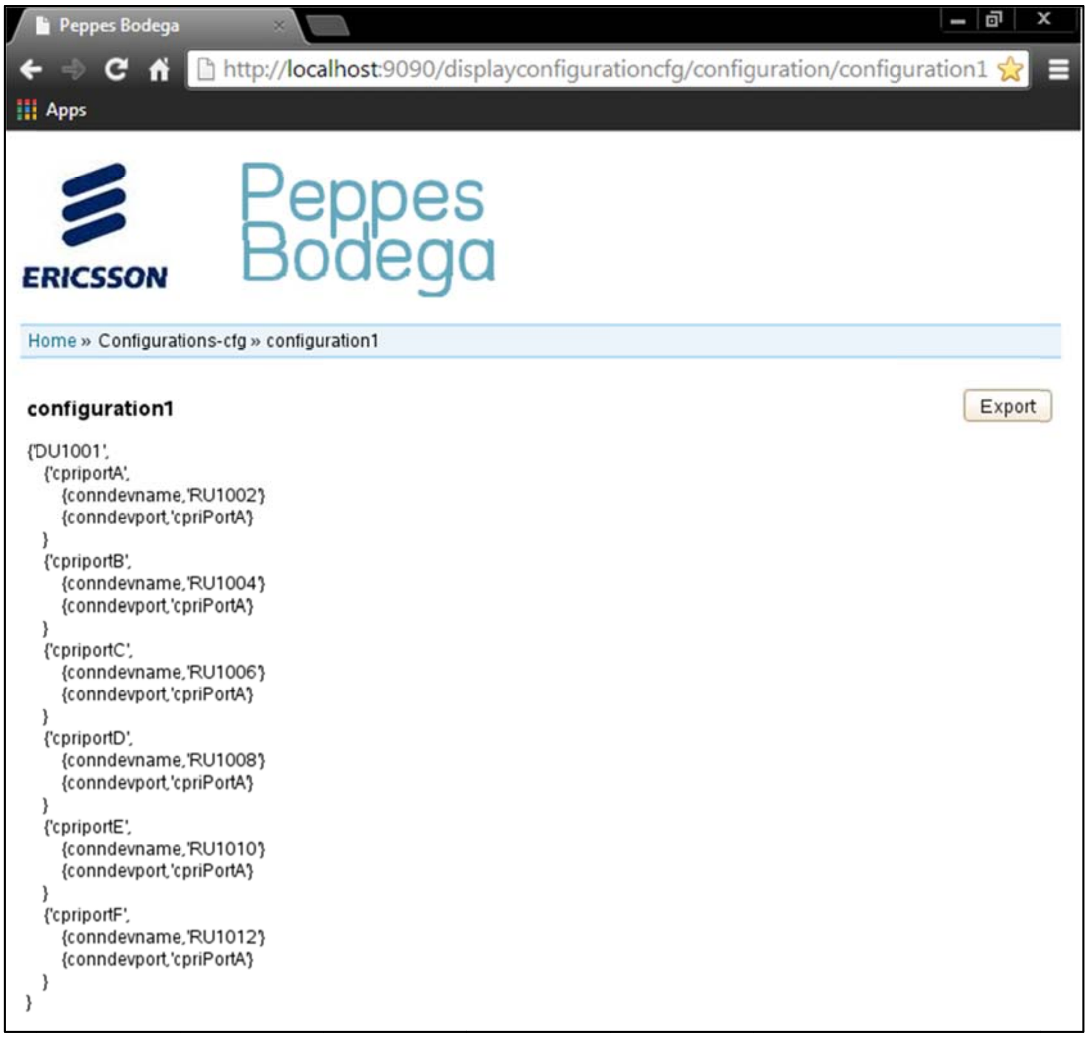
Figure 5-4: Topology management view

The development of the next view for connection of interconnections between devices was stopped due to the front-controller being able to handle large amounts of configuration topology data. The development of a new front-controller was viewed as a time consuming task that would have resulted in major deviation from the goals of this Master's thesis. So a decision was made that a new front-controller should be part of future work (see section 7.2).

- *Topology textual views*

In this view, the topological information developed in *Topology management view* is displayed in two formats: (1) XML format and (2) Erlang-config format.

For the Erlang-config format HTML tags* were used for formatting purposes in beard templates. In Figure 5-5 the topological information of configuration1 is displayed in Erlang-config format. Also this textual information can be exported to a file by using the 'Export' button.

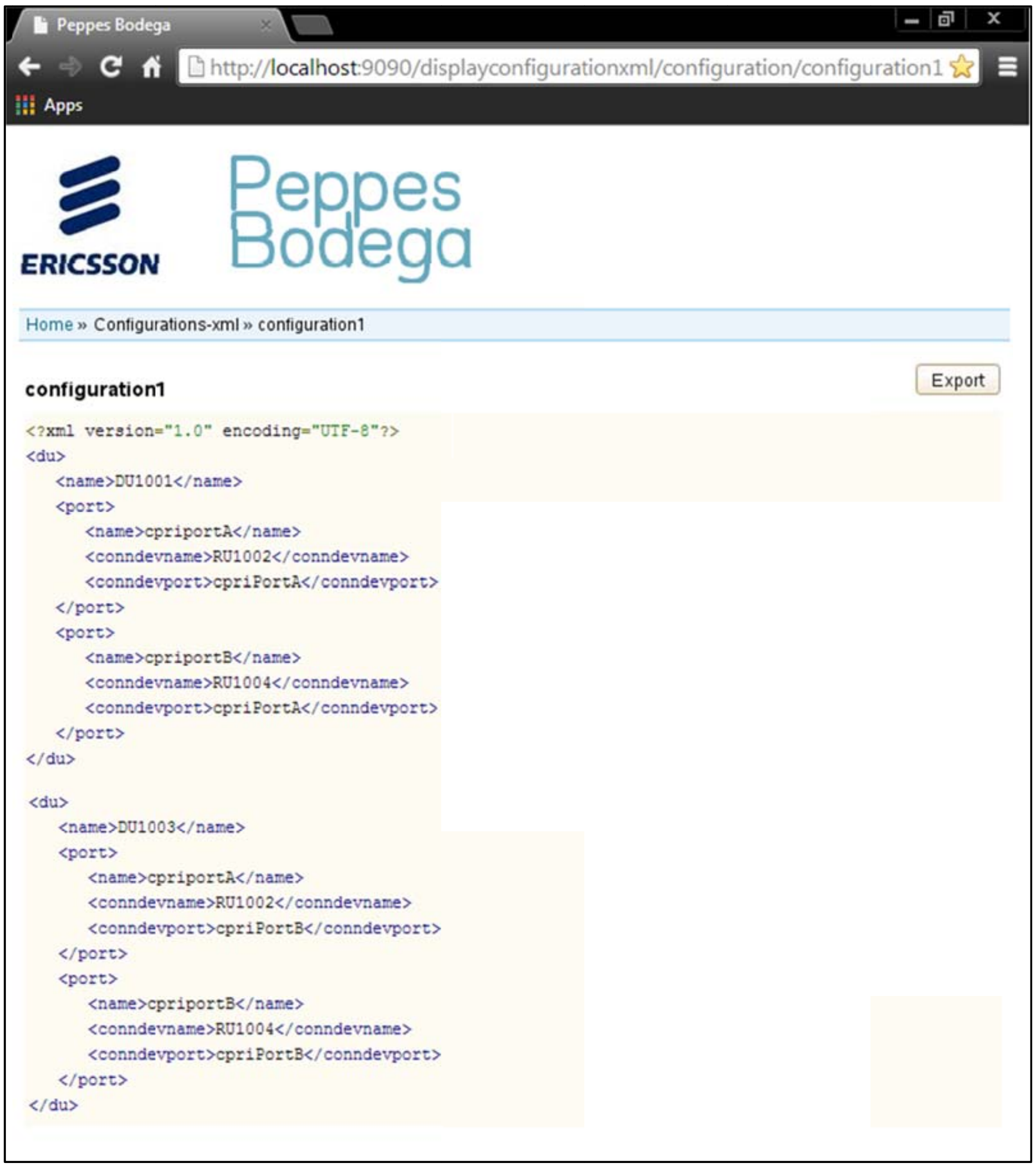

Peppes Bodega
http://localhost:9090/displayconfigurationcfg/configuration/configuration1
Apps
ERICSSON
Peppes Bodega
Home » Configurations-cfg » configuration1
configuration1
Export

```
{'DU1001',
   {'cpriportA',
      {conndevname,'RU1002'}
      {conndevport,'cpriPortA'}
   }
   {'cpriportB',
      {conndevname,'RU1004'}
      {conndevport,'cpriPortA'}
   }
   {'cpriportC',
      {conndevname,'RU1006'}
      {conndevport,'cpriPortA'}
   }
   {'cpriportD',
      {conndevname,'RU1008'}
      {conndevport,'cpriPortA'}
   }
   {'cpriportE',
      {conndevname,'RU1010'}
      {conndevport,'cpriPortA'}
   }
   {'cpriportF',
      {conndevname,'RU1012'}
      {conndevport,'cpriPortA'}
   }
}
```

Figure 5-5: Textual configuration - Erlang-config

* For more information about HTML tags see: http://www.w3schools.com/tags/

For the XML format, there were mainly two options: (1) to use the same approach that we used for the Erlang-config format and (2) modification of the front-controller as the front-controller of PeppesBodega-RMS only supported html encoding. Option 2 was selected as this displays nicely formatted XML in a colored format that is easy to read. Figure 5-6 displays the topological information of configuration1 in XML format. Additionally, this textual information can be exported to a file using the ‘Export’ button.


Peppes Bodega
http://localhost:9090/displayconfigurationxml/configuration/configuration1
Apps
ERICSSON
Peppes Bodega
Home » Configurations-xml » configuration1
configuration1
Export

```
<?xml version="1.0" encoding="UTF-8"?>
<du>
   <name>DU1001</name>
   <port>
      <name>cpriportA</name>
      <conndevname>RU1002</conndevname>
      <conndevport>cpriPortA</conndevport>
   </port>
   <port>
      <name>cpriportB</name>
      <conndevname>RU1004</conndevname>
      <conndevport>cpriPortA</conndevport>
   </port>
</du>

<du>
   <name>DU1003</name>
   <port>
      <name>cpriportA</name>
      <conndevname>RU1002</conndevname>
      <conndevport>cpriPortB</conndevport>
   </port>
   <port>
      <name>cpriportB</name>
      <conndevname>RU1004</conndevname>
      <conndevport>cpriPortB</conndevport>
   </port>
</du>
```

Figure 5-6: Textual configuration - XML

These results fulfil one of the goals of this Master’s thesis project, i.e. output a TEC for the SUT in textual format. After the desired configuration is displayed via the Web UI interface the user can simply export the configuration as a text file. This textual file can now be used by other tools to facilitate the desired testing.

5.2.2.3 Core Application

PeppesBodega -TF extended the two main APIs: bodega.erl and bodega_adm.erl. As mentioned in section 4.2.6.3, these API are used to modify the database. The implemented functionality has two main functional components:

- *Grammar* – A detail discussion was provided in section 5.2.1
- *Configuration* - This component supports the creation of configuration (topological) information. As described earlier the resulting configuration is saved in two formats: erlang(cfg) and java(xml). These formats providing TEC input to the two different test environments (see section 1.2) and provides a textual display in both formats.

Data structures are used extensively for organizing data in structured manner. Examples of data structures include trees, sets, hash tables, queues, stacks, etc. The primary reason for selection of a specific data structure for solving a complex problem is that a particular data structure guarantees certain features and we can build a solution on top of existing (solution) blocks. For examples, sets (implemented in different programming languages) implements union, intersection, subset, size, iterate, searching, … functionalities. Also some algorithms optimize the execution time required for set operations. We will not provide details of data structures and algorithms as this is a wide domain and each concept needs extensive discussion. The interested reader is referred to Aho et al. [40] which explains data structures, algorithms, and related topics with detailed discussions.

The selection of a data structure for encoding the configuration information was not an easy decision because the selected data structure will serve as the foundation for the next steps in the implementation. After analysing our needs (see section 1.4 and 2.5.2) two data structures (trees and graphs) were selected. We implemented a directed tree based structure for organizing the configuration information, but later when we tried to retrieve the information the loading lead to an infinite recursion. We therefore concluded that since the topological information was undirected, it was impossible to add and delete the saved configuration information in a directed tree data structure using conventional tree grafting and pruning operations. Also as there are multiple connections originating/terminating at a single device (due to multiple interconnections – see section 5.2.1.1.2), a tree due to its acyclic property (there exists only one route from one point to another) was not an appropriate solution. Furthermore, a tree cannot have unconnected interconnections that would be a valid configuration in our domain as a device may or may not be connected to another device. After learning these lessons, we re-analysed our needs for a topological RMS and found that an undirected graph better catered to all of our needs. The following are the main features with respect to interconnections that we obtained by encoding our configuration information in graphs:

Cyclic	More than one interconnection can originate from and terminate at all devices
Connected	A device can have interconnection(s) which are **not** connected to any other device's interconnection
Undirected	Regardless of the order of connections being made by specifying the origin and destination interconnections, the outcome will remain same. Thus a connection originating from interconnection A and terminating at interconnection B is the same as a connection originating from interconnection A and terminating at interconnection B.

This functional core application will be released for use within the company once it has been thoroughly tested (See section 6.3).

5.2.2.4 Database

As mentioned in section 4.2.6, the Mnesia database is currently used by PeppesBodega-RMS. In PeppesBodega-TF a new record 'topology' was added for structuring and a corresponding Mnesia table was introduced for storing topological information. The existing library for database transactions (addition/deletion and modification) to Mnesia tables was extended to incorporate the transactions for the 'topology' Mnesia table.

5.2.3 Interfaces

The CLI interfaces were developed first and then the beard structure was used to provide the web UI. As mentioned earlier the CLI only addresses the need for an administrator to invoke specific functionality, while the Web UI performs extensive validation before performing any actions. This means that the 'User group' should not have the full functionality of the CLI in order to avoid the risk of users performing operations that could result in a corrupted database or crash the server. For these reasons, the CLI functionality is only provided to the 'Moderator' for debugging and development purposes.

All the interfaces mentioned in section 5.2.2.2 were only for 'User groups'. The restriction of interconnections for an existing PID or a newly added PID need to be manually added to the grammar file, then PeppesBodega-TF is compiled to reflect the changes. These operations are only done by the 'Moderator'.

6 Testing and Evaluation

This chapter describes the testing and evaluation of PeppesBodega-TF (as presented in Chapter 5). We will evaluate to what extent the goals of the Master's thesis project were fulfilled and we will identify and analyze any discrepancies between the implemented solution and desired solution (as was specified in Section 1.4). After this the implementation of PeppesBodega-TF will be evaluated in terms of the characteristics of an RMS defined by Kulkarni et al.[21]. Last but not least reliability testing and performance testing of the topological information will be carried out.

6.1 Achievements and Discrepancies

In this section we will analyze the fulfillment of the goals (as defined in section 1.4) of this Master's thesis. Next we discuss the implementation that has been made in order to fulfill these goals. Finally, we will see if there are some discrepancies between the implementation and the proposed solution.

6.1.1 Achievements

The implementation and design of a topology aware RMS has been completed and details of each implemented feature were provided in Chapter 5. The implemented solution has been designed and implemented to flexible and it is easily expandable (see Section 5.2.1.2). The following enhancements were made to the existing RMS during the implementation of the proposed solution:

1. Improved security,
2. Restructured source code,
3. Improved frontend and backend controller, and
4. Removal of timeworn functionality.

6.1.1.1 Security

The credentials of administrators are now being processed using SHA-1 encryption. This means that plain text versions of the credentials are no longer being stored, thus increasing the security of the system.

6.1.1.2 Restructure of the source code

During the implementation phase, it was noticed that some of the parts of the source code were not using the beard structure, but rather used another method to do the same thing. Although this was not a problem in and of itself, in order to make the source code better aligned with the rest of the source, the source code was converted to consistently us the beard structure. This clearly increases the reusability of source code and allows direct communication between different parts of the code.

6.1.1.3 Improved frontend and backend controllers

In section 6.1.1.2, the restructuring of the source code also helped us to modify the frontend and backend controllers. This modification enabled us to remove unused code, directly reducing future efforts to maintain the source code and increasing the maintainability of the source code.

6.1.1.4 Removal of deprecated functionality

Identification and removal of outdated functionality was also done as part of this Master's thesis project.

6.1.2 Discrepancies

The implemented functionality is not yet completely available for all of the stake-holders (see section 4.2.2) because of the lack of a complete web UI. A detailed discussion of this was included in section 5.2.2.2.

The goal for providing detailed information about all of the devices in the LAB was fulfilled to the desired extent. The major challenge that we faced and which caused us to deviate from the goal of providing detailed information was the decision to not increase the number of devices available for DIDs. An example of the problem is that we do not track all the power distribution units despite the fact that there are quite a lot of them, as they were not actually part of each STP. Because these devices do not have unique DIDs a decision was made to not to include these devices into the extended RMS.

6.2 Analysis of the characteristics of PeppesBodega-TF

In this section we will analyze the implemented PeppesBodega-TF according to the metrics defined by Kulkarni et al.[21]. During the implementation phase, along with the requirement fulfillment these metrics were kept in mind. It was observed that the existing PeppesBodega-RMS implemented some of these metrics, but as the topological framework was introduced the RMS's functionality with respect to some of these metrics was also extended. The metrics defined by Kulkarni et al.[21] can be divided into three categories:

1. Administrative operations,
2. Components of a topological RMS, and
3. Features of a topological RMS.

6.2.1 Administrative operations

As discussed in section 2.5.2.1, the RMS should be able to provide following three essential operations to administrator: Monitor, Manage, and Control the RMS.

6.2.1.1 Monitor

In PeppesBodega-RMS the devices are used for testing purposes, thus monitoring should **not** affect the ongoing tests using the devices. For this reason, monitoring of network connected components was implemented in PeppesBodega-RMS by sending a 'ping' request to those devices with one or more IP addresses. This monitoring technique only monitored if the device was reachable or not. This technique was unable to completely monitor all of the devices, as not all of the devices in the LAB had an Ethernet connection; hence monitoring all of the devices was impossible using only this technique. In PeppesBodega-TF, no additional monitoring technique was needed as the devices in each configuration were the same and we could not perform any action other than checking the 'ping' status of devices with Ethernet connections and that was already implemented.

6.2.1.2 Manage

In PeppesBodega-RMS after getting information, the management of a reachable device required no changes for these devices, but for non-reachable devices the main reasons for a device not being reachable was due to the device being in faulty condition, a problem in the network, or because the device was powered off. In the case of a faulty device (due to a hardware fault) the non-reachable device should be replaced. In the case of a faulty device (due to a software fault) the device should have its software re-installed. A powered off devices should be manually removed from the testing LAB after at most one week, i.e., the device should be returned to the pool of available devices so that more effective use could be made of it. In the case of network problems the network issue needed to be manually identified and resolved. With the implementation of PeppesBodega-TF, the control of devices has improved as now the administrator simply has to check if the non-reachable devices are being used in any configuration that has other devices power-on in that configuration. Consider an example configuration, configuration-1, consisting of two devices: device-A and device-B. If device-A is non-reachable due to power-off then the administrator must look into the configuration containing device-As. In this case the administrator looks at configuration-1, if any of the devices in this configuration are powered-on then the administrator can assume that device-A is powered on, otherwise this device-A should be removed from the testing LAB, returning it to the pool of available devices. The overall management process for a powered off device is illustrated in Figure 6-1.

As described above, PeppesBodega-TF does take advantage of the transitivity information from the topology information in the RMS. A potential future enhancement would be to *monitor* devices using the transitivity information in the topology database, i.e., if we have connectivity between two devices within a STP, then if we have network connectivity to one of them – we can assume that the other devices is present – since it is part of the STP.

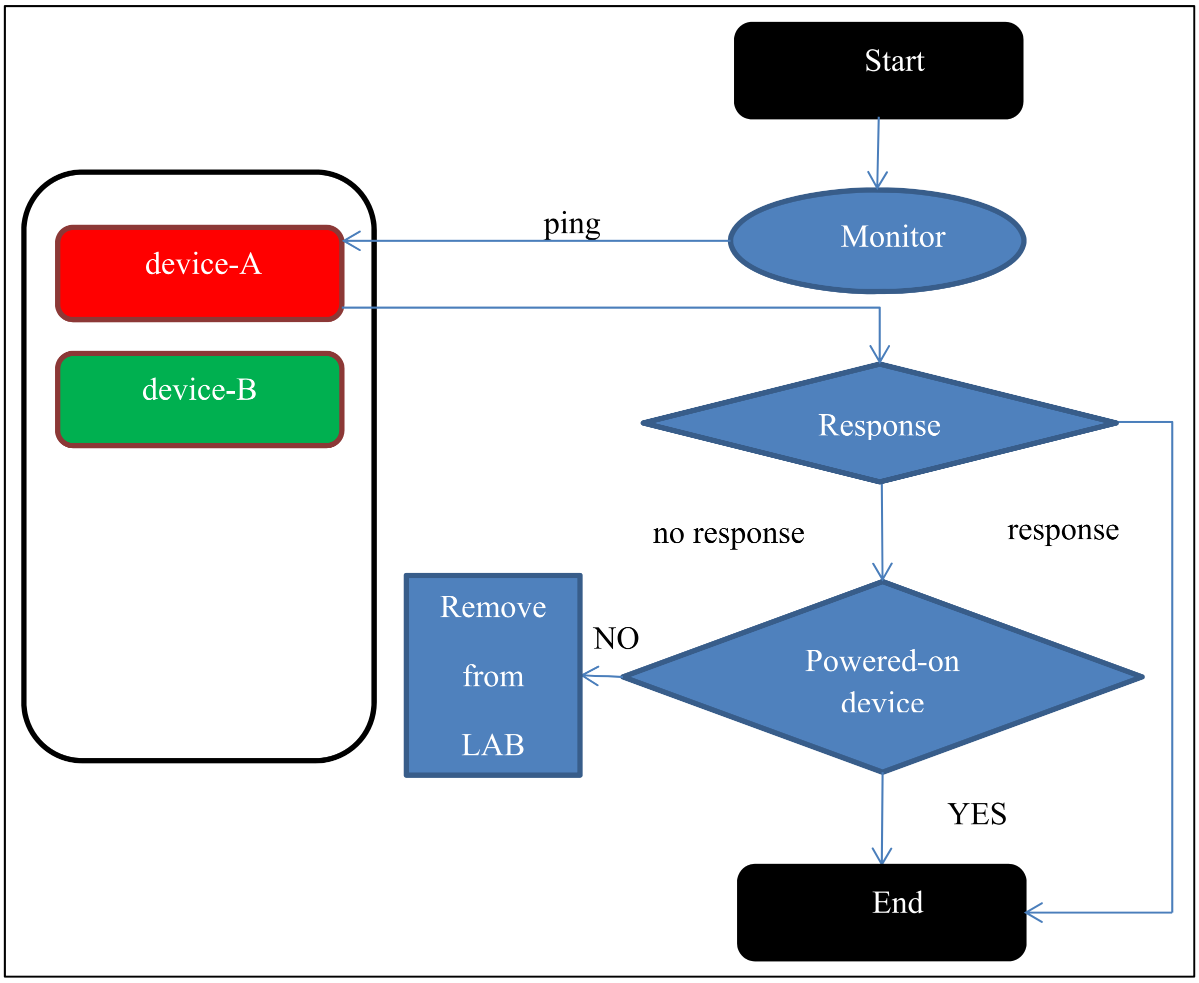


Figure 6-1: Device management of a powered-off device

6.2.1.3 Control

PeppesBodega-RMS had a limited ability control the managed resources. Even though the devices were grouped into STPs having a label, such as "used for test purpose-x", the RMS lacked control of the usage of these devices. Now that PeppesBodega-TF stores the topological information of the devices in one or more STPs. This information can be used to help the tester have greater controls over the devices because the system will generate an error if someone else, mistakenly, defines an interconnection to a device that already has a defined interconnection in another STP. For example, in configuration-X of Alice, if device-1 has interconnection port-A connected to interconnection port-B of device-2, and then if Bob tries to define a connection for interconnection port-A of device-3 to interconnection port-A of device-1 an error will be generated. This is shown in Figure 6-2.

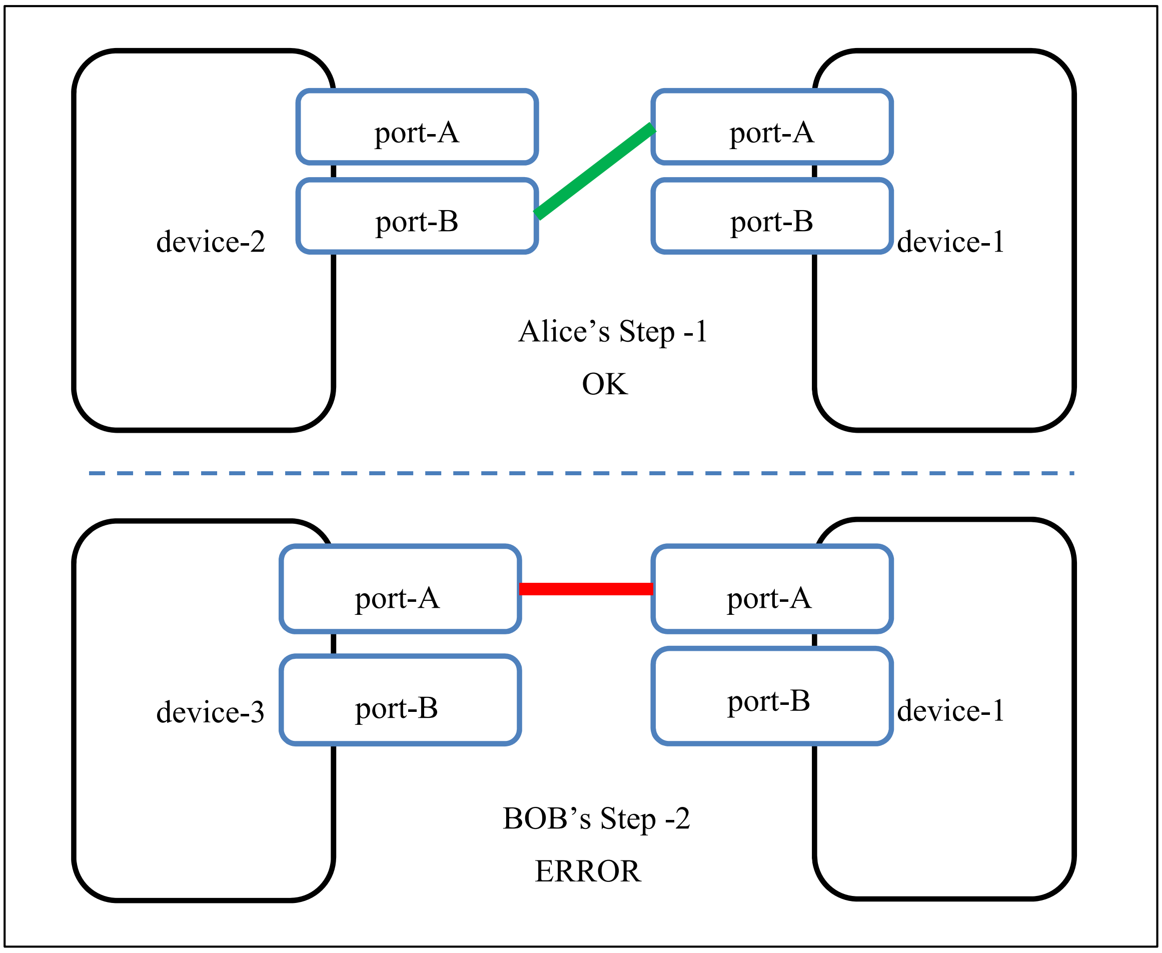


Figure 6-2: Control of devices

6.2.2 Components of a topological RMS

In this section we will analyse if PeppesBodega-TF has the basic components of a topological managed RMS according to the criteria described by Kulkarni et al. We will use the same approach as in section 6.2.1 due to some characteristics already being present in PeppesBodega-RMS. Kulkarni identified the following essential components of a topological RMS:

1. Plurality of nodes in the network,
2. Plurality of interconnections among these nodes,
3. A management system consisting of managed network resources stored in a database,
4. Database of managed network resources, and
5. Plurality of network administrators.

6.2.2.1 Plurality of nodes in the network

As mentioned in section 4.2, all of the devices in PeppesBodega-RMS are uniquely identified by DIDs, i.e., their device identifiers. So this component already existed before the development of PeppesBodega-TF. No specific enhancements were made for this component.

6.2.2.2 Plurality of interconnection among these nodes

Improvements to this component comprise the main contribution of this Master’s thesis project. PeppesBodega-TF is implemented to provide the functionality for all (usually more than one) interconnections between a device and another device. The detailed description of this functionality was provided in section 5.2.

6.2.2.3 A management system consisting of managed network resources stored in a database

This component refers to storing the data of each specific configuration. PeppesBodega-RMS stored the data specific to device, STP, and booking information. A similar approach was used in PeppesBodega-TF to store this data into a Mnesia database. In PeppesBodega-TF this same database was also used for storing topological information (see Section 5.2.2.4 for details).

6.2.2.4 Plurality of network administrators

As mentioned in section 4.2.2, there are local administrators for three regions of stakeholders for PeppesBodega-RMS administration. The implementation of PeppesBodega-TF enables more administrative usage by the *User group*, rather than restricting these administrative functions to the *Administrator group*. This is because the '*Moderator*' can define the possible interconnections (see Section 5.2.1.3) for each new device PID (see Section 5.2.1.1.1) in PeppesBodega-TF database. Given these definitions most of the topological information about a given configuration will be done by a member of the *User group* in order to generate their desired TEC input.

6.2.3 Features of a topological RMS

In addition to administrative operations (see section 6.2.1) and components of the topological RMS (see section 6.2.2), we will analyze if the implemented solution has following essential features:

1. Modification of configuration data,
2. Visualization of "Node view", and
3. Auto-update of "Node view" with addition of a new parent node.

6.2.3.1 Modification of configuration data

This feature is implemented and was discussed in Section 6.2.2.4 during discussion of '*Plurality of network administrators*'.

6.2.3.2 Visualization of "Node view"

This feature has **not** been implemented due to the limited duration of this Master's thesis project. We have made a suggestion about its implementation as part of future work (see Section 7.2). However, the textual TEC (see Section 5.2.2.2 : Topology textual view) contributes a good conceptual picture to understand the topology of a configuration.

6.2.3.3 Auto-update of "Node view" with addition of parent node

PeppesBodega-TF strongly supports this important feature. The core application uses undirected graphs (as mentioned in section 5.2.2.3) as its primary data structure. This guarantees that the information is updated for *both* devices when their interconnections are connected.

6.3 Testing of PeppesBodega-TF

This Master's thesis targets the ease of verification by providing a topological RMS (see section 1.3). The development of PeppesBodega-TF is the result of this effort to ease the verification process, but first PeppesBodega-TF itself needs to be tested.

Figure 6-3 shows the complete cycle of requirements' delivery. As testing is mainly dependent upon PeppesBodega-RMS, testing of each new step in the implementation of this extended RMS along with legacy testing was very important. The functionality of the PeppesBodega-TF is completely developed (in the alpha phase) and it fulfills the goals of this Master's thesis project. This implementation has also been tested at the module level (in the beta phase). We will evaluate this implementation in the following two subsections.

Yet another important aspect is to test CLI interface along with the Web-UI. As mentioned in section 5.2.2.2 front-end controller is unable to handle the request to perform 'edit configuration' so system testing for Web-UI is not performed. After future work for the front-end controller is done(see section 7.2), PeppesBodega-TF will then be released for the target audience after delivery test (QuickCheck[35]) has been performed. This delivery test, with its dependency for efficient front-end controller, remains as part of future work.

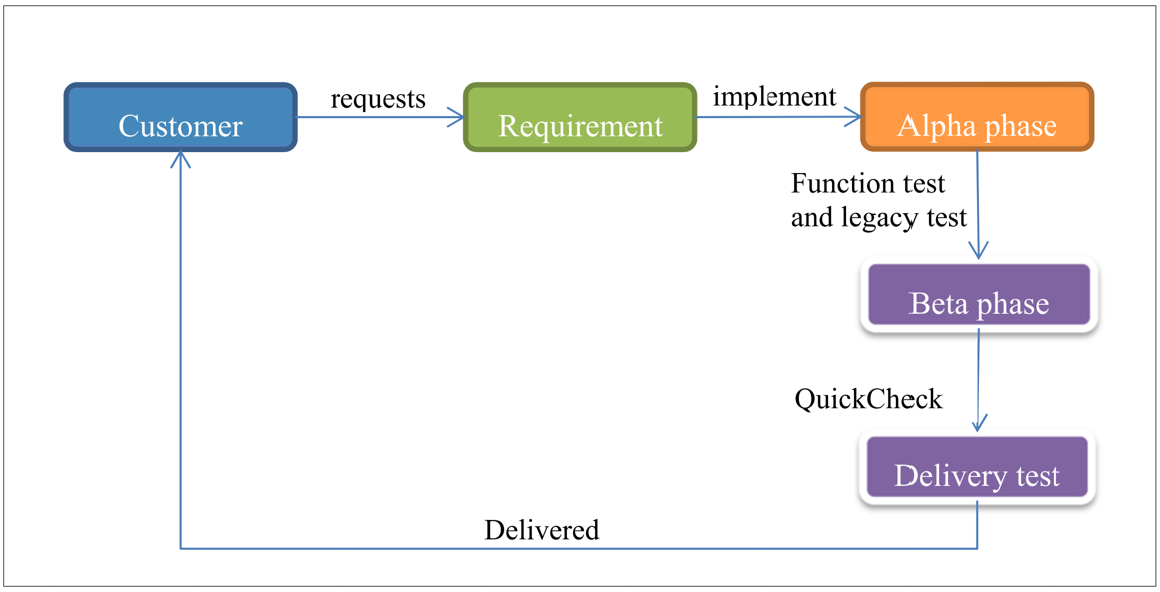


Figure 6-3: Requirement delivery plan

6.3.1 Test configuration

Currently, PeppesBodega-RMS contains 397 PIDs bound to 49 logical bindings and 745 devices grouped into 370 STPs. Use of PeppesBodega-RMS started to track this information roughly a year ago, thus no statistical analysis can yet be made to predict the future state for next year. According to LAB team, the number of devices will double in next year (as of now) and preparation of a new LAB is currently taking place to accommodate these new devices. This means that the implemented solution should be analyzed with atleast the size of the data-set prediction for one year from now.

We have simulated a database of 1500 devices and randomly allocated devices to 750 STPs. Also these 1500 devices are associated with 800 PIDs which are bound to 100 logical bindings. The implemented solution is mostly concerned supporting with the expected number of devices, but the pre-requisite PIDs and logical bindings are also needed for the creation of these devices. STPs are an integral part for the creation of a configuration for a STP (See section 5.2.2.2), so simulation of STPs was necessary.

The computer used for the performance testing was a **' HP EliteBook 8560w'** with a **'Intel 2nd Gen Core i5 i5-2540M**' processor running at **2.6 GHz**. This computer has **8 Gbytes** of memory with a clock speed of 1033 MHz and **320 Gbyte** disk (model **WD3200BEKT** at **7200 RPM)** connected via a **SATA** interface operating at 3 Gbps. The underlying operating system was **'SUSE Linux Enterprise 11'** and **Erlang** and **Yaws** were the only tasks running, other than the OS internal services and tasks, during the time that the testing was taking place.

6.3.2 Reliability testing

Reliability of software refers to its ability to produce the same result under the same circumstances. In other words this software should have no *side-effects*[41]. However, in practice this is very difficult to achieve. Of course the fewer the side-effects, the more reliable the software can be. Another aspect of the reliability of software is its ability to handle all of the desired use-cases, thus the software should be able to handle all the possible correct and incorrect inputs and outputs the desired results (including errors) gracefully.

Based upon the discussion above, we will evaluate PeppesBodega-TF with respect to following two criteria:

1) *Validity:* To verify that the software under test correctly handles the range of input in accordance with the specification. To check this we will inspect the software if the software contains any side-effect, i.e. if it behaves differently depending upon the state of the software. This testing will be done to ensure that the implemented functionality generates exactly same results for a given input.
2) *Completeness:* To verify that the software under test gracefully handles all possible inputs even in cases outside of the specification.

6.3.2.1 Validity

As mentioned earlier PeppesBodega-RMS was designed following good software design principles. One of those principles is the restriction of side-effects to a small number of functional modules. This way of structuring the code limits the software bugs to only those modules that have side-effects; hence it is easier to troubleshoot the resulting implementation when any problem is found. PeppesBodega-TF also follows the same design principle.

As mentioned in section 4.2.6.1, the CLI mode is not available to the user group, so the only way to use functionality is via the web UI. Also as mentioned in section 4.2.6.2 the view-logic contains the functionality that is provided via the web UI; this logic also provides filters for input requests to the module's functionality. So the validity of each input to the system is provided by the view-logic and restrictions on user group providing invalid input to functional module strengthens the software's validity.

Equation 6-1 shows the entry point of a web UI that provides the interface for adding a new configuration. We have two test criterions for the creation of a new configuration:

1. Configuration name
2. Signum

Equation 6-2 and Equation 6-3 checks for configuration name to only have alpha-numeric characters [A..Z a…z 0...9] and signum only contains characters from the alphabet [A...Z]. If either criterion is not met, then the interface should prompt the user with an error description and should not allow the user to proceed with the creation of a new configuration.

$$\{is\_valid\_signum(Signum), is\_valid\_name(Name)\}$$

Equation 6-1: Web UI validation for configuration name and signum

$$(C >= \$a \; andalso \; C =< \$z) orelse \; (C >= \$A \; andalso \; C =< \$Z) \; orelse \; (C >= \$0 \; andalso \; C =< \$9)$$

Equation 6-2: Validation for alpha-numeric characters

$$(C >= \$a \; andalso \; C =< \$z) orelse \; (C >= \$A \; andalso \; C =< \$Z)$$

Equation 6-3: Validation for alphabetic characters

The validation checks are not only limited to above mentioned checks as they are apparently checking only the character input. The input of is_valid_signum (see Equation 6-1) is validated against x500[*]; if and only if a signum exists in the x500 should the new configuration be created. This is the last step that in the current implementation is required as of now for signum validation to pass. Another aspect is that users have no restriction with regarding to adding new configurations to any signum, restricting the users from creating a new configuration for any signum other than their own remains as future work (See section7.2).

The configuration names can be same for two users since when we write to the database we concatenate the signum with the configuration name, thus making a unique name. This means that if the configuration name is already defined for a signum, the interface will prompt the user with an error message stating “Configuration already exists!!!” and will not create a new configuration in the database.

6.3.2.2 Completeness

In the previous section we described how we checked the validity of all the inputs that are provided to the core application. When it comes to proving that all the use cases are handled successfully, the core application must undergo a completeness test. The core application of PeppesBodega-TF revolves around the grammar for all provided functionality (see Section 5.2 for more details)

6.3.2.2.1 Integrated design test

As stated above that grammar must undergo a completeness test for the core application to function properly, hence we have defined a recursive parser that checks for the definition of all the logical bindings of a PID, its interconnections, and interconnection options.

Figure 6-4 shows the relationship between the different grammar definitions. The grammar definitions have been explained in the context of PeppesBodega-TF in Section 5.2.1.3. The parser takes as input all the defined PIDs under a logical binding, and then checks the definitions of its interconnections, then continues to check for interconnections options.

[*] X500, http://www.x500standard.com/index.php?n=Main.HomePage, Last accessed : 2013-12-23

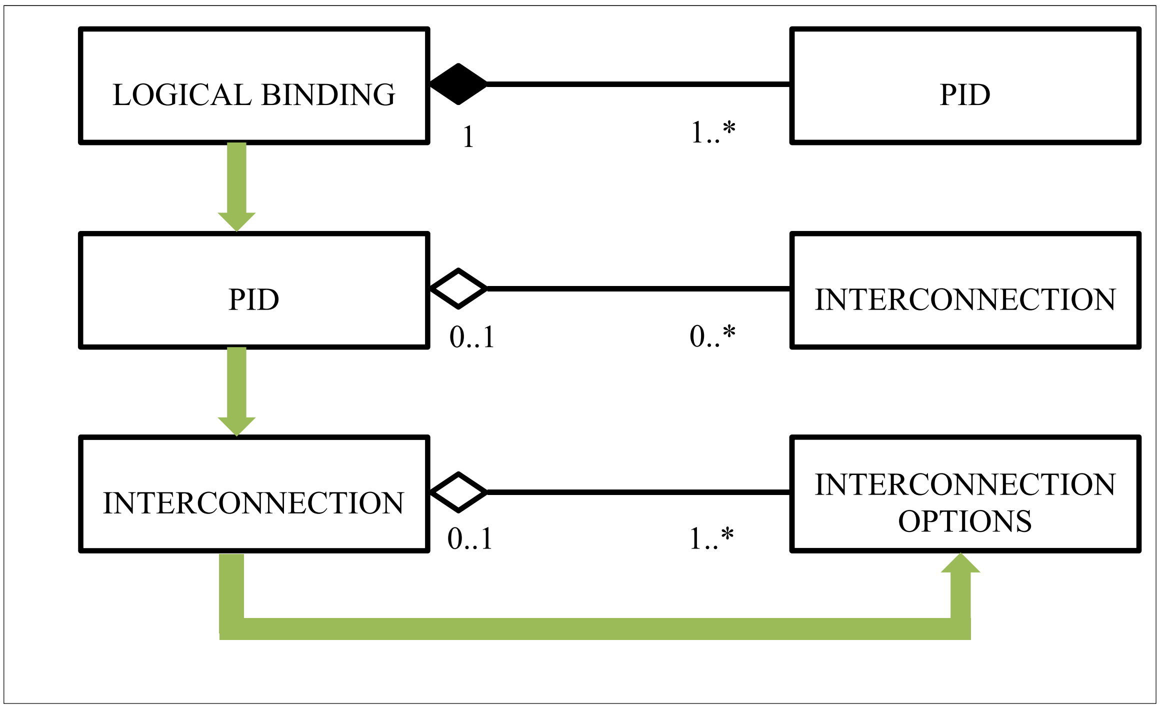


Figure 6-4: Recursive grammar parsing

If at any phase of the recursive loop fails for any PID belonging to logical binding, the PeppesBodega-TF application is not started. In other words, the start-up of the PeppesBodega-TF is the proof that the system is able to parse the correct input.

6.3.2.2.2 *Fuzz testing with QuickCheck*

To further check the completeness of the grammar and analyse the parser, we have used QuickCheck to generate samples of the grammar to test two parts:

1. Grammar
2. Parser of the grammar

As mentioned above the parser of the grammar is working fine with the current logical bindings, PIDs, interconnection, and interconnections options. However this is insufficient as a satisfactory test verdict for the parser of grammar to be working fine, nor does it confirm if the grammar produced is valid or not. To test this we have defined formal specifications of logical bindings, PIDs, interconnection, and interconnection options. The complete code for the QuickCheck module is provided in Appendix-C. We firstly define the rules for the grammar that confirms the validity of the grammar. This means that we have control over both the generation of valid and invalid grammars.

The generation of a valid grammar will be used to test if the parser fails with the defined rules. The generation of an invalid grammar will be used to test the behaviour of the SUT. We performed an extensive testing for this using QuickCheck. We checked all the rules that could be defined for the input of the grammar in terms of logical bindings, PIDS, interconnection, and interconnection options. These rules were then used multiple times to generate a large grammar that we testing for validity with and without injecting faults.

QuickCheck has its own built-in APIs which provide certain functionalities, we will only provide an overview of the functionality being used for testing the grammar. The intricate details of using QuickCheck-APIs will not be discussed in detail, as this is a vast domain and lies outside the scope of this Master's thesis.

For generation of input for logical bindings, interconnections and PIDs, we completely randomized the input as shown in Equation 6-4. The QuickCheck testing failed for one input of a sentence in the grammar because the definition contained the level of grammar itself as shown in Equation 6-5. The QuickCheck test failed and our analysis shows that the parser entered a infinite recursive loop, hence it would never exit. The implementation of parser was then changed for it to handle the case where the same name is used for definition of levels of grammar (See Table 5-1 for details); an error "Error: Self-definition" is generated on console followed by graceful exit.

```
atom() ->
  frequency(
    [{0,?LET(Str, non_empty(list(char())), list_to_atom(Str))},
     {9,?LET({Lc,Str},
                  {lowercase(), list(alphanumeric())},
                  list_to_atom([Lc|Str]))}]).
```

Equation 6-4: Random generation of grammar levels

```
{a,[a,b,c]}
```

Equation 6-5: Error grammar for same definition level

After analysing the failure of input for same input in definition of levels of grammar the range of input for the number of logical bindings, interconnection, and PIDs was redefined as shown in Equation 6-6. All the inputs are enumerated from 1 to 10 and for them (logical bindings, interconnection, and PIDs) to be able to be distinguished them from each other we have appended the initials from their names, for example logical binding: LB_. As we are randomly generating the grammar, this naming convention ensures that later-on it will be easy to interpret the generated grammar.

```
    logical_binding() ->      oneof([ list_to_atom("LB_ " ++ integer_to_list(X))
|| X <- lists:seq(1,10)]).


    interconnection_level()   ->   oneof([   list_to_atom("ICL_   "   ++
integer_to_list(X)) || X <- lists:seq(1,10)]).


    product_identifier() ->
     ?LET({Prefix,Atom},   {oneof(["KDU   ",   "KRC   "]),   oneof([
list_to_atom("PID_ " ++ integer_to_list(X)) || X <- lists:seq(1,10)])},
          list_to_atom(lists:concat([Prefix, Atom]))).
```

Equation 6-6: Random generation of input for grammar

The inputs to the grammar have been defined in Equation 6-6. The next step is to define the rules for a valid grammar. Equation 6-7 shows the rules for a valid grammar. Earlier we have explained in Table 5-1 (in conjunction with explanation of Equation 5-1, Equation 5-2, Equation 5-3, Equation 5-4, and Equation 5-5) that the logical binding holds a list of PIDs, with each PID is associated with a list of interconnections, and each interconnection associated to a list of logical bindings that it can be connected to. These specify the valid rules for the generation of a grammar. Also as mentioned in the explanation of Figure 6-4 all the definitions for PIDs, interconnections, and logical bindings must be defined. Any deviation from these rules will generate an invalid grammar. In Equation 6-7 grammar is generated from the rules mentioned above and then passed to the next step where again new set of grammar is generated from the same set of logical bindings, interconnection level, and product identifier. This provided an additional check for our analysis that *"the parser is able to handle same sentences in the grammar file"*. This is of course a valid grammar case but it should not add anything to the functional behaviour of parser.

```
rules(LBs) ->
  ?LET(
     {LB, PIDs, ILs},
     {logical_binding(),
      non_empty(list(product_identifier())),
      non_empty(list(interconnection_level()))},
     begin
       [{LB,PIDs}]++
             [pid(Pid,ILs) || Pid <- PIDs]++
             [il(IL,LBs++[LB]) || IL <- ILs]
     end).
```

Equation 6-7: Rules for valid grammar

We observe that the functional behaviour of parser is not affected by repetition of the same sentences in the grammar file.

Equation 6-8 shows the generation of a grammar from a set of valid rules as defined in Equation 6-7. QuickCheck also provides the possibility to have a user defined size for grammar repetitive generation i.e. control of input size by macro function '*?SIZED*' . If no input is specified, then by default 100 testscases are run; in our case this causes the generation of 100 different grammars.

```
grammar() ->
  ?LET(G, ?SIZED(Size, grammar(Size)),
      lists:flatten(G)).
grammar(N) when N =< 0 ->
  [];
grammar(N) ->
  ?LET(
    Grammar, grammar(N - 2),
    begin
      LBs = logical_bindings(Grammar),
      Grammar ++ rules(LBs)
    end).
```

Equation 6-8: Grammar generation

The test cases are executed against two properties

1. Valid grammar
2. Invalid grammar

A valid grammar is then provided as input to the parser to check if the parser correctly outputs the record definitions and getter/setter functions (as mentioned in section 5.2.1.4), in the case of failure QuickCheck will abort the ongoing testing and will output the grammar used with the parser. This valid grammar check is shown in Equation 6-9.

```
prop_valid() ->
  ?FORALL(
    G, grammar(),
    complete(G) == true).
```

Equation 6-9: Valid grammar check

The invalidity of grammar is introduced by adding an invalid sentence to the valid grammar in random pattern (start/middle and end of the generated grammar). In this way we simulate a fault due to any kind of corruption in an existing valid grammar. The invalid grammar check is shown in Equation 6-10.

```
prop_invalid() ->
 ?LET({{A1,A2},G}, {non_equal_atoms(),grammar()},
 ?FORALL(
   G1, shuffle(G++[{A1,[A2]}]),
     complete(G1) == [{A2,false}])).
```

Equation 6-10: Invalid grammar check

6.3.2.2.3 Analysis

In section 6.3.2.2.1 we have integrated a simple test that confirms the well-formed grammar and checks if the grammar is complete or not. In section 6.3.2.2.2 we have developed a property based test bed for generating the grammar with all the inputs and tested it with two properties, i.e. valid and invalid, against the parser's functionality.

It was observed that even though we handled the subtle and obvious bugs some of following bugs still remained, specifically.

- The definition of any level of the grammar was not handling the case where grammar level was part of the list containing a relationship to another level of the grammar. (See section 6.3.2.2.2)
- Definitions for repeated sentences were not tested, and
- An empty grammar, even though it does not make sense, was not included in the testing of parser.

Currently in the LAB setup there exist 49 logical bindings for 397 PIDS. The biggest device configuration, in terms of interconnection, has 32 interconnections. We multiply all of the current values (number of logical bindings, PIDs and interconnections) by a factor of 2 to estimate future[*] state of the grammar. The size of the test cases was selected to be 500 and no errors were observed when testing both valid and invalid grammars.

The results from the testing are satisfactory and no bugs were found. We do not claim that the currently implementation is completely free of bugs, but we have attempted to perform fuzzy testing in order to reach satisfactory level of testing for the implemented solution.

6.3.3 Performance testing

In the previous section, we evaluated the functional aspects of our implementation. Nonetheless, the non-functional (responsiveness and concurrency) quality testing is also very important. For example, Weng et al. [84] stated following a survey conducted by Microsoft's

[*] Future, expectation for a year from now (See section 6.3.1).

Office development team that user complaints about what they perceived as bad performance were almost as frequent as complaints about crashes. Perceptible performance or responsiveness of an interactive tool can be described in terms of its latency of when handling events[42]. A tool with low responsiveness (i.e. high latency of handing events) is likely to induce anger, frustration, and annoyance of its user[43] and can also greatly negatively affect user productivity[44]. That is why Milan Jovic and Matthias Hauswirth[45] consider perceptible performance testing as an important part of the evaluation of an interactive application.

We did not perform tests for the concurrency of our implemented solution, as tests have been performed which compared yaws and apache concurrency[46]. As the test results for concurrency of yaws indicate that yaws is able to handle 80,000 parallel sessions and that is much greater than the current number of users of PeppesBodega-TF. A second reason for not performing concurrency tests is that concurrency measurements takes a lot of time (and it was not a core focus of this Master's thesis) as many factors are involved in its calculation (such as available processing power, latency, RAM, bandwidth, number of sockets etc).

In this section we will examine perceptible performance (i.e. specifically the latency of handling events, such as the creation of a new configuration, loading, editing, etc.) of PeppesBodega-TF. Additionally, this evaluation will help us in identifying performance bottlenecks. The detection of performance bottlenecks is very important as it enables the developer(s) to identify the part(s) of the system that is critical for improving the overall performance (by modifying the identified part(s)) of the system to remove the bottleneck.

As Shneiderman [44] found the threshold of responsiveness to be around 100 ms for a single event, therefore we are considering 100 ms as our threshold of responsiveness (i.e. if handling of an event takes more than 100 ms then the implementation fails with respect to the responsiveness criteria).

For evaluation, we will use the fprof [*] module to dump Erlang function calls (as the implementation language is Erlang). The best feature of this profiling module is that it also calculates time taken by its *own* function call, thus giving accurate measurements. The fprof profiles any given function in three steps:

1. The **tracer** provides information about all the called functions, execution time, processes scheduling, etc.
2. The **profiler** reads the trace files, simulates the execution call stack and calculates raw profile data from this execution stack.
3. The **analyzer** sorts and filters the raw profile and then converts the output into a readable text file.

The output of the 'fprof' results in the three metrics:

CNT	CNT is the total number of function calls during the trace.
ACC	ACC is the accumulated time from the start of the trace to the end of the trace.

[*] fprof is a profiler module that collects and analyzes the statistics about the execution of an Erlang function. http://www.erlang.org/doc/man/fprof.html Last visited: 2013-12-10

OWN	OWN is the sum of the execution time of the functions found in the trace *excluding* the called functions, i.e., the time spent doing the profiling.

We are mainly interested in the difference between ACC and OWN because this indicates the total execution time of the actions performed by PeppesBodega-TF. We will first observe whether the execution time has a predictable or random pattern. If it has a predictable pattern, then we will only analyze the limits of the dataset in our analysis.

Following are the frequent operations that are to be executed by the users of PeppesBodega-TF.

1. Create new configuration
2. Transactions of devices to configuration
3. Remove configuration

6.3.3.1 Create new configuration

Today each STP has a configuration file associated to it which defines the STP interconnections. So we will analyze the implemented solution for the creating at least the same number of configuration as we have forecast for the number of STPs. We have executed the profile for 1000 configurations (more than the 750 STPs forecast for the next year).

Table 6-1 shows the trend of CNT, ACC, and OWN for 5 executions of creation of 100, 200, 300,… 1000 configurations. We observed that the time for creation of a configuration increase as the number of existing configurations in the Mnesia database is increased. This is normal behavior because the current implementation checks if the same key is used if so the record is updated rather than creating a new record. So increasing existing configurations increases the time required for creating next configuration.

We do not consider the update function to be bottleneck for the creation of new configuration because it gives us a nice way to update the existing configuration when we want to add/delete a device to a configuration, update connections between devices, etc.

Table 6-1 shows a calculation of the maximum, minimum, and deviation of CNT, ACC, and OWN for the 5 executions of creating the different numbers of configurations. The maximum time is the most relevant as the users are mostly concerned with the maximum time that they have to wait for the output of a result after they have entered a request. The minimum of each is used to calculate the expected deviation. We observe that for all three criteria CNT, ACC, and OWN the deviation is lower than 100ms, hence there is no indication of any performance bottle-neck.

The fitting curves for Figure 6-6 and Figure 6-7 are linear so we will calculate the maximum time required for the creation of 901-1000 configurations that also includes the time to check at-least 900 configuration records for the same key as the requested key for 901-1000 configurations.

Create Configurations	1			2			3			4			5			MAX			MIN			DEVIATION		
	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN
100	24457	450	356	24457	447	353	24454	440	363	24449	435	373	24446	440	384	24457	450	384	24446	435	353	11	15	31
200	24543	480	369	24553	473	365	24552	473	369	24548	467	380	24561	476	373	24561	480	380	24543	467	365	18	13	15
300	24663	507	490	24677	516	488	24684	528	501	24690	537	508	24689	552	510	24690	552	510	24663	507	488	27	45	22
400	24772	660	567	24783	657	579	24790	672	588	24800	679	597	24794	678	602	24800	679	602	24772	657	567	28	22	35
500	24891	780	615	24898	776	616	24896	775	624	24893	784	637	24898	782	632	24898	784	637	24891	775	615	7	9	22
600	24981	861	725	24987	867	735	24988	880	740	24993	881	753	24993	886	753	24993	886	753	24981	861	725	12	25	28
700	25089	998	815	25082	992	814	25078	1007	817	25087	1020	815	25084	1027	823	25089	1027	823	25078	992	814	11	35	9
800	25217	1077	909	25212	1077	904	25218	1070	907	25231	1071	904	25225	1082	897	25231	1082	909	25212	1070	897	19	12	12
900	25334	1042	975	25332	1050	973	25335	1051	984	25348	1054	987	25358	1053	982	25358	1054	987	25332	1042	973	26	12	14
1000	25405	1252	1014	25405	1253	1029	25409	1259	1043	25411	1255	1037	25413	1254	1048	25413	1259	1048	25405	1252	1014	8	7	34

Table 6-1: fprof profile for create configuration

Total time for calculations = ACC - OWN

= 1259 - 1048

= 211 ms

Number of calculations = Creation of new configurations

= 100

Responsiveness of a single operation = Total time for calculation / number of calculations

= 211 / 100

= 2.11 ms

As 2.11ms is less than 100 ms the implemented solution will be able to successfully create 1000 configurations (and a lot more) within the threshold of perceptibility as found by Shneiderman in [44], hence we consider the system sufficiently responsive.

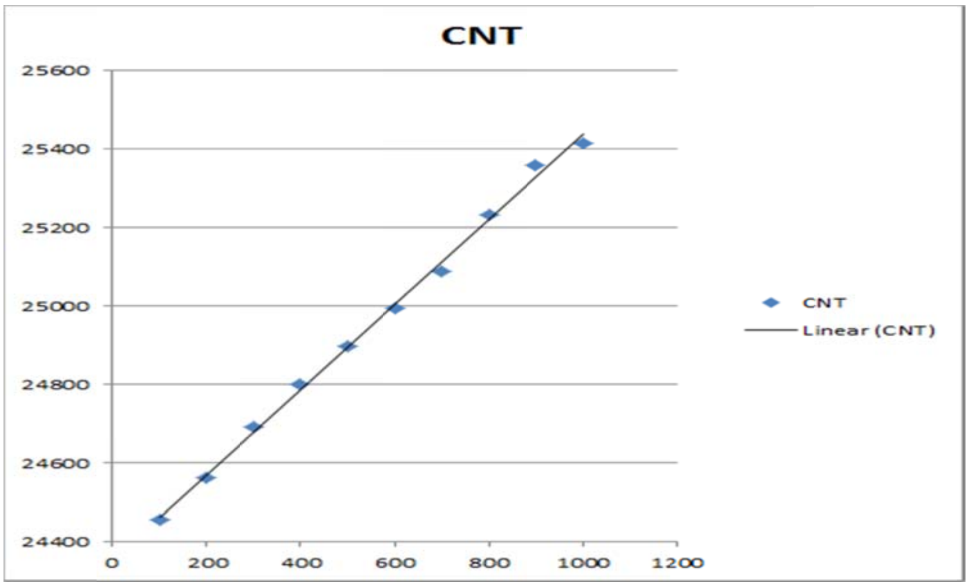


Figure 6-5: Create configuration - max CNT

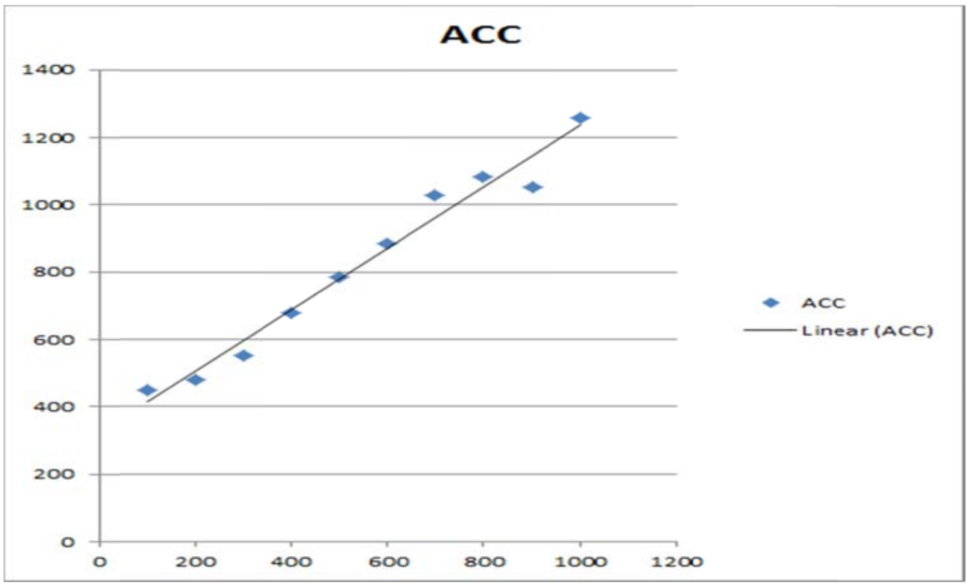


Figure 6-6: Create configuration - max ACC

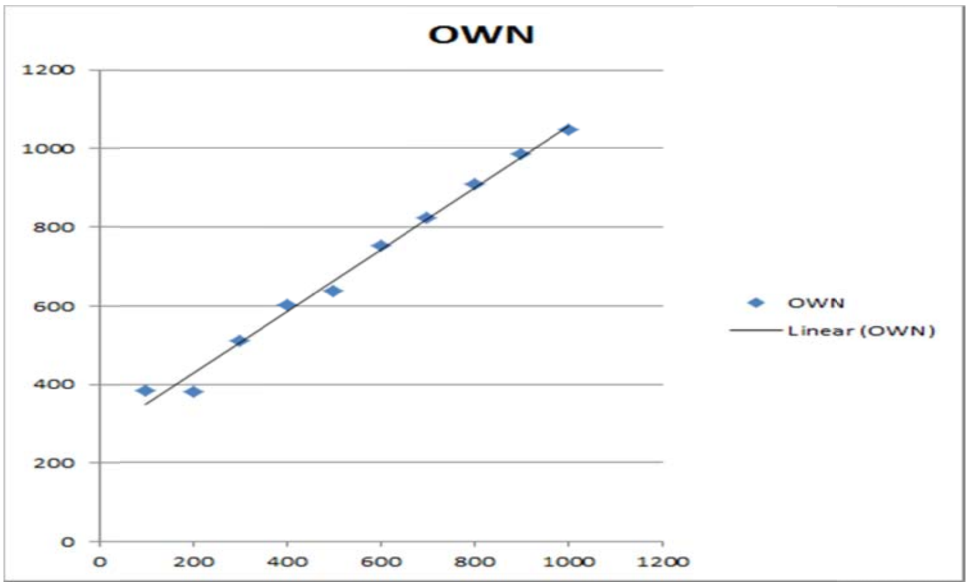


Figure 6-7: Create configuration - max OWN

6.3.3.2 Transactions of devices to configuration

There are two main data entities within PeppesBodega-TF that should affect the performance evaluation of the implemented solution for transaction of devices in configuration. These data entities for which variance can lead to interesting results are: Devices and Configurations. For this reason, the dataset has been divided into two subsets:

1. Constant number of configurations with variable number of devices and
2. Variable number of configurations with constant number of devices.

The performance tests for above mentioned two data-sets have been performed in two time intervals. In first interval the performance tests were executed only once and in second interval performance test were executed five times to check the variance between different executions.

6.3.3.2.1 Constant configuration with variable devices

Table 6-2 shows a profile for dataset-1; first interval, where the total number of configurations (storing topological information) was constant and we made addition, modification, and deletion transactions of varying devices to these configurations. Table 6-3 shows second interval for performing same operation as mentioned above for Table 6-2.

The results from the 'fprof' for dataset-1, first interval, are plotted in Figure 6-8, Figure 6-9 and Figure 6-10. The results from 'fprof' for dataset-1, second interval, are plotted in Figure 6-11, Figure 6-12 and Figure 6-13.

As Table 6-2 and Table 6-3 show almost similar execution times for execution, but Table 6-3 contains values for the execution for 5 times. As mentioned earlier in this performance testing the expected behavior is observed, thus we test the highest limits as users are most concerned about the maximum time for the output. So we will consider the maximum value for the largest configuration, i.e. last row of Table 6-3.

Total time for calculations = ACC - OWN

= 203326 - 132665
= 70661 ms

Number of calculations = Number of devices/configurations * Total configurations * number of transactions/configuration:

= 100*49*3
= 14 700

Responsiveness of a single operation = Total time for calculation / number of calculations

= 70661 / 14 700
= 4.8068707 ms

As the maximum responsiveness (4.8 ms) is less than 100 ms the implemented solution (with dataset-1 as input and future forecasted values of LAB equipment) is well below the threshold of perceptibility as found by Shneiderman in [44], hence we consider the system sufficiently responsive with regard to making changes to 49 different configurations.

Table 6-2: fprof profile for ‘Transaction of devices to configuration (dataset-1)'

Number of devices / configuration	Total number of configurations	CNT	ACC	OWN
10	49	637995	21606	14166
20	49	1169233	42902	27666
30	49	1700776	62454	40876
40	49	2232170	79881	51721
50	49	2763557	99562	65276
60	49	3294941	117732	76104
70	49	3826412	137584	89952
80	49	4357728	156153	101684
90	49	4888647	171544	112029
100	49	53954128	195486	129061

devices / 100	Total number of	1			2			3			4			5			MAX			MIN			DEVIATION		
configurations	configurations	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN
10	49	649584	22668	15268	651090	24127	16347	652768	25798	17618	654501	27484	19570	655512	29259	21481	655512	29259	21481	651090	24127	16347	4422	5132	5134
20	49	1071125	44585	29555	1072463	46510	31263	1074151	47741	32847	1076036	49367	34180	1077544	50485	35969	1077544	50485	35969	1072463	46510	31263	5081	3975	4706
30	49	1697500	64241	42137	1698578	65961	43538	1700115	67469	45030	1701637	69166	46274	1703387	70964	48235	1703387	70964	48235	1698578	65961	43538	4809	5003	4697
40	49	2241841	80916	53186	2243168	82714	55138	2244184	84393	56191	2245602	86041	57293	2247037	87974	58428	2247037	87974	58428	2243168	82714	55138	3869	5260	3290
50	49	2705132	100660	66409	2706415	102575	67522	2707751	103608	68991	2708979	105268	70104	2710377	106829	71214	2710377	106829	71214	2706415	102575	67522	3962	4254	3692
60	49	3248412	119694	77292	3249894	121249	79064	3251225	122548	80804	3253213	123580	82150	3254402	125123	83933	3254402	125123	83933	3249894	121249	79064	4508	3874	4869
70	49	3841252	139005	91252	3842655	140500	93144	3843694	141599	94483	3845210	142710	95567	3846318	144239	97534	3846318	144239	97534	3842655	140500	93144	3663	3739	4390
80	49	4278412	157769	103088	4280238	158783	104702	4281878	160487	105872	4283681	162310	107781	4284786	164119	109374	4284786	164119	109374	4280238	158783	104702	4548	5336	4672
90	49	4784152	172965	113418	4785728	174540	115056	4786827	176021	116976	4788189	177332	118947	4789781	178994	120927	4789781	178994	120927	4785728	174540	115056	4053	4454	5871
100	49	5274415	197142	127105	5275523	198166	128592	5277218	199401	130427	5279195	200949	131554	5280335	202470	133108	5280335	202470	133108	5275523	198166	128592	4812	4304	4516

Table 6-3: fprof profile for 'Transactions of devices to configuration (dataset-1)'

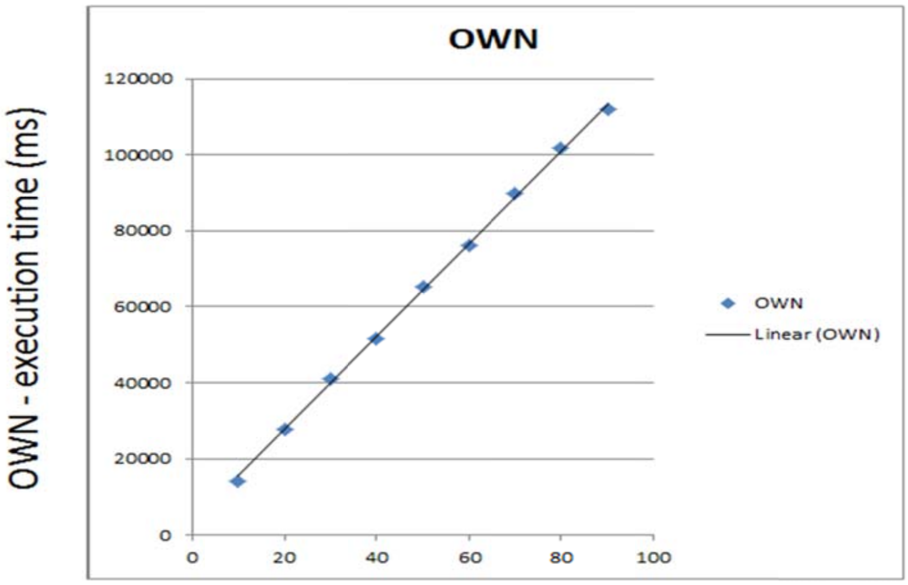


Figure 6-8: Transaction of devices to configuration (dataset-1) - OWN with 49 devices

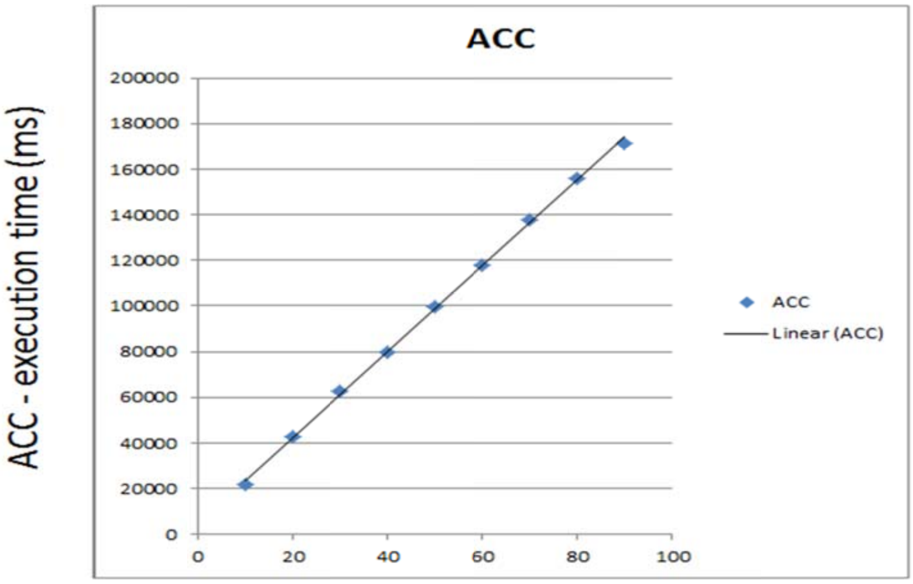


Figure 6-9: Transaction of devices to configuration (dataset-1)- ACC with 49 devices

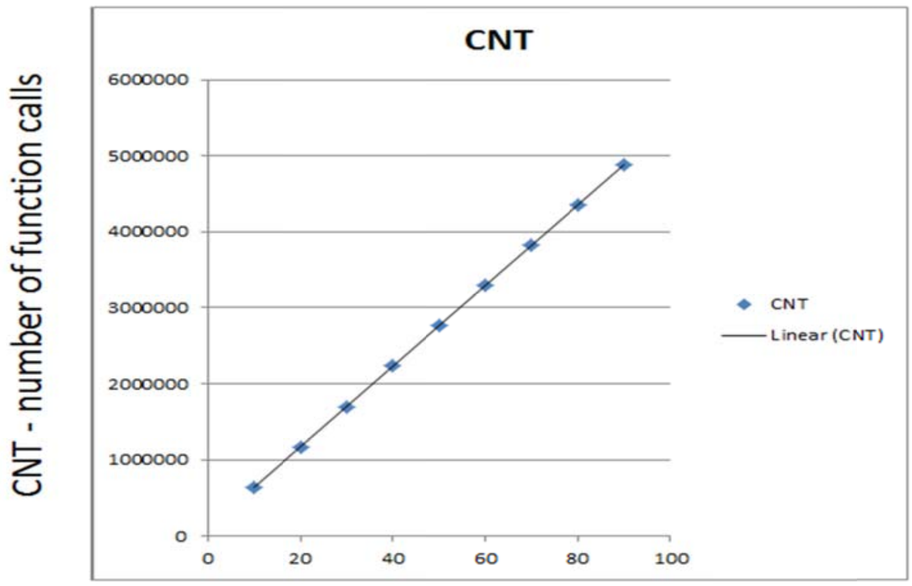


Figure 6-10: Transaction of devices to configuration (dataset-1) - CNT with 49 devices

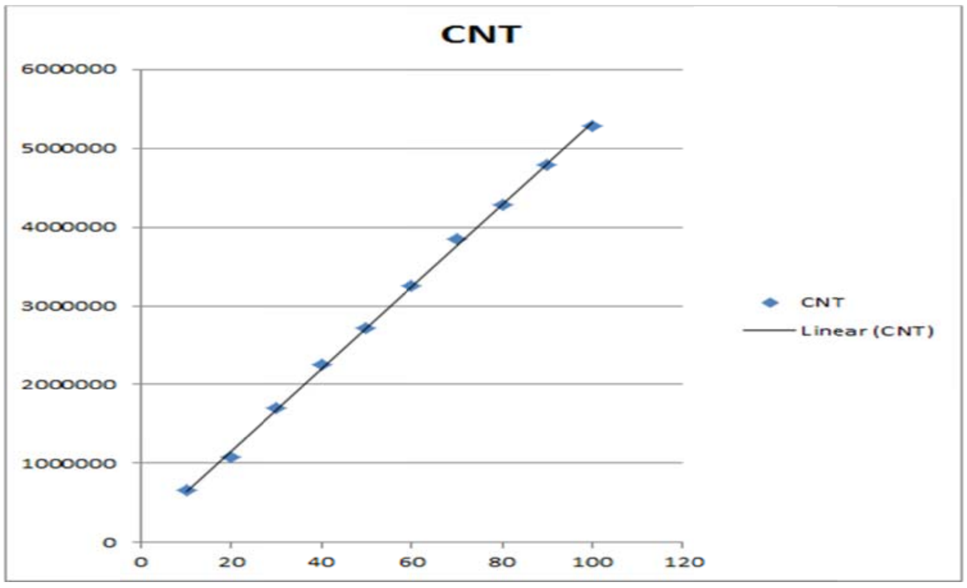


Figure 6-11: Transaction of devices to configuration (dataset-1) – max CNT with 49 devices

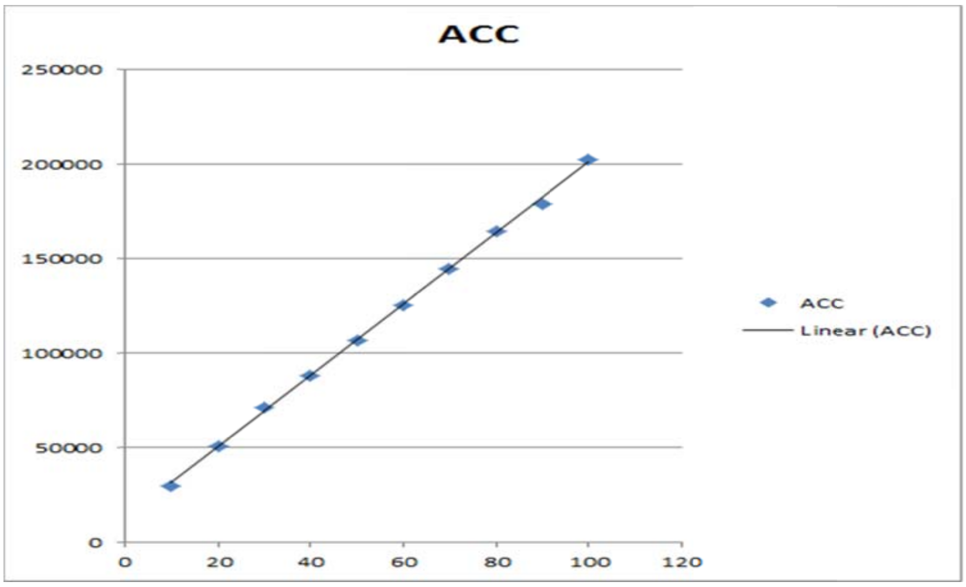


Figure 6-12: Transaction of devices to configuration (dataset-1) – max ACC with 49 devices

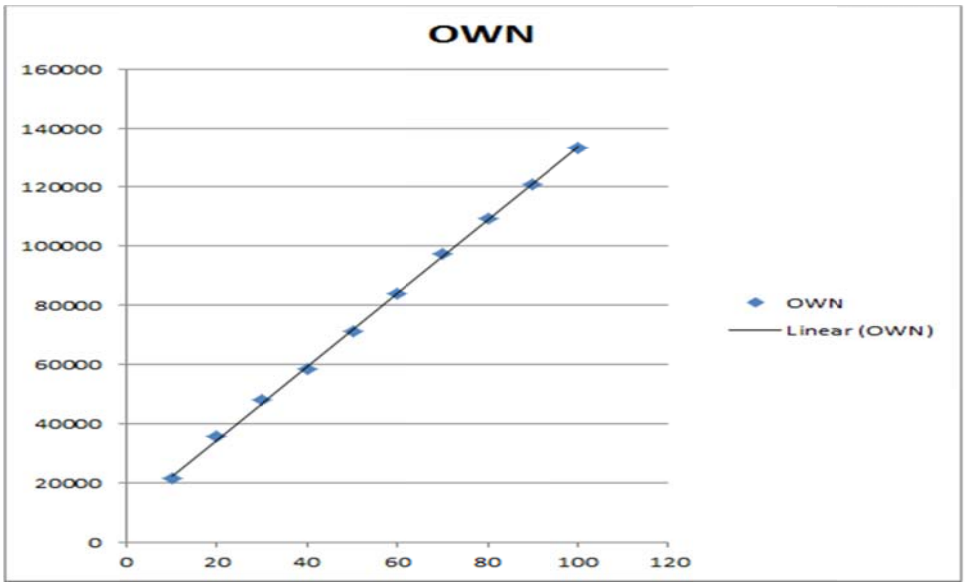


Figure 6-13: Transaction of devices to configuration (dataset-1) - max OWN with 49 devices

6.3.3.2.2 *Variable number of configurations with a constant number of devices*

Table 6-4 shows a profile for dataset-2, first interval, where the total number of devices used in addition, modification, and deletion transactions were kept constant, but these actions were performed on different numbers of configurations. For example, the first entry of Table 6-4 shows the case for 10 configurations where each configuration has 100 devices added, modified, and deleted from all 10 configurations during the performance measurement. Table 6-5 shows second interval for performing same operation as mentioned above for Table 6-4.

The results from the 'fprof' for dataset-2, first interval, are plotted in Figure 6-14, Figure 6-15 and Figure 6-16. We observe a sudden rise for OWN and ACC in the case of 40 configurations. However, if we consider the difference between both OWN and ACC, we see that the execution time of the application is increasing linearly; hence this sudden rise could be due to a high load on the processor by another running task. We have not observed this kind of observation for any other of the performance test results but this is quite normal behavior. The results from 'fprof' for dataset-2, second interval, are plotted in Figure 6-17, Figure 6-18 and Figure 6-19.

Once again since the graphs are linear we will make our calculation based upon the last row of the Table 6-5.

Total time for calculations = ACC - OWN

= 361046 - 252688
= 108358 ms

Number of calculations = Number of devices/configurations * Total configurations * number of transactions/configuration:

= 100*100*3
= 30 000

Responsiveness of single operation = Total time for calculation / number of calculations

= 108358 / 30 000
= 3.611933 ms

As the responsiveness (3.6 ms) is less than 100 ms the implemented solution (with dataset-2 as input) falls below the threshold of perceptibility as found by Shneiderman in [44], hence we consider the system sufficiently responsive with regard to making changes to these different numbers of configurations with 100 devices.

In Table 6-4 and Table 6-5 the deviation from maximum and minimum CNT,ACC and OWN is quite low so we do not consider it to indirect cause of any bottleneck (that might occur in future).

Table 6-4: fprof profile for 'Transaction of devices to configuration (dataset-2)'

Number of devices / configuration	Total number of configurations	CNT	ACC	OWN
100	10	1062931	39449	25465
100	20	2125871	76837	50353
100	30	3188302	116134	76174
100	40	4251012	201747	128768
100	50	5313685	193112	127494
100	60	6376860	227639	148345
100	70	7439321	267706	201595
100	80	8502174	305522	222783
100	90	9565127	336117	235506
100	100	10587549	378754	244069

devices / configuration	Total number of configurations	1			2			3			4			5			MAX			MIN			DEVIATION		
		CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN
100	10	1064276	40866	27134	1065998	42176	28929	1067451	44121	30544	1068877	45419	31585	1070718	46502	33519	1070718	46502	33519	1065998	42176	28929	4720	4326	4590
100	20	2127029	78223	51382	2128869	79392	52637	2130630	80702	54144	2132609	82250	56097	2134099	83543	57511	2134099	83543	57511	2128869	79392	52637	5230	4151	4874
100	30	3189312	117786	77890	3190950	119429	79509	3192917	121053	81307	3194282	122516	83187	3195833	123740	84953	3195833	123740	84953	3190950	119429	79509	4883	4311	5444
100	40	4252782	174419	120379	4254585	176101	121385	4256525	177596	122424	4258063	178955	124300	4259600	180705	125947	4259600	180705	125947	4254585	176101	121385	5015	4604	4562
100	50	5315495	203480	130269	5316693	205115	132052	5318633	206263	133332	5320532	207460	134762	5321989	209441	136036	5321989	209441	136036	5316693	205115	132052	5296	4326	3984
100	60	6378116	229339	150139	6380105	231122	151569	6381534	232991	152572	6382988	234754	154096	6384072	236742	155174	6384072	236742	155174	6380105	231122	151569	3967	5620	3605
100	70	7441168	269495	203511	7442559	270854	204715	7444511	272660	205762	7446250	273955	207286	7447776	275830	208775	7447776	275830	208775	7442559	270854	204715	5217	4976	4060
100	80	8503844	307263	224202	8505664	308666	225338	8506877	309969	226592	8507902	311957	228270	8509848	313157	229469	8509848	313157	229469	8505664	308666	225338	4184	4491	4131
100	90	9566157	338080	237359	9567404	339894	238923	9568877	341349	240078	9570301	342727	241469	9571949	344090	243176	9571949	344090	243176	9567404	339894	238923	4545	4196	4253
100	100	10454812	354812	247485	10455813	355983	248540	10457211	357418	250214	10459041	359243	251759	10460884	361204	253597	10460884	361204	253597	10455813	355983	248540	5071	5221	5057

Table 6-5: fprof profile for ‘Transaction of devices to configuration (dataset-2)

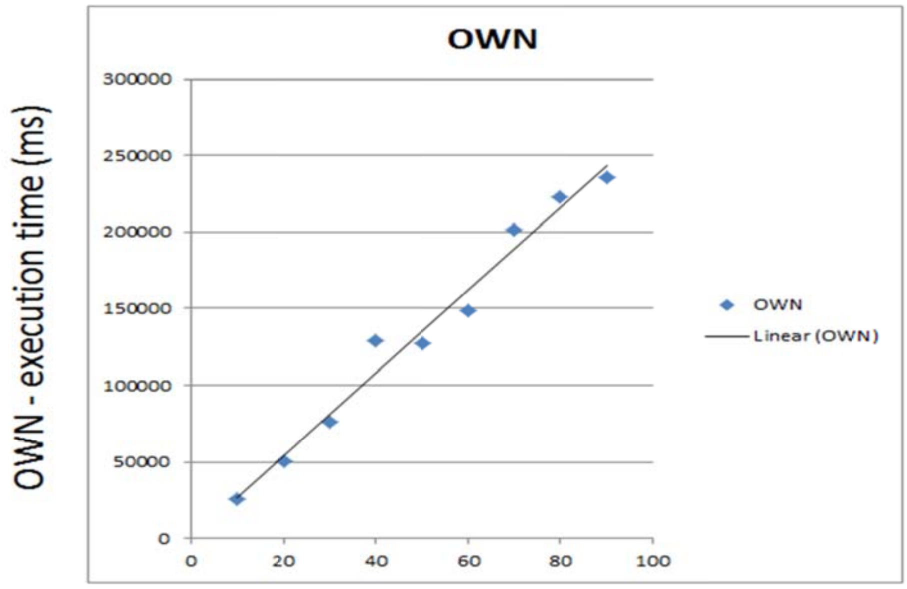


Figure 6-14: Transaction of devices to configuration (dataset-2) - OWN with 100 configurations

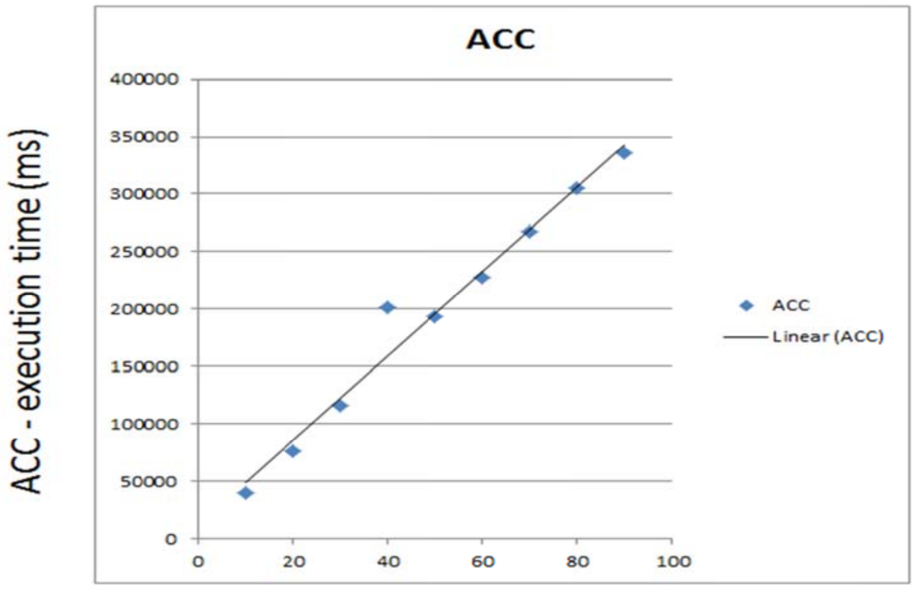


Figure 6-15: Transaction of devices to configuration (dataset-2) - ACC with 100 configurations

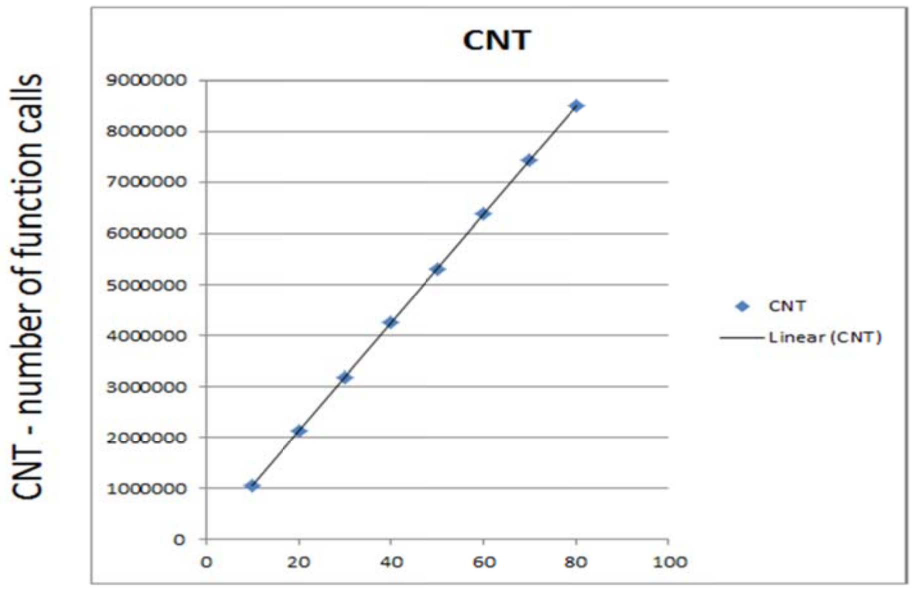


Figure 6-16: Transaction of devices to configuration (dataset-2) - CNT with 100 configurations

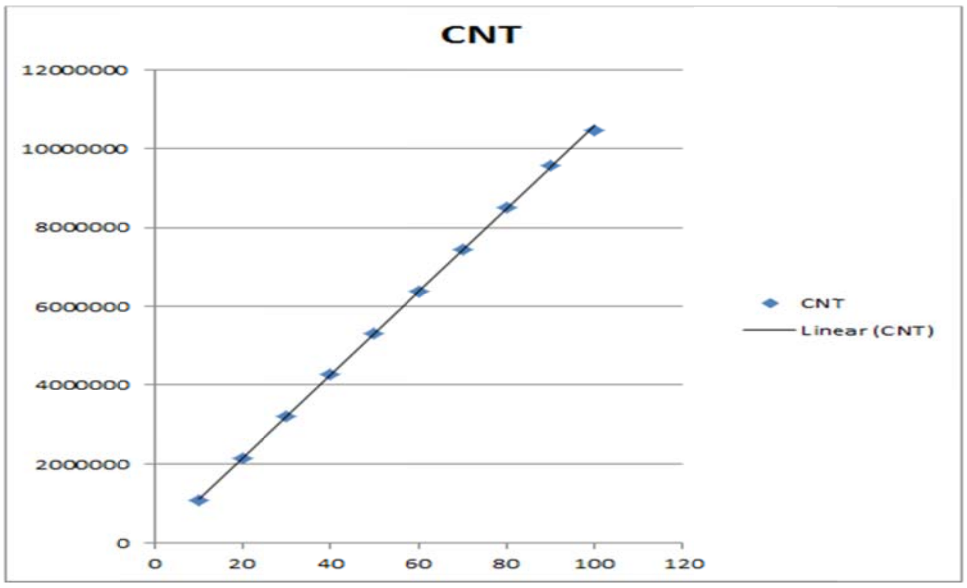


Figure 6-17: Transaction of devices to configuration (dataset-2) - max CNT with 100 configurations

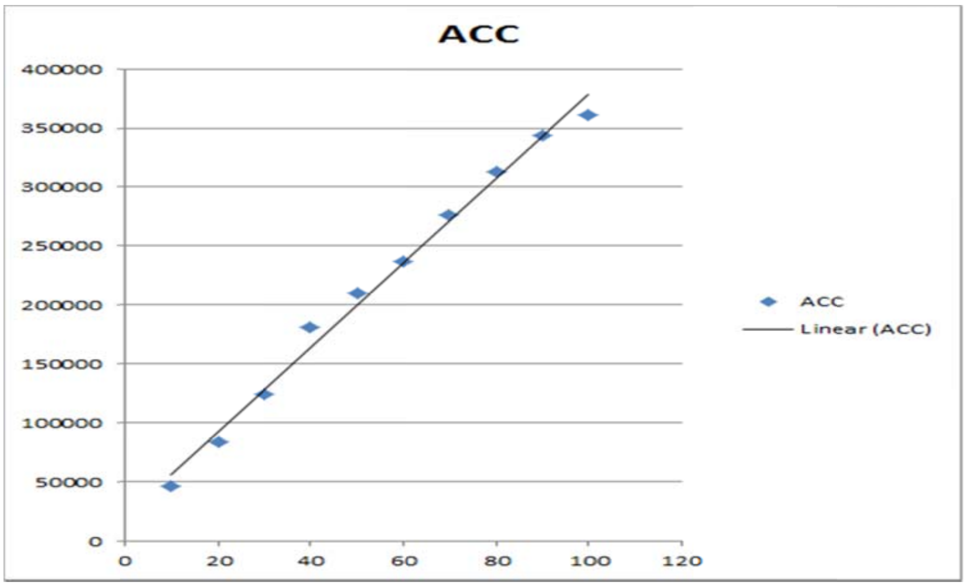


Figure 6-18: Transaction of devices to configuration (dataset-2) – max ACC with 100 configurations

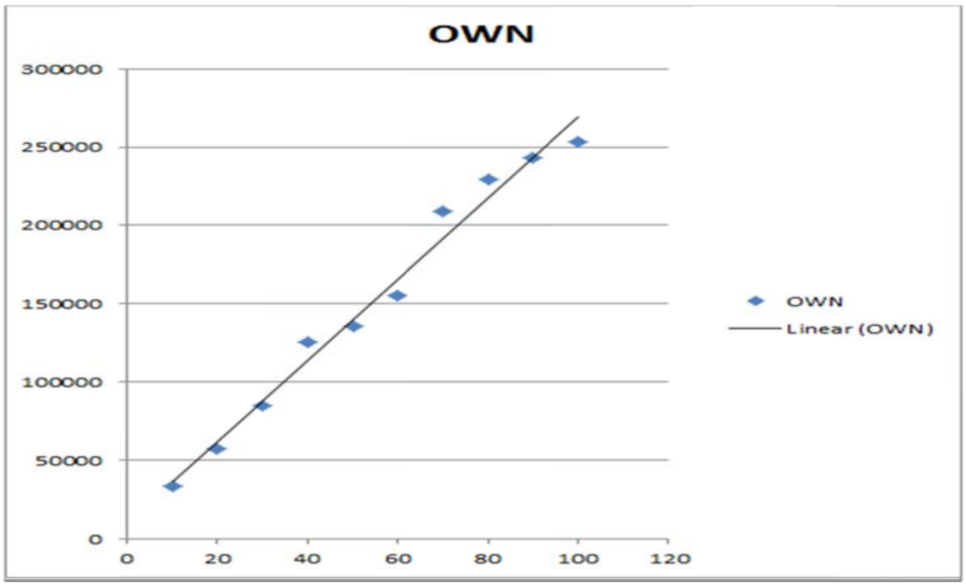


Figure 6-19: Transaction of devices to configuration (dataset-2) - max OWN with 100 configurations

6.3.3.3 Remove configuration

In section 6.3.3.1 we added 1000 configurations, and analysed the performance of addition of the configuration. In this section we will perform deletion of the configuration and observe the performance of deletion operation of configuration. In this section we have performed deletion of devices 1-100,101-200,…900-1000. As the number of devices that were deleted were the same so we were expecting that we will have a constant time with some deviation. But our results proved our initial assumption to be wrong.

Table 6-6 shows profile for deletion of configuration. Row-1 of Table 6-6 indicates the values of ACC, OWN and CNT for deletion of 1-100 configurations from Mnesia database. We are deleting constant number of configurations in each step and observe a linear decrease of ACC and OWN value with CNT being constant as shown from fitting curves in Figure 6-20, Figure 6-21 and Figure 6-22. The reason is that when the first 100 delete configuration request is performed the Mnesia database has to traverse all the keys of the 1000 configurations; later-on the time will decrease as the size of database decreases.

For maximum time of ACC and CNT, we will perform addition

Total ACC_{max}	= 27460 ms
Total OWN_{max}	= 11072 ms
Total time for calculation	= Total ACC_{max} - Total OWN_{max}
	= 27460 – 11072
	= 16388 ms
Number of calculations	= 1000
Responsiveness of single operation	= Total time for calculation / number of calculations
	= 16388 / 1000
	= 16.388 ms

This value is higher than that from creation and transaction of devices for the configuration, i.e. the calculations in sections 6.3.3.1 and 6.3.3.2 (respectively). The sole reason for this increased time for a single deletion of configuration is that we are traversing all the devices to delete the references to the deleted configuration. The responsiveness (16.388 ms) is still less than 100 ms, so the implemented solution (with dataset-2 as input) falls below the threshold of perceptibility as found by Shneiderman in [44]. Hence we consider the system sufficiently responsive with regard to deletion of configurations with 100 (or more) devices.

Delete Configurations	1			2			3			4			5			MAX			MIN			DEVIATION		
	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN	CNT	ACC	OWN
1-100	97661	3872	1841	97665	3867	1845	97664	3872	1850	97663	3868	1849	97664	3864	1850	97665	3872	1850	97661	3864	1841	4	8	9
101-200	87646	3623	1708	87647	3619	1712	87646	3618	1717	87646	3618	1716	87643	3615	1712	87647	3623	1717	87643	3615	1708	4	8	9
201-300	77639	3528	1571	77642	3528	1574	77647	3525	1576	77644	3529	1581	77639	3533	1581	77647	3533	1581	77639	3525	1571	8	8	10
301-400	67632	3376	1441	67633	3380	1445	67637	3380	1448	67640	3384	1451	67642	3387	1455	67642	3387	1455	67632	3376	1441	10	11	14
401-500	57647	2892	1240	57649	2891	1238	57646	2895	1238	57643	2890	1233	57644	2885	1232	57649	2895	1240	57643	2885	1232	6	10	8
501-600	47636	2735	1048	47640	2736	1052	47636	2732	1048	47636	2732	1049	47631	2731	1046	47640	2736	1052	47631	2731	1046	9	5	6
601-700	37642	2514	909	37638	2510	914	37643	2515	919	37643	2518	922	37638	2523	920	37643	2523	922	37638	2510	909	5	13	13
701-800	27638	1868	629	27642	1872	633	27646	1872	634	27650	1870	639	27649	1867	638	27650	1872	639	27638	1867	629	12	5	10
801-900	17622	1597	438	17618	1601	438	17617	1601	442	17621	1600	441	17626	1597	442	17626	1601	442	17617	1597	438	9	4	4
901-1000	7606	1433	206	7602	1437	208	7605	1439	208	7605	1437	207	7605	1436	202	7606	1439	208	7602	1433	202	4	6	6

Table 6-6: prof profiler for delete configurations

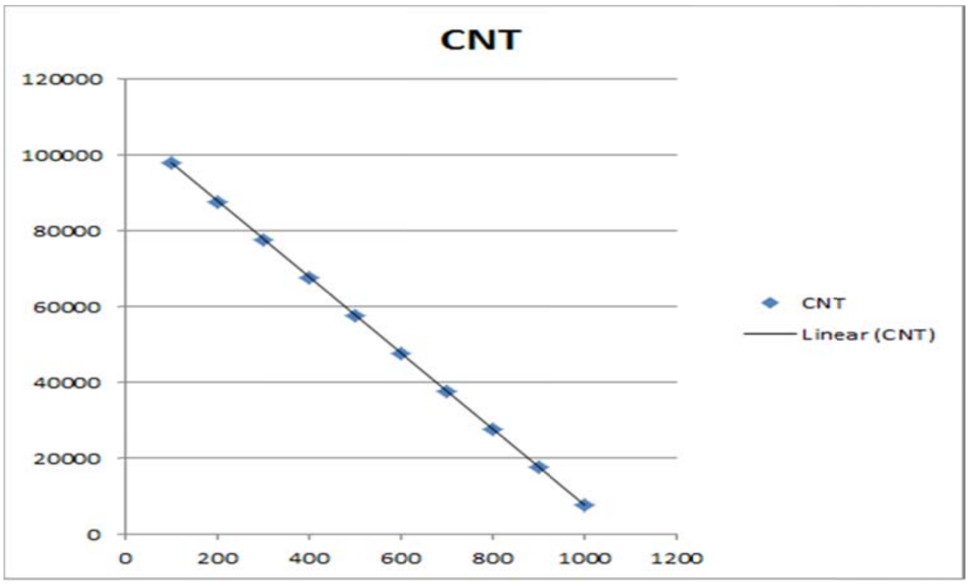


Figure 6-20: Delete configuration - max CNT

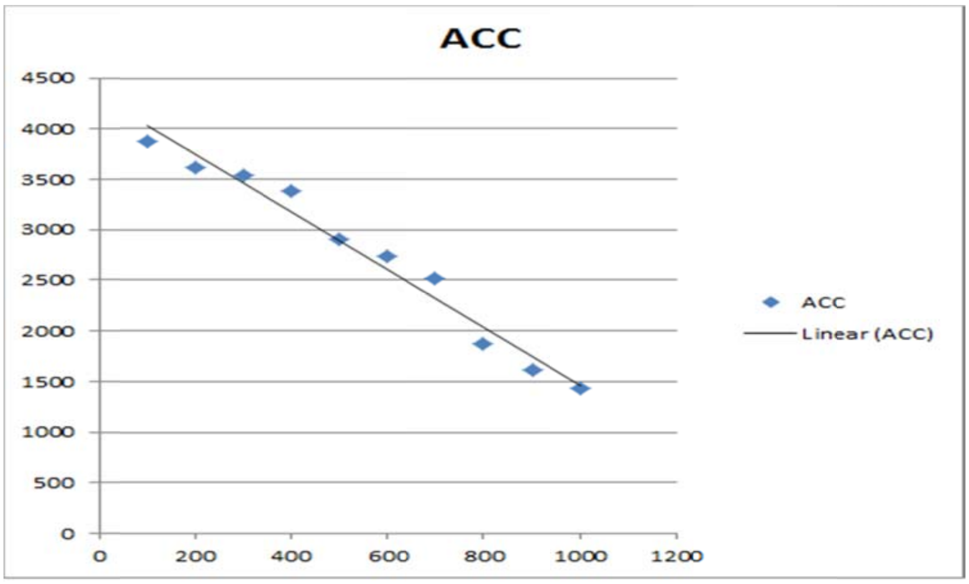


Figure 6-21: Delete configuration - max ACC

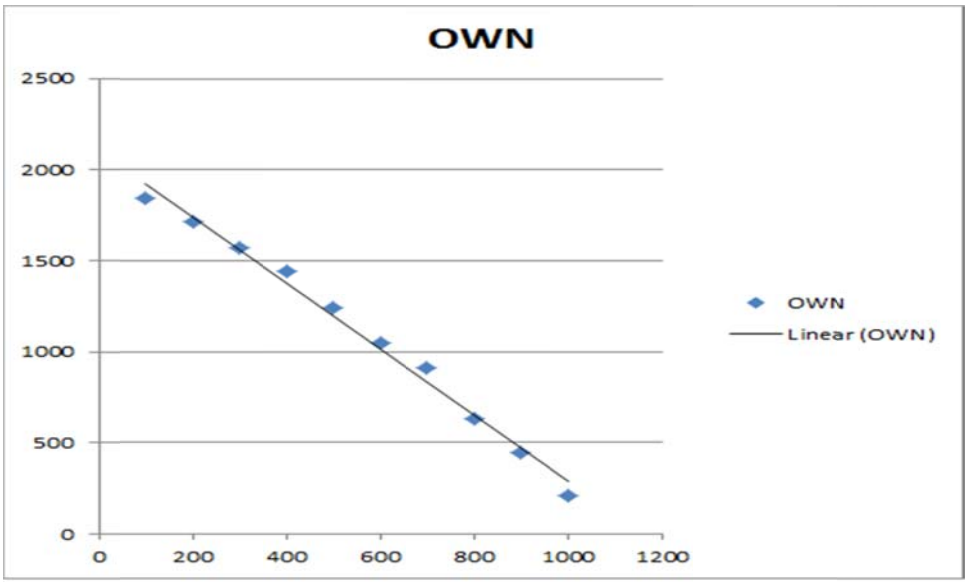


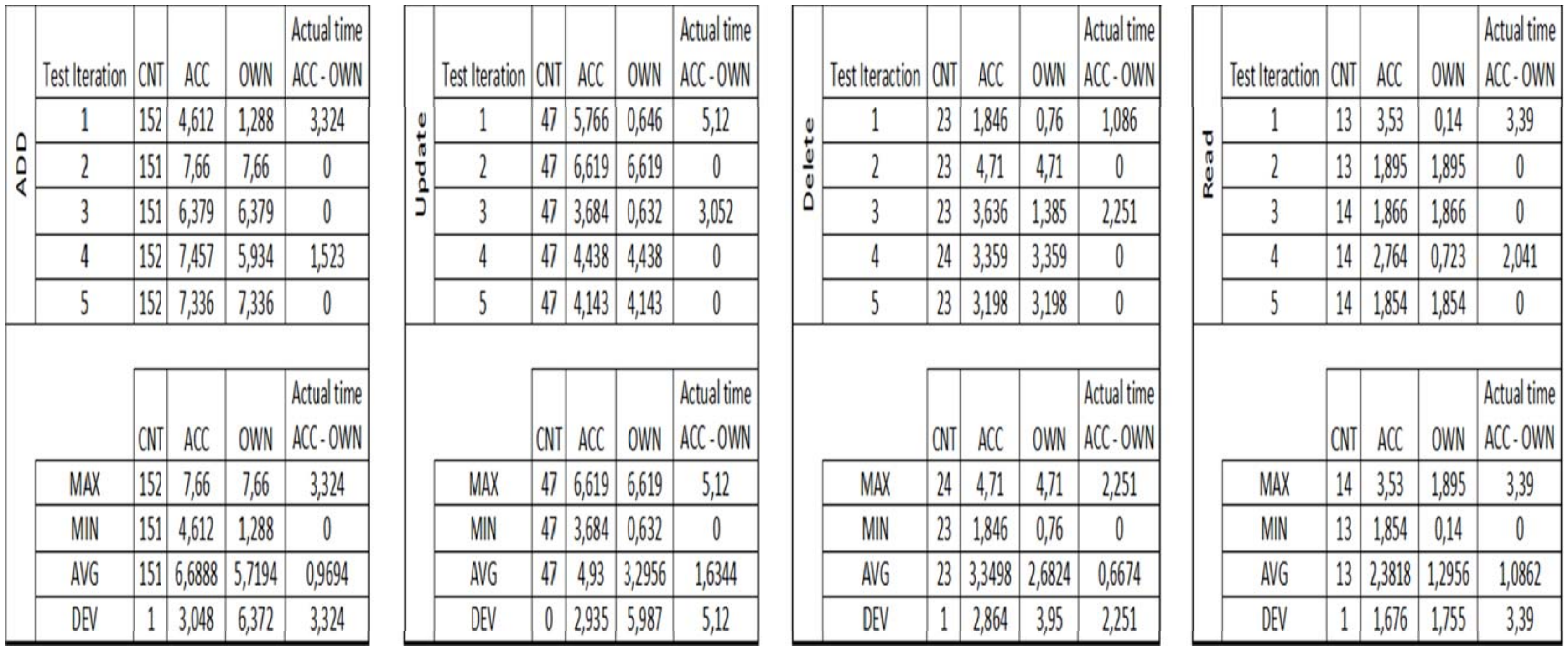
Figure 6-22: Delete configuration - max OWN

6.3.3.4 Summary of performance measurements

The maximum time to perform a transaction in all above cases (sections 6.3.3.1, 6.3.3.2, and 6.3.3.3) is not more than 16.388 ms.

The transactions perform read and write operations on the Mnesia database and some validation checks (See section 5.2.2.3 and section 6.3.2.1). As mentioned in section 4.2.6.4, the Mnesia database in PeppesBodega-RMS is using Dets tables (that is disk storage), while the remaining transactions are being done in the Erlang runtime system (that resides in the RAM) so obviously the major portion of the transaction's time in PeppesBodega-TF is due to the Mnesia database query operations. We will therefore perform a simple test which will perform common queries (add, update, delete, and read) to Mnesia database to calculate average time of each query. In order to understand the time taken for each transaction we have to consider the semantics of a transaction in the Mnesia database.

As discussed earlier, in each transaction queries to the database (requiring an update of the disk) take more time than other operations in transaction. Figure 6-23 show the maximum, minimum and average time for queries (add, read, update and delete) for given rotational speed of the disk. The calculation of this time was also performed by fprof. The average time for add, read, update and delete queries is 0.9694ms, 1.0862ms, 1.6344ms and 0.6674ms respectively. We verified this estimate by writing a simple program to perform queries on the database. Based upon our analysis we conclude that the performance of systems is dominated by operations of database, and as it lies within the threshold of perceptibility as found by Shneiderman in [44] hence no bottlenecks occur.



ADD	Test Iteration	CNT	ACC	OWN	Actual time ACC - OWN
	1	152	4,612	1,288	3,324
	2	151	7,66	7,66	0
	3	151	6,379	6,379	0
	4	152	7,457	5,934	1,523
	5	152	7,336	7,336	0

	CNT	ACC	OWN	Actual time ACC - OWN
MAX	152	7,66	7,66	3,324
MIN	151	4,612	1,288	0
AVG	151	6,6888	5,7194	0,9694
DEV	1	3,048	6,372	3,324

Update	Test Iteration	CNT	ACC	OWN	Actual time ACC - OWN
	1	47	5,766	0,646	5,12
	2	47	6,619	6,619	0
	3	47	3,684	0,632	3,052
	4	47	4,438	4,438	0
	5	47	4,143	4,143	0

	CNT	ACC	OWN	Actual time ACC - OWN
MAX	47	6,619	6,619	5,12
MIN	47	3,684	0,632	0
AVG	47	4,93	3,2956	1,6344
DEV	0	2,935	5,987	5,12

Delete	Test Iteraction	CNT	ACC	OWN	Actual time ACC - OWN
	1	23	1,846	0,76	1,086
	2	23	4,71	4,71	0
	3	23	3,636	1,385	2,251
	4	24	3,359	3,359	0
	5	23	3,198	3,198	0

	CNT	ACC	OWN	Actual time ACC - OWN
MAX	24	4,71	4,71	2,251
MIN	23	1,846	0,76	0
AVG	23	3,3498	2,6824	0,6674
DEV	1	2,864	3,95	2,251

Read	Test Iteraction	CNT	ACC	OWN	Actual time ACC - OWN
	1	13	3,53	0,14	3,39
	2	13	1,895	1,895	0
	3	14	1,866	1,866	0
	4	14	2,764	0,723	2,041
	5	14	1,854	1,854	0

	CNT	ACC	OWN	Actual time ACC - OWN
MAX	14	3,53	1,895	3,39
MIN	13	1,854	0,14	0
AVG	13	2,3818	1,2956	1,0862
DEV	1	1,676	1,755	3,39

Figure 6-23: Queries to Mnesia database

6.3.4 Test Coverage

In section 6.3.2 and 6.3.3 we have performed several tests to confirm the validity and performance of the implemented solution. Yet another aspect of testing is to check the test coverage, i.e. to check how much of the SUT is tested with the test cases. TheInternational Software Testing Qualifications Board (ISTQB) has defined three basic levels of test coverage criteria

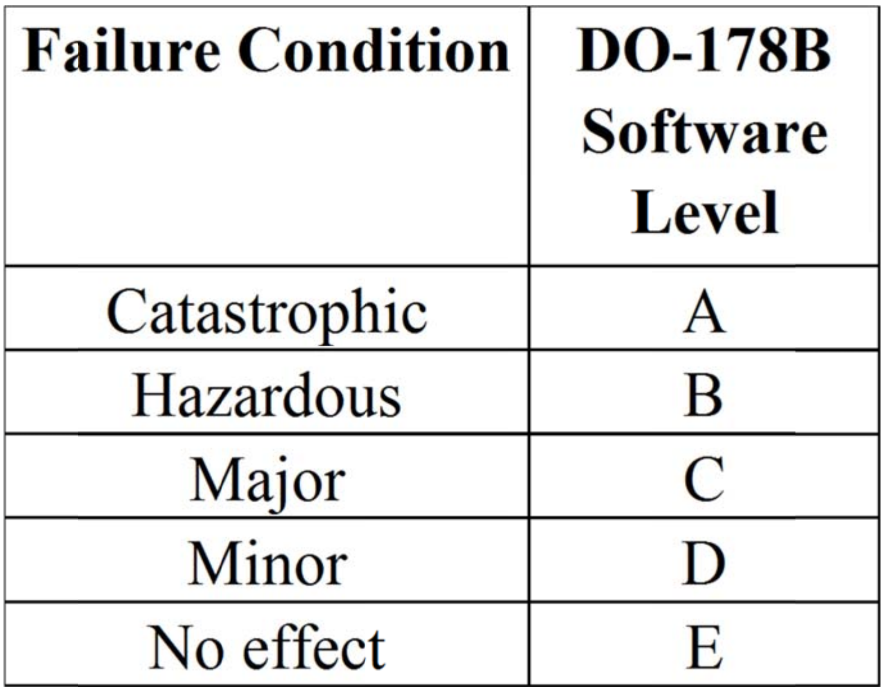
1. Statement coverage	To measure the percentage of statements being executed while performing tests
2. Decision coverage	To measure the percentage of decision outcomes (for example containing one or more conditions having entry and exit at-least once).
3. Condition coverage	To measure the percentage of each condition being evaluated to both true and false

It is recommended to use combination of above mentioned basic levels of test coverage criterion to perform rigorous test coverage. Two combinations of the basic levels of test coverage that are being extensively used are:

4. Decision-condition coverage	Both the decision and condition coverage should be satisfied
5. Modified Condition/Decision coverage (MC/DC)	MC/DC is achieved with Decision-condition coverage but with following three condition also fulfilled: a) At least one test; if the atomic condition is TRUE will change the decision outcome b) At least one test; if the atomic condition is FALSE will change the decision outcome c) All the atomic condition have a) and b) requirements

The author of this Master's thesis has completed the ISTQB foundation level course[47]. Additionally, he has prepared for the technical test analyst and test manager certifications[48]. This Master's thesis provided the basic domain knowledge needed for understanding of this section, however the interested reader is encouraged to see references[47-48].

The level of coverage differs for different software systems. We will provide an example to illustrate that test coverage for different software levels are defined separately. Airborne environments use the ED-12B[49] standard for its software test coverage. According to this standard the following five failure conditions are mapped to five software levels:



Failure Condition	DO-178B Software Level
Catastrophic	A
Hazardous	B
Major	C
Minor	D
No effect	E

Figure 6-24: Failure conditions and software levels of ED-12B, Adopted from[49]

According to the ED-12B standard the software level-A must undergo the MC/DC test coverage, level-B must undergo decision coverage; MC/DC test coverage is optional, level-C must undergo statement coverage and so-on for the rest of the levels.

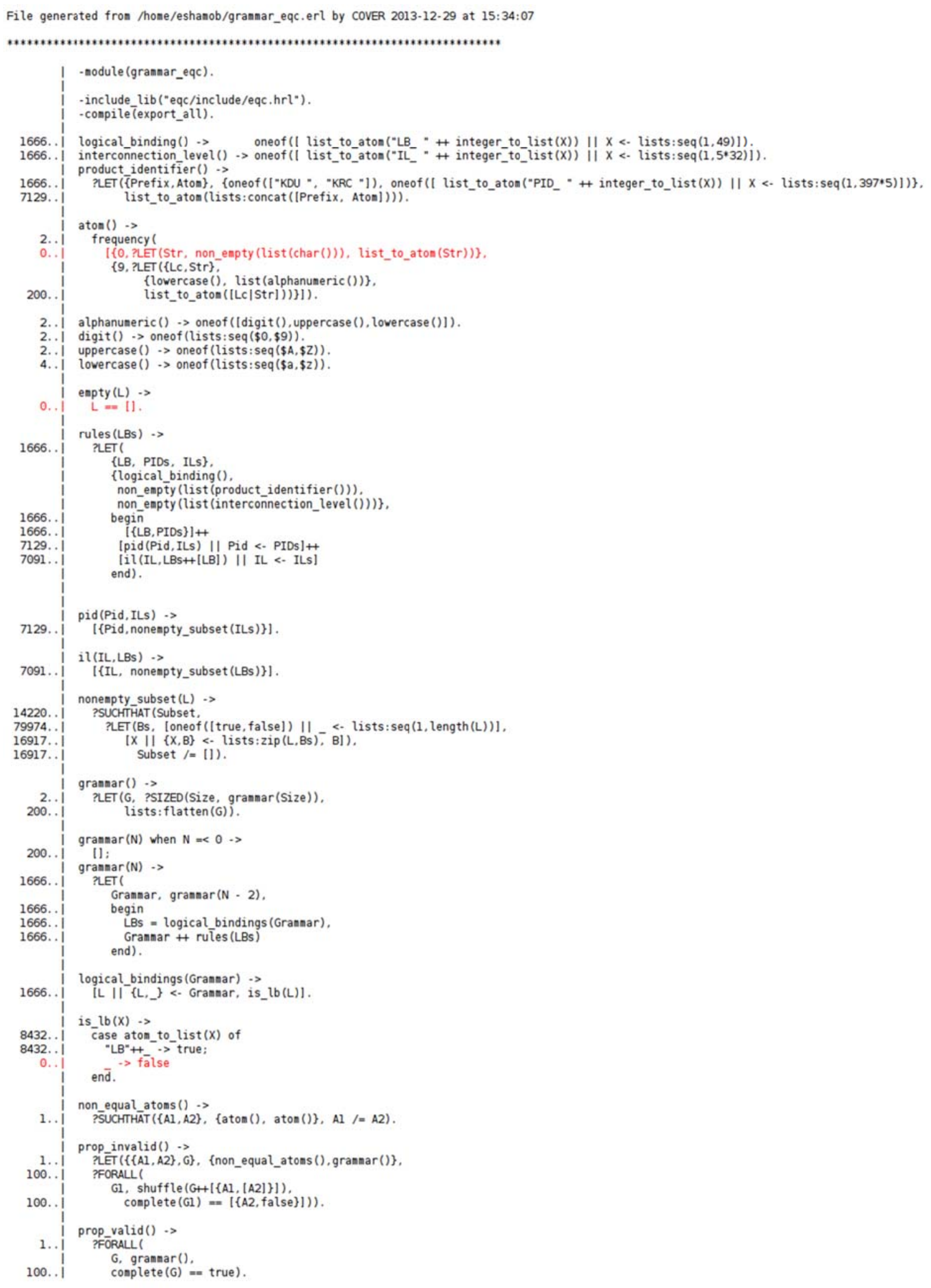
The above example shows that the significance of test coverage increases from 1-Statement coverage to 5-MC/DC coverage. Also different sub-systems of the software may require different level of coverage.

We have researched the available options for the test coverage for Erlang programs. We have found that Erlang has built-in tool *cover*[*] that provides statement, function, and module coverage. Another open source tool *smother*[†] is available for MC/DC coverage. These were the options available to us for test coverage. The test coverage results for the test module 'grammar_eqc' are presented in Figure 6-25 and Figure 6-26 for *Cover* and *Smother* respectively.

Analyzing the results from Figure 6-25 and Figure 6-26, we see that the tests are performed with 100% statement coverage and 100% condition coverage. As mentioned earlier *Smother* has the possibility to perform MC/DC coverage, but the results acquired from *Smother* do not add much to analysis in terms of test coverage as the module itself did not have complex conditions. Additionally, *Cover* and *Smother* indicated some unused code, i.e. code that was never used. The red lines with 0 count (in *cover,* see Figure 6-25) and red/orange square brackets (in *smother, see* Figure 6-26) are indications of unused code. The unused code has now been removed from the module.

[*] Cover, http://www.erlang.org/doc/man/cover.html, Last accessed : 2013-12-29

[†] Smother, http://ramsay-t.github.io/Smother/, Last accessed : 2013-12-29

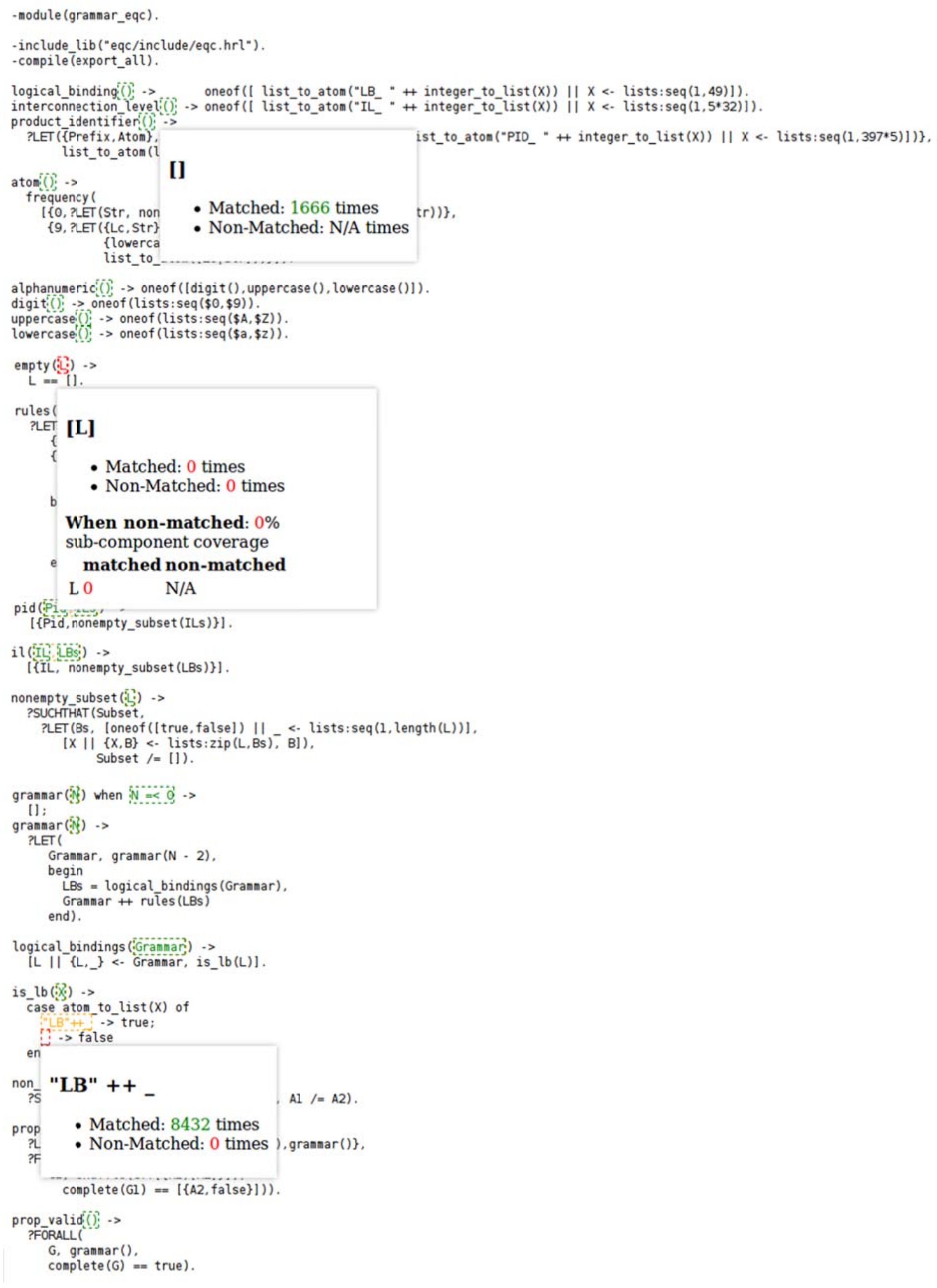


```
File generated from /home/eshamob/grammar_eqc.erl by COVER 2013-12-29 at 15:34:07

********************************************************************************

        |  -module(grammar_eqc).
        |
        |  -include_lib("eqc/include/eqc.hrl").
        |  -compile(export_all).
        |
 1666..|  logical_binding() ->       oneof([ list_to_atom("LB_ " ++ integer_to_list(X)) || X <- lists:seq(1,49)]).
 1666..|  interconnection_level() -> oneof([ list_to_atom("IL_ " ++ integer_to_list(X)) || X <- lists:seq(1,5*32)]).
        |  product_identifier() ->
 1666..|    ?LET({Prefix,Atom}, {oneof(["KDU ", "KRC "]), oneof([ list_to_atom("PID_ " ++ integer_to_list(X)) || X <- lists:seq(1,397*5)])},
 7129..|         list_to_atom(lists:concat([Prefix, Atom]))).
        |
        |  atom() ->
    2..|    frequency(
    0..|      [{0,?LET(Str, non_empty(list(char())), list_to_atom(Str))},
        |       {9,?LET({Lc,Str},
        |            {lowercase(), list(alphanumeric())},
  200..|            list_to_atom([Lc|Str]))}]).
        |
    2..|  alphanumeric() -> oneof([digit(),uppercase(),lowercase()]).
    2..|  digit() -> oneof(lists:seq($0,$9)).
    2..|  uppercase() -> oneof(lists:seq($A,$Z)).
    4..|  lowercase() -> oneof(lists:seq($a,$z)).
        |
        |  empty(L) ->
    0..|    L == [].
        |
        |  rules(LBs) ->
 1666..|    ?LET(
        |       {LB, PIDs, ILs},
        |       {logical_binding(),
        |        non_empty(list(product_identifier())),
        |        non_empty(list(interconnection_level()))},
 1666..|       begin
 1666..|         [{LB,PIDs}]++
 7129..|        [pid(Pid,ILs) || Pid <- PIDs]++
 7091..|        [il(IL,LBs++[LB]) || IL <- ILs]
        |       end).
        |
        |
        |  pid(Pid,ILs) ->
 7129..|    [{Pid,nonempty_subset(ILs)}].
        |
        |  il(IL,LBs) ->
 7091..|    [{IL, nonempty_subset(LBs)}].
        |
        |  nonempty_subset(L) ->
14220..|    ?SUCHTHAT(Subset,
79974..|      ?LET(Bs, [oneof([true,false]) || _ <- lists:seq(1,length(L))],
16917..|         [X || {X,B} <- lists:zip(L,Bs), B]),
16917..|           Subset /= []).
        |
        |  grammar() ->
    2..|    ?LET(G, ?SIZED(Size, grammar(Size)),
  200..|         lists:flatten(G)).
        |
        |  grammar(N) when N =< 0 ->
  200..|    [];
        |  grammar(N) ->
 1666..|    ?LET(
        |       Grammar, grammar(N - 2),
 1666..|       begin
 1666..|         LBs = logical_bindings(Grammar),
 1666..|         Grammar ++ rules(LBs)
        |       end).
        |
        |  logical_bindings(Grammar) ->
 1666..|    [L || {L,_} <- Grammar, is_lb(L)].
        |
        |  is_lb(X) ->
 8432..|    case atom_to_list(X) of
 8432..|      "LB"++_ -> true;
    0..|      _ -> false
        |    end.
        |
        |  non_equal_atoms() ->
    1..|    ?SUCHTHAT({A1,A2}, {atom(), atom()}, A1 /= A2).
        |
        |  prop_invalid() ->
    1..|    ?LET({{A1,A2},G}, {non_equal_atoms(),grammar()},
  100..|    ?FORALL(
        |       G1, shuffle(G++[{A1,[A2]}]),
  100..|         complete(G1) == [{A2,false}])).
        |
        |  prop_valid() ->
    1..|    ?FORALL(
        |       G, grammar(),
  100..|       complete(G) == true).
```

Figure 6-25: 'cover' Statement coverage



```
-module(grammar_eqc).

-include_lib("eqc/include/eqc.hrl").
-compile(export_all).

logical_binding() ->       oneof([ list_to_atom("LB_ " ++ integer_to_list(X)) || X <- lists:seq(1,49)]).
interconnection_level() -> oneof([ list_to_atom("IL_ " ++ integer_to_list(X)) || X <- lists:seq(1,5*32)]).
product_identifier() ->
  ?LET({Prefix,Atom},                                 ist_to_atom("PID_ " ++ integer_to_list(X)) || X <- lists:seq(1,397*5)])},
       list_to_atom(l

atom() ->
  frequency(
    [{0,?LET(Str, non                                 tr))},
     {9,?LET({Lc,Str}
             {lowerca
             list_to_

alphanumeric() -> oneof([digit(),uppercase(),lowercase()]).
digit() -> oneof(lists:seq($0,$9)).
uppercase() -> oneof(lists:seq($A,$Z)).
lowercase() -> oneof(lists:seq($a,$z)).

empty(L) ->
  L == [].

rules(
  ?LET
    {
    {

    b

    e

pid(Pi
  [{Pid,nonempty_subset(ILs)}].

il(IL,LBs) ->
  [{IL, nonempty_subset(LBs)}].

nonempty_subset(L) ->
  ?SUCHTHAT(Subset,
    ?LET(Bs, [oneof([true,false]) || _ <- lists:seq(1,length(L))],
       [X || {X,B} <- lists:zip(L,Bs), B]),
          Subset /= []).

grammar(N) when N =< 0 ->
  [];
grammar(N) ->
  ?LET(
     Grammar, grammar(N - 2),
     begin
       LBs = logical_bindings(Grammar),
       Grammar ++ rules(LBs)
     end).

logical_bindings(Grammar) ->
  [L || {L,_} <- Grammar, is_lb(L)].

is_lb(X) ->
  case atom_to_list(X) of
    "LB"++_ -> true;
    _ -> false
  en

non_
  ?S                                 , A1 /= A2).

prop
  ?L                                 ),grammar()},
  ?F
       complete(G1) == [{A2,false}])).

prop_valid() ->
  ?FORALL(
     G, grammar(),
     complete(G) == true).
```

[]

- Matched: 1666 times
- Non-Matched: N/A times

[L]

- Matched: 0 times
- Non-Matched: 0 times

When non-matched: 0%
sub-component coverage

```
  matched non-matched
L 0       N/A
```

"LB" ++ _

- Matched: 8432 times
- Non-Matched: 0 times

Figure 6-26: 'smother' MC/DC coverage

6.3.5 Testing of Web-UI

In section 6.3.2 and section 6.3.3 the core application logic was tested both from reliability and performance perspective. Also stated in section 5.2.2.1 tests initially used the CLI, rather than the Web-UI; the reason was that we want to separate the crashes of the core logic from the beard template solution (if any). As section 6.3.2 and section 6.3.3 provided satisfactory test results for the core logic, we will test the Web-UI.

The Web-UI was tested with the Selenium WebDriver [50] and WebClient in a Java application. Selenium is used to automatically navigate for transactions (See section 5.2.2.2) made on configurations and WebClient[51] is used to validate that the URL exists. We have performed several tests to test the interfaces (create a configuration and modify a configuration) described in section 5.2.2.2. The rest of the WEB-UI required some manual actions from the CLI so these part of the user interface are not part of this test, but we manually tested these interfaces and we experienced no problems whatsoever. The source code for testing of WEB-UI can be found in Appendix-D.

We will not perform load testing because we feel satisfied with the performance* of yaws.

*performance, http://www.sics.se/~joe/apachevsyaws.html , Last accessed : 2013-12-15

7 Conclusions and Future work

In this chapter, the core of work done in this thesis is concluded in section 7.1. General development experience during this thesis is also discussed in the same section. Section 7.2 highlights areas for future work. Finally, section 7.3 provides some reflections relevant to domain of this Master's thesis project.

7.1 Conclusions

This Master's thesis was carried out at Ericsson AB. A topological RMS system is the major contribution of this Master's thesis project. The use of an RMS has a clear advantage in terms of supporting the efficient utilization of resources and offers insights into the used and unused resources of the LAB. However, the RMS also provides input to the test environments. A topological RMS plays an important role by providing information about the utilization of resources and increases the visibility of the interconnections between these resources across the organization. In this way an alignment of TECs across different testing levels can be done, therefore interpretation of different TECs in different testing levels is not required, hence reducing the time it takes for a tester to be productive when switching between different testing levels. This can also potentially decrease the time required to setup a test environment by facilitating the reuse of a given test environment for testing on different levels by different testers.

The goal of implementing a topological RMS system that delivers as an output the TEC for a test environment has been successfully completed. The major challenges that we encountered were the selection of a path towards the implementation. Initially we encountered a major problem when we selected the wrong data structure for the topology information (see Section 5.2.2.3). Erlang was previously used as the development language for PeppesBodega-RMS and the implementation done as part of this thesis project, i.e. PeppesBodega-TF, was also done in this language. Erlang was new to the author of this Master's thesis and detailed features of this language were unknown. The implementation was modified from time to time as my knowledge of Erlang developed over time. The author of this Master's thesis was also new to web application development. Once again I would like to thank my supervisor, Magnus Kronqvist, for helping me develop my technical skills.

During the implementation of this Master's thesis, the front-controller of PeppesBodega-RMS was identified as a hindrance to the completion of the web UI for the management of PeppesBodega-TF. Although the author of this Master's thesis worked on modifications to the front-controller for quite some time, a decision was made in conjunction with my supervisor not to continue this development because it was taking too much time. This resulted in only partially fulfilling the goal of a providing web UI for management of PeppesBodega-TF. In retrospect the time spent on the development of front-controller could have better been used for development of a tool for visualization of the topology. As a result these two parts are left as future work for the further development of PeppesBodega-TF.

This Master's thesis contributes to the general development of a topological resource management system. The author identified several choices that should not be adopted in the development of a topological resource management system, such as the usage of a tree data structure for encoding configuration data of the topology and the maintainability problem of conventional methods (XML and Erlang-config) to define the grammar of the configuration data of topologies. Furthermore, in this thesis, two classes of devices (self-forming and lazy: see sections 2.3.1 and 2.3.2) specific to discovery of the network topologies were introduced.

Logical and physical topologies are two major geometric views of network topology. The developers of a topological RMS should include support for both geometric views, even if both views seem the same (as this need not be the case in the future).

As PeppesBodega-TF is extension to the PeppesBodega-RMS, the choices of tools were derived from PeppesBodega-RMS. We did not have a discuss the advantages and disadvantages of these choices because it would lead to a catch-22[*] situation.

7.2 Future work

This thesis was the first attempt towards integration of a topological RMS in PeppesBodega-RMS. Obviously we cannot claim this first attempt is the optimal or ultimate choice. Nor do we claim to have considered all the perspectives. Additionally, some of the goals were only partially fulfilled due to the limited duration of this Master's thesis project. For all of these reasons there are areas to be explored where the existing design could be modified and enhancements can be developed.

An essential future step is performing the delivery testing so that PeppesBodega-TF can be released to the target groups. This testing requirement was described in Section 6.3.

The goal of providing a web UI for administration of configuration data of the topology was not completed due to lack of functionality in front-controller in conjunction with the beard library. The front-controller needs to be able to handle requests in more efficient way. Currently the data for the web UI is transferred in a URL and in a topological RMS a large data request is sent making this approach infeasible. As described in Section 5.2.2.2 the implementation of the front-controller should be changed to efficiently send the data between webpages. There are quite a number of different methods that could be used to do this, including sending data in form-post/get methods, saving and retrieving session data, using cookies, etc. We will not discuss in detail which method should be adopted and the reasons for such a choice, because this requires a detailed investigation which was not possible within the scope of this Master's thesis project.

Visualization in general and particularly in the scope of a topological RMS serves an important role for debugging purposes by expert users and reducing the learning curve of novice users. PeppesBodega-TF lacks a visualization scheme, so it would be of great benefit to implement such a visualization scheme. Although a visualization scheme was included in the goals of this Master's thesis project, there was not sufficient time to implement any visualization scheme. However, we found several Erlang libraries for graphical visualization, including gtknode[†], wxErlang[‡], and Erlang-graphviz[§], that could be candidates for this visualization scheme. As we encode the configuration data of a topology as undirected graphs, the best suited library would seem to be Erlang-graphviz.

[*] Catch-22, http://en.wikipedia.org/wiki/Catch-22_(logic) , Last accessed : 2013-12-15

[†] gtknode, https://code.google.com/p/gtknode/, Last accessed : 2013-12-15

[‡] wxErlang , http://www.erlang.org/doc/man/wx.html , Last accessed : 2013-12-15

[§] Erlang-graphviz, https://github.com/glejeune/erlang-graphviz, Last accessed : 2013-12-15

An interesting set of issues that should be addressed in future work is the migration of the devices to IPv6 and the use of names to access devices, rather than fixed addresses. This would address one of the limitations described in Section 4.2.5.4.

Another future extension of the system would entail the assignment of unique DIDs to **all** devices that are maintained by the LAB. This would require the addition of these devices to the relevant STPs in the RMS. For further discussion of devices without unique DIDs see Section 6.1.2.

A security aspect should also be considered when the users can create the configurations of other signum other than their own. One way of tracking what is being created of one's signum is to initiate an email to corresponding user. Also secure login mechanism should be integrated via Ericsson corporate ID authentication and then one can only create configuration for its own signum.

The choices of these programming paradigms were inherited from the design and implementation of the existing PeppesBodega-RMS. While these choices were straight forward, it was also important to understand the reasons why these programming paradigms had been chosen. The selection of Erlang was due to the fact that its performance is far superior to Java[*] (one of the most widely used programming language today) [52]. The selection of Yaws is also attributed to its better performance[†] as compared to Apache[46].

An important lesson learned is that knowledge of the development tools is important when devising a solution for a problem. During the implementation, the author frequently implemented certain functions which could have been done in an easier way if more time had been spent to acquire deeper knowledge of development tools. This was especially true for Erlang, Yaws, and computer grammars. For example, we implemented the grammar using our own format, but later we found out that the grammar could have been defined in yecc[‡] with much less effort. However, the learning curve of yecc is relatively high, so the programmer has to make a conscious decision to learn how to use the proper tool rather than using the *ad hoc* approach used in this Master's thesis.

7.3 Reflections

This Master's thesis was encouraged and motivated by MSV department at Ericsson AB. During the course of this Master's thesis project the author worked closely with the verification engineers and software developers from different departments to gather the needs for the topological resource management system that offers a scalable solution for all the departments involved.

The **social** contribution of this Master's thesis is the topological resource management system which will be used within Ericsson AB after its successful deployment at MSV department. Another **social** aspect of this Master's thesis is that the grammar translation will

[*] Java, http://www.java.com/en/ , Last accessed : 2013-12-15

[†] performance, http://www.sics.se/~joe/apachevsyaws.html , Last accessed : 2013-12-15

[‡] yecc, http://www.erlang.org/doc/man/yecc.html , Last accessed : 2013-12-15

be made publically available in GitHub[*] for projects with limited time, as yecc has a greater learning curve. The implemented solution (and proposed future work) in its complete form will make an **economic** contribution as the execution time required for testing will be reduced and also the learning curve reduction for fresh engineer - both of which will have an economic benefit to Ericsson AB**.** The social and economic contributions stated above will be benefitting Ericsson AB by this Master's thesis and will contribute to increasing competitiveness in Swedish industry.

The **ethical** aspects were also considered pertaining to not a) disclose any kind of confidential information of the Ericsson's LAB equipment b) manage any data of a personal nature whatsoever, and a generic model of topological resource management system is now available to the research community via this thesis report. The information collected by implemented topological resource management system contains only physical topologies which may, when in place and executing in the final live environment, contain confidential data (that will be managed by Ericsson's confidential policy); but will not manage any data of a personal nature.

[*] GitHub, "Powerful collaboration, code review, and code management for open source and private projects", https://www.github.com/, Last accessed : 2013-12-29

References

[1] B. B. Agarwal, M. Gupta, and S. P. Tayal, *Software engineering & testing: an introduction.* Sudbury, Mass.: Jones and Bartlett Publishers, 2010, ISBN: 978-0763782993.

[2] G. M. Parker, *Cross-functional teams working with allies, enemies, and other strangers*. San Francisco, Calif.: Jossey-Bass, 2003, ISBN: 978-0787965600.

[3] B. Beizer, *Software system testing and quality assurance*. New York, NY, USA: Van Nostrand Reinhold Co., 1984, ISBN: 0-442-21306-9.

[4] S. Naik, *Software Testing and Quality Assurance*. John Wiley & Sons, 2007, ISBN: 0471789119.

[5] R. D. Craig and S. P. Jaskiel, *Systematic Software Testing*. Artech House, 2002, ISBN: 9781580537926.

[6] M. Mazurkiewicz, 'Gui test automation with swtbot', Vaasan Ammattikorkeakoulu University of Applied Sciences, 2010.

[7] A. Gutierrez Lopez, M. Viela, and I. Manuel, *Automated Telecommunication Software Testing : An automated model generator for Model-Based Testing*. Masters's thesis, KTH Royal Institute of Technology, School of Information and Communications Systems,Communication Systems, Stockholm, Sweden: , 2012, Available at http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-93852.

[8] 'Examples of Test Oracles'. [Online]. Available: http://www.testingeducation.org/k04/OracleExamples.htm. [Accessed: 14-August-2013].

[9] 'Comparison of project management software', *Wikipedia, the free encyclopedia*, 15-August-2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Comparison_of_project_management_software&oldid=568593005. [Accessed: 15-August-2013].

[10] K. Horikiri and S. Kawabe, 'Resource management system', U.S. Patent 576515409-June-1998Available at http://www.google.com/patents?id=DSEhAAAAEBAJ.

[11] J. C. Wu, 'Automatic discovery of network elements', U.S. Patent 518586009-February-1993Available at http://www.google.com/patents?id=4ZcXAAAAEBAJ.

[12] A. Sharon, R. Levy, Y. Cohen, A. Haiut, A. Stroh, and D. Raz, 'Automatic network topology analysis', U.S. Patent 620512220-March-2001Available at http://www.google.com/patents?id=YisGAAAAEBAJ.

[13] D. Chatwani, R. Subramanian, W. Chiang, J. Davar, A. Opher, and S. Sawant, 'Method for providing for automatic topology discovery in an ATM network or ...', U.S. Patent 566410702-September-1997Available at http://www.google.com/patents?id=EEsiAAAAEBAJ.

[14] N. Migas, W. J. Buchanan, and K. A. McArtney, 'Mobile agents for routing, topology discovery, and automatic network reconfiguration in ad-hoc networks', in *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the*, 2003, pp. 200–206, DOI:10.1109/ECBS.2003.1194800.

[15] D. Knertser and V. Tsarinenko, *Network Device Discovery*, Master's thesis. KTH Royal Institute of Technology, School of Information and Communication Technology: TRITA-ICT-EX-2013:90, June 2013, Available at http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-123509.

[16] 'Comparison of network diagram software', *Wikipedia, the free encyclopedia*, 14-August-2013. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Comparison_of_network_diagram_software&oldid=568516332. [Accessed: 03-September-2013].

[17] 'Computer Networks Demystified | Network Topology'. [Online]. Available: http://networking.layer-x.com/p020000-1.html. [Accessed: 05-September-2013].

[18] 'Logical vs. physical topology'. [Online]. Available: http://thought1.org/nt100/module3/logical_vs.html. [Accessed: 05-September-2013].

[19] C. Schurgers, V. Tsiatsis, S. Ganeriwal, and M. Srivastava, 'Topology management for sensor networks: exploiting latency and density', in *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking &amp; computing*, New York, NY, USA, 2002, pp. 135–145, DOI:10.1145/513800.513817, Available at http://doi.acm.org/10.1145/513800.513817.

[20] J. Pan, L. Cai, Y. T. Hou, Y. Shi, and S. X. Shen, 'Optimal base-station locations in two-tiered wireless sensor networks', *IEEE Transactions on Mobile Computing*, vol. 4, no. 5, pp. 458–473, 2005, DOI:10.1109/TMC.2005.68.

[21] W. Hsu and A. S. Kulkarni, 'Network topology management system through a database of managed network resources including logical topolgies', U.S. Patent US5848243 A08-December-1998.

[22] N. Chomsky, 'Three models for the description of language', *IRE Transactions on Information Theory*, vol. 2, no. 3, pp. 113–124, 1956, DOI:10.1109/TIT.1956.1056813.

[23] A. M. Natarajan, *Theory of Automata & Formal Languages: As Per UPTU Syllabus*. New Age International, 2005, ISBN: 9788122417296.

[24] 'LL parser'. [Online]. Available: http://www.princeton.edu/~achaney/tmve/wiki100k/docs/LL_parser.html. [Accessed: 04-October-2013].

[25] A. R. Hevner and S. Chatterjee, *Design research in information systems theory and practice*. New York; London: Springer, 2010, ISBN: 1441956530 9781441956538, Available at http://dx.doi.org/10.1007/978-1-4419-5653-8.

[26] 'DESRIST: Design Science Research in Information Systems Overview'. [Online]. Available: http://www.desrist.org/desrist/. [Accessed: 31-October-2013].

[27] A. R. Hevner, S. T. March, J. Park, and S. Ram, 'Design science in information systems research', *MIS Q.*, vol. 28, no. 1, pp. 75–105, March 2004.

[28] 'XML Technology - W3C'. [Online]. Available: http://www.w3.org/standards/xml/. [Accessed: 12-August-2013].

[29] 'Portable Network Graphics'. [Online]. Available: http://www.w3.org/Graphics/PNG/. [Accessed: 12-August-2013].

[30] 'Strangeloop - Acceptable website response times - Web Performance Optimization'. [Online]. Available: http://www.strangeloopnetworks.com/resources/infographics/why-

luxury-websites-are-disappointing-chinese-consumers/acceptable-website-response-times/. [Accessed: 27-August-2013].

[31] ‘Erlang Programming Language’. [Online]. Available: http://www.erlang.org/. [Accessed: 10-October-2013].

[32] ‘beard’, *GitHub*. [Online]. Available: https://github.com/danabr/beard. [Accessed: 10-October-2013].

[33] ‘mustache.erl’, *GitHub*. [Online]. Available: https://github.com/mojombo/mustache.erl. [Accessed: 17-October-2013].

[34] ‘HTML Tutorial’. [Online]. Available: http://www.w3schools.com/html/. [Accessed: 17-October-2013].

[35] ‘QuviQ homepage’. [Online]. Available: http://www.quviq.com/index.html. [Accessed: 18-October-2013].

[36] ‘Erlang -- mnesia’. [Online]. Available: http://www.erlang.org/doc/man/mnesia.html. [Accessed: 21-October-2013].

[37] ‘About BNF notation’. [Online]. Available: http://cui.unige.ch/db-research/Enseignement/analyseinfo/AboutBNF.html. [Accessed: 25-November-2013].

[38] ‘BNF Notation for syntax’. [Online]. Available: http://www.w3.org/Notation.html. [Accessed: 25-November-2013].

[39] ‘Notations for context-free grammars: BNF, Syntax Diagrams, EBNF’. [Online]. Available: http://www.cs.man.ac.uk/~pjj/bnf/bnf.html. [Accessed: 25-November-2013].

[40] A. V. Aho, J. E. Hopcroft, and Ullman, *Data structures and algorithms*. Reading, Mass.: Addison-Wesley, 1983, ISBN: 0201000237 9780201000238.

[41] D. A. Turner, *Research topics in functional programming*. Addison-Wesley Pub. Co., 1990, ISBN: 9780201172362.

[42] M. Jovic and M. Hauswirth, ‘Measuring the performance of interactive applications with listener latency profiling’, 2008, p. 137, DOI:10.1145/1411732.1411751, Available at http://sape.inf.usi.ch/publications/pppj08.

[43] B. Shneiderman and C. Plaisant, *Designing the user interface: strategies for effective human-computer interaction*. Boston: Addison-Wesley, 2010, ISBN: 9780321537355 0321537351 9780321601483 0321601483.

[44] B. Shneiderman, ‘Response Time and Display Rate in Human Performance with Computers’, *ACM Comput. Surv.*, vol. 16, no. 3, pp. 265–285, September 1984, DOI:10.1145/2514.2517.

[45] M. Jovic and M. Hauswirth, ‘Performance Testing of GUI Applications’, 2010, pp. 247–251, DOI:10.1109/ICSTW.2010.27, Available at http://sape.inf.usi.ch/publications/testbeds10.

[46] ‘Apache vs. Yaws’. [Online]. Available: http://www.sics.se/~joe/apachevsyaws.html. [Accessed: 15-December-2013].

[47] ‘Foundation Level Syllabus - ISTQB® International Software Testing Qualifications Board’. [Online]. Available: http://www.istqb.org/downloads/syllabi/foundation-level-syllabus.html. [Accessed: 29-December-2013].

[48] ‘Advanced Level Syllabus - ISTQB® International Software Testing Qualifications Board’. [Online]. Available: http://www.istqb.org/downloads/syllabi/advanced-level-syllabus.html. [Accessed: 29-December-2013].

[49] T. K.Ferrell and U. D.Ferrel, ‘RTCA DO-178B/EUROCAE ED-12B’. [Online]. Available: http://www.davi.ws/avionics/TheAvionicsHandbook_Cap_27.pdf. [Accessed: 29-December-2013].

[50] ‘Selenium WebDriver’. [Online]. Available: http://docs.seleniumhq.org/projects/webdriver/. [Accessed: 30-December-2013].

[51] ‘WebClient (HtmlUnit 2.13 API)’. [Online]. Available: http://htmlunit.sourceforge.net/apidocs/com/gargoylesoftware/htmlunit/WebClient.html. [Accessed: 30-December-2013].

[52] ‘Performance Measurements of Threads in Java and Processes in Erlang’. [Online]. Available: http://www.sics.se/~joe/ericsson/du98024.html. [Accessed: 15-December-2013].

Appendices

A. BNF-Standard grammar

```
%% DU                  ::= KDU 127 161/1 R1A/4 | KDU 137 930/1 P1B
%% Radio               ::= KRC 118 75/1
%% KDU 127 161/1 R1A/4  ::=  CpriPort CpriPort CpriPort CpriPort IDLPort GPSOption
%% KDU 137 930/1 P1B    ::=  CpriPort CpriPort CpriPort CpriPort IDLPort GPSOption
%% KRC 118 75/1        ::=  CpriPort CpriPort
%% CpriPort            ::=  undefined | DU | CPRIC | Radio
%% IDLPort             ::=  undefined | DU
%% GPSOption           ::=  undefined | on
%% PowerPort           ::=  undefined | IP
%% IP                  ::= Integer
%% CPRIC               ::= cpri-2.5 | cpri-5 | cpri-10
%% CPRI-2.5            ::= CpriPort CpriPort
%% CPRI-5              ::= CpriPort CpriPort
%% CPRI-10             ::= CpriPort CpriPort
```

B. BNF-Erlang grammar

```
 [{du,['KDU 127 161/1 R1A/4']},
 {du,['KDU 137 930/1 P1B']},
 {radio,['KRC 118 75/1']},
 {'KDU 127 161/1 R1A/4',[cpriport,cpriport,cpriport,cpriport,idlport,gpsoption]},
 {'KDU 137 930/1 P1B',[cpriport,cpriport,cpriport,cpriport,idlport,gpsoption]},
 {'KRC 118 75/1',[cpriport,cpriport]},
 {cpriport,[undefined]},
 {cpriport,[du]},
 {cpriport,[cpric]},
 {cpriport,[radio]},
 {idlport,[undefined]},
 {idlport,[du]},
 {gpsoption,[undefined]},
 {gpsoption,[on]},
 {powerport,[undefined]},
 {powerport,[ip]},
 {ip,[integer]},
 {cpric,['cpri-2.5']},
 {cpric,['cpri-5']},
 {cpric,['cpri-10']},
 {'cpri-2.5',[cpriport,cpriport]},
 {'cpri-5',[cpriport,cpriport]},
 {'cpri-10',[cpriport,cpriport]}].
```

C. QuickCheck Testing

```
-module(grammar_eqc).

-include_lib("eqc/include/eqc.hrl").
-compile(export_all).

logical_binding() ->
  oneof([ list_to_atom("LB_ " ++ integer_to_list(X)) || X <- lists:seq(1,10)]).
interconnection_level() ->
  oneof([ list_to_atom("ICL_ "++integer_to_list(X)) || X <- lists:seq(1,10)]).
product_identifier() ->
  ?LET({Prefix,Atom},
    {oneof(["KDU ", "KRC "]),
     oneof([ list_to_atom("PID_ " ++ integer_to_list(X)) || X <- lists:seq(1,10)])},
       list_to_atom(lists:concat([Prefix, Atom]))).

atom() ->
  frequency(
    [{0,?LET(Str, non_empty(list(char())), list_to_atom(Str))},
     {9,?LET({Lc,Str},
              {lowercase(), list(alphanumeric())},
              list_to_atom([Lc|Str]))}]).

alphanumeric() -> oneof([digit(),uppercase(),lowercase()]).
digit() -> oneof(lists:seq($0,$9)).
uppercase() -> oneof(lists:seq($A,$Z)).
lowercase() -> oneof(lists:seq($a,$z)).

empty(L) ->
  L == [].

rules(LBs) ->
  ?LET(
     {LB, PIDs, ILs},
     {logical_binding(),
      non_empty(list(product_identifier())),
      non_empty(list(interconnection_level()))},
     begin
       [{LB,PIDs}]++
           [pid(Pid,ILs) || Pid <- PIDs]++
           [il(IL,LBs++[LB]) || IL <- ILs]
     end).



pid(Pid,ILs) ->
  [{Pid,nonempty_subset(ILs)}].

il(IL,LBs) ->
  [{IL, nonempty_subset(LBs)}].

nonempty_subset(L) ->
  ?SUCHTHAT(Subset,
    ?LET(Bs, [oneof([true,false]) || _ <- lists:seq(1,length(L))],
       [X || {X,B} <- lists:zip(L,Bs), B]),
             Subset /= []).
```

```
grammar() ->
  ?LET(G, ?SIZED(Size, grammar(Size)),
       lists:flatten(G)).

grammar(N) when N =< 0 ->
  [];
grammar(N) ->
  ?LET(
     Grammar, grammar(N - 2),
     begin
       LBs = logical_bindings(Grammar),
       Grammar ++ rules(LBs)
     end).

logical_bindings(Grammar) ->
  [L || {L,_} <- Grammar, is_lb(L)].

is_lb(X) ->
  case atom_to_list(X) of
    "lb"++_ -> true;
    _ -> false
  end.

non_equal_atoms() ->
  ?SUCHTHAT({A1,A2}, {atom(), atom()}, A1 /= A2).

prop_invalid() ->
  ?LET({{A1,A2},G}, {non_equal_atoms(),grammar()},
  ?FORALL(
     G1, shuffle(G++[{A1,[A2]}]),
       complete(G1) /= true)).

prop_valid() ->
  ?FORALL(
     G, grammar(),
     complete(G) == true).
```

D.   Testing of Web-UI

```java
import java.io.IOException;
import java.net.MalformedURLException;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.htmlunit.HtmlUnitDriver;

import com.gargoylesoftware.htmlunit.FailingHttpStatusCodeException;
import com.gargoylesoftware.htmlunit.WebClient;


public class App {
    public static void Validate_status(String URL)
            throws FailingHttpStatusCodeException, MalformedURLException,
            IOException {
        WebClient webClient = new WebClient();
        webClient.getPage(URL).getWebResponse().getStatusCode();
        webClient.closeAllWindows();
    }

    public static void main(String[] args)
            throws FailingHttpStatusCodeException, MalformedURLException,
            IOException {

        for (int i : new int[] { 1, 2, 3, 4, 5 }) {
            WebDriver driver = new HtmlUnitDriver();
            String Create_configuration_url = "http://localhost:9090/topology";
            Validate_status(Create_configuration_url);
            driver.get(Create_configuration_url);

            WebElement name = driver.findElement(By.name("name"));
            WebElement signum = driver.findElement(By.name("signum"));
            char c = (char) (i + 97);
            name.sendKeys("conf" + c);
            signum.sendKeys("signum");
            name.submit();

            String Modify_configuration_url = driver.getCurrentUrl();
            for (int ii : new int[] { 1000, 1001, 1002, 1003, 1004 }) {
                driver.get(Modify_configuration_url);
                driver.findElement(By.name("adddid")).submit();
                WebElement id = driver.findElement(By.name("id"));
                id.sendKeys(Integer.toString(ii));
            }

            driver.close();
        }
    }
}
```

Figure 7-1:   Web-UI testing with Selenium and WebClient

TRITA-ICT-EX-2014:1