

# **Wedndah Asaha Asong**

Establishing a Communication Infrastructure to  
Optimize a Distributed Energy Control System

Master thesis based on the examination and study regulations for  
the Master of Engineering degree programme  
Information Engineering  
at the Department of Information and Electrical Engineering  
of the Faculty of Engineering and Computer Science  
of the University of Applied Sciences Hamburg

Supervising examiner : Prof. Dr. Wolfgang Fohl  
Second examiner : Prof. Dr. Ulrich Sauvagerd

Day of delivery December 10<sup>th</sup> 2009

**Wedndah Asaha Asong**

**Title of the Master Thesis**

Establishing a Communication Infrastructure to Optimize a Distributed Energy Control System

**Keywords**

Energy grid, energy controller, communal heating system, communication protocol, solar thermal plant

**Abstract**

This thesis describes the development of a communication infrastructure for a distributed energy control system. This system has multiple controllers which communicate with each other and with some web servers. The controllers can as well be accessed remotely via a centralized server.

End of text

**Wedndah Asaha Asong**

**Thema der Masterarbeit**

Entwicklung einer Kommunikationsinfrastruktur für die Optimierung eines verteilten Energiesteuerungssystems

**Stichworte**

Energienetz, Energiesteuerungsgerät, kommunale Heizung, Kommunikationsprotokol, solarthermische Anlage

**Kurzzusammenfassung**

Diese Arbeit umfasst die Entwicklung einer Kommunikationsinfrastruktur für ein verteiltes Energiesteuerungssystem. Dieses System hat mehrere Steuerungsgeräte, die miteinander und mit einigen Webservern kommunizieren. Die Geräte können auch über einen zentralen Server fernzugriffen werden.

Ende des Textes

## **Dedication**

I dedicate my thesis to my parents for their much love, care, support and encouragement. They sacrificed a lot to make me what I am today and the principles they brought me up with, guide me to this very day. Simply said; without them, this thesis would not be possible. I love you mum and dad and thank you so much for all you have done to my life.

# Contents

- 1 INTRODUCTION.....7**
  - 1.1 PROJECT OVERVIEW .....8
  - 1.2 THESIS OUTLINE .....9
  
- 2 REQUIREMENTS ANALYSIS.....10**
  - 2.1 THE ENERGY GRID .....10
  - 2.2 THE T400 ENERGY CONTROLLER.....11
    - 2.2.1 Use Case .....11
    - 2.2.2 Activity Diagram .....13
  - 2.3 COMMUNICATION REQUIREMENTS .....14
    - 2.3.1 T400 – T400 .....14
    - 2.3.2 T400 – District Heat Authority .....15
    - 2.3.3 T400 – Web DB .....15
    - 2.3.4 T400 – Remote Administrator .....15
  
- 3 SYSTEM DESIGN .....16**
  - 3.1 NETWORK ARCHITECTURE.....16
  - 3.2 DESIGN CONSTRAINTS.....17
    - 3.2.1 Solution 1: Direct Access .....17
    - 3.2.2 Solution 2: Access via a Centralized Server .....18
  
- 4 SYSTEM SPECIFICATION .....21**
  - 4.1 DATABASE MODEL.....22
    - 4.1.1 User Table.....22
    - 4.1.2 Group Table .....23
    - 4.1.3 UserGroup Table .....23
    - 4.1.4 Device Table.....23
    - 4.1.5 Users View .....24
    - 4.1.6 User\_roles Table .....24
  - 4.2 SERVICE SPECIFICATION .....24
    - 4.2.1 Service Signal .....25
    - 4.2.2 Init Service .....27
    - 4.2.3 List Service .....27
    - 4.2.4 Resource Service .....27
    - 4.2.5 Update Service .....28
    - 4.2.6 Error Service.....28
    - 4.2.7 Connect Service.....28
    - 4.2.8 Data Service.....29
    - 4.2.9 Disconnect Service .....29

4.3	PROTOCOL SPECIFICATION .....	29
4.3.1	<i>Static Protocol Specification</i> .....	30
4.3.2	<i>Dynamic Protocol Specification</i> .....	33
<b>5</b>	<b>SYSTEM IMPLEMENTATION .....</b>	<b>38</b>
5.1	DEVELOPMENT ENVIRONMENT .....	38
5.1.1	<i>NanoLIAB Board</i> .....	38
5.1.2	<i>Windows XP PC</i> .....	39
5.2	MYSQL DATABASE .....	39
5.2.1	<i>User Table</i> .....	40
5.2.2	<i>Groups Table</i> .....	40
5.2.3	<i>UserGroup Table</i> .....	41
5.2.4	<i>Device Table</i> .....	41
5.2.5	<i>Users View</i> .....	42
5.2.6	<i>User_roles Table</i> .....	42
5.3	CENTRALIZED SERVER .....	42
5.3.1	<i>Software Architecture</i> .....	43
5.3.2	<i>Package dk.sunsil.cs.util</i> .....	45
5.3.3	<i>Package dk.sunsil.cs.bean</i> .....	47
5.3.4	<i>Package dk.sunsil.cs.dao</i> .....	49
5.3.5	<i>Package dk.sunsil.cs.dao.impl</i> .....	51
5.3.6	<i>Package dk.sunsil.cs.signal</i> .....	53
5.3.7	<i>Package dk.sunsil.cs.appentity</i> .....	54
5.3.8	<i>Package dk.sunsil.cs.servlet</i> .....	56
5.3.9	<i>Package dk.sunsil.cs.servlet.form</i> .....	59
5.3.10	<i>Package dk.sunsil.cs.servlet.action</i> .....	60
5.3.11	<i>Summary</i> .....	61
<b>6</b>	<b>SYSTEM TESTING.....</b>	<b>62</b>
6.1	AUTHENTICATION .....	62
6.2	THE HOMEPAGE .....	62
6.3	THE DEVICE PAGE.....	64
6.4	THE USER PAGE .....	66
6.5	THE GROUPS PAGE .....	68
6.6	THE USERGROUP PAGE.....	70
6.7	THE CONNECT PAGE.....	71
<b>7</b>	<b>CONCLUSION .....</b>	<b>73</b>
	<b>REFERENCES.....</b>	<b>74</b>
	<b>ABBREVIATIONS.....</b>	<b>75</b>
	<b>APPENDIX.....</b>	<b>76</b>
	<b>ACKNOWLEDGEMENT .....</b>	<b>84</b>
	<b>DECLARATION.....</b>	<b>85</b>

# Figures

FIGURE 1-1: TYPICAL SETUP FOR A SOLAR THERMAL SYSTEM.....	7
FIGURE 1-2: A SIMPLE COMMUNAL SYSTEM .....	8
FIGURE 2-1: SIMPLE USE CASE OF ENERGY GRID .....	10
FIGURE 2-2: USE CASE OF T400 ENERGY CONTROLLER.....	12
FIGURE 2-3: ACTIVITY DIAGRAM OF T400 ENERGY CONTROLLER .....	13
FIGURE 3-1: NETWORK ARCHITECTURE FOR COMMUNAL HEATING SYSTEMS.....	16
FIGURE 3-2: DIRECT REMOTE ACCESS TO T400 DEVICE .....	17
FIGURE 3-3: REMOTE ACCESS TO T400 DEVICE VIA A CENTRALIZED SERVER .....	19
FIGURE 4-1: SYSTEM SPECIFICATION.....	21
FIGURE 4-2: DATABASE MODEL .....	22
FIGURE 4-3: SERVICE SPECIFICATION .....	25
FIGURE 4-4: STRUCTURE OF A SERVICE SIGNAL .....	26
FIGURE 4-5: SEQUENCE DIAGRAM .....	30
FIGURE 4-6: BLOCK SPECIFICATION OF TAPPENTITY .....	31
FIGURE 4-7: BLOCK SPECIFICATION OF CSAPPENTITY .....	32
FIGURE 4-8: STATE DIAGRAM OF TCOM PROCESS .....	34
FIGURE 4-9: STATE DIAGRAM OF CSCOM PROCESS .....	35
FIGURE 4-10: PROCEDURE CONNECT OF CSCOM .....	36
FIGURE 5-1: NANOLIAB BOARD .....	39
FIGURE 5-2: SOFTWARE ARCHITECTURE OF CENTRALIZED SERVER.....	43
FIGURE 5-3: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.UTIL.....	45
FIGURE 5-4: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.BEAN .....	47
FIGURE 5-5: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.DAO.....	49
FIGURE 5-6: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.DAO.IMPL .....	51
FIGURE 5-7: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.SIGNAL .....	53
FIGURE 5-8: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.APPENTITY.....	54
FIGURE 5-9: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.SERVLET.....	56
FIGURE 5-10: CONNECTSERVLET TEMPLATE.....	57
FIGURE 5-11: BASESERVLET TEMPLATE.....	58
FIGURE 5-12: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.SERVLET.FORM .....	59
FIGURE 5-13: CLASS DIAGRAM OF PACKAGE DK.SUNSIL.CS.SERVLET.ACTION .....	60
FIGURE 6-1: USER AUTHENTICATION PROMPT.....	62
FIGURE 6-2: THE HOMEPAGE .....	63
FIGURE 6-3: EDIT PROFILE FORM .....	63
FIGURE 6-4: THE DEVICE PAGE .....	64
FIGURE 6-5: ADD DEVICE FORM.....	65
FIGURE 6-6: EDIT DEVICE FORM.....	65
FIGURE 6-7: DELETE DEVICE PROMPT.....	66
FIGURE 6-8: THE USER PAGE.....	66
FIGURE 6-9: ADD USER FORM .....	67
FIGURE 6-10: EDIT USER FORM .....	67

FIGURE 6-11: DELETE USER PROMPT .....	68
FIGURE 6-12: THE GROUPS PAGE.....	68
FIGURE 6-13: ADD GROUP FORM .....	69
FIGURE 6-14: EDIT GROUP FORM .....	69
FIGURE 6-15: DELETE GROUP PROMPT.....	69
FIGURE 6-16: THE USERGROUP PAGE .....	70
FIGURE 6-17: ADD USER GROUP FORM.....	70
FIGURE 6-18: DELETE USER GROUP PROMPT.....	71
FIGURE 6-19: THE CONNECT PAGE – DEVICE PROPERTIES .....	71
FIGURE 6-20: THE CONNECT PAGE - PARAMETERS .....	72
FIGURE 6-21: THE CONNECT PAGE - EDIT FORM.....	72

## Tables

TABLE 4-1: SUMMARY OF SERVICE SIGNALS .....	26
TABLE 4-2: HTTP ENCODING OF APDU'S FOR SERVICE REQUESTS .....	32
TABLE 4-3: HTTP ENCODING OF APDU'S FOR SERVICE RESPONSES.....	33
TABLE 5-1: SOFTWARE COMPONENTS OF CENTRALIZED SERVER .....	44
TABLE 5-2: LANGUAGE CODES AND NAMES .....	46

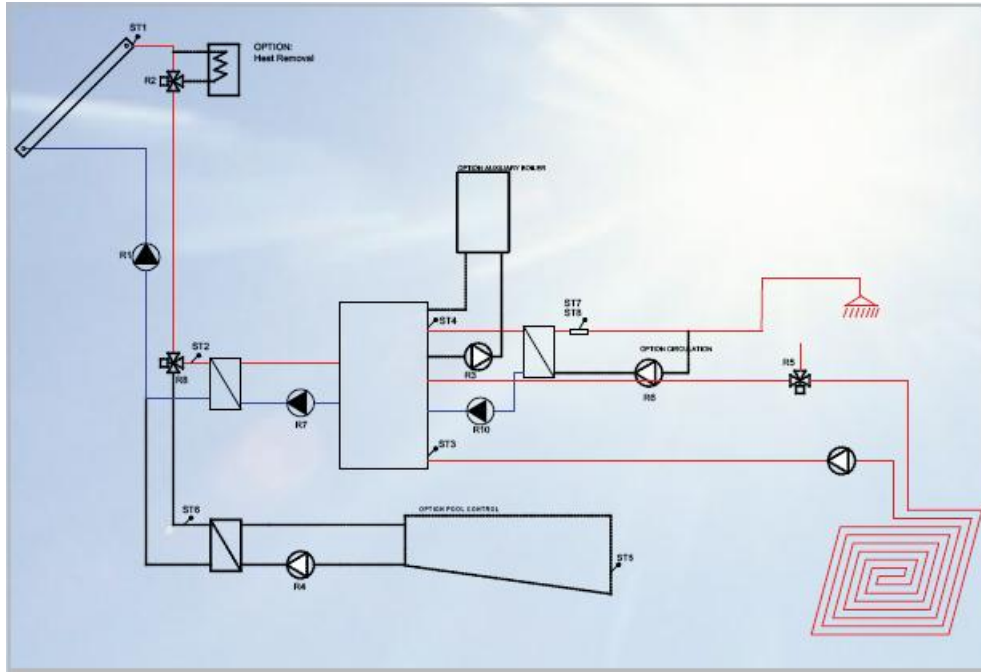


## Listings

LISTING 5-1: DEFINITION OF USER TABLE.....	40
LISTING 5-2: DEFINITION OF GROUPS TABLE.....	40
LISTING 5-3: DEFINITION OF USERGROUP TABLE .....	41
LISTING 5-4: DEFINITION OF DEVICE TABLE .....	41
LISTING 5-5: DEFINITION OF USERS VIEW.....	42
LISTING 5-6: DEFINITION OF USER_ROLES TABLE .....	42
LISTING 5-7: LOOKUP OF ENVIRONMENT CONTEXT .....	45
LISTING 5-8: FILE LANG.PROPERTIES .....	46
LISTING 5-9: CREATING A CONNECTION FROM A DATA SOURCE .....	52
LISTING 5-10: LOADING DAO IMPLEMENTATIONS.....	52

# 1 Introduction

There is currently a high demand for customized controllers for solar thermal systems in single-family households. Such systems are used for water and often floor heating. There are more than a hundred setups to realize these systems depending on the vendor and country in which they are installed. The Thermius T80, a product of SunSil A/S, is a control unit for such a system. It can control 24 different setups of a solar thermal system. A typical setup for such a system is depicted in the following figure;



**Figure 1-1: Typical Setup for a Solar Thermal System**

The T80 regulates the pumps and valves in the system to control the flow of water in the pipes in order to efficiently and optimally heat up water for household use and for floor heating. If the water is not used quickly enough, it may attain unexpectedly high temperatures. As such the heat removal component at the top, close to the solar panel, dispatches the excess heat into the air. This may lead to as much as 70% loss of energy production by the solar panel.

Existing energy controllers in the market do have communication capabilities, but are used only for the purpose of remote monitoring and configuration. The implementation of intelligent distributed control for a communal warm water system is unique to this project.

There is ongoing research in distributed energy systems for a small local community. Here, a couple of households share energy from a thermal grid connected to a heat accumulator (located at the solar thermal plant). A household may or may not have a solar panel. A household with a solar panel dispatches extra heat it does not need to the grid and absorbs energy from the grid if

the panel produces insufficient energy. There are also heat pumps available to supplement the energy if the grid and panel do not provide enough energy. Households which extract heat from the grid are charged and when they dispatch heat to the grid, they are monetarily rewarded. This master project aims at developing communication protocols for the control devices located at each household in such a communal energy sharing system.

The more advanced controllers, which will be used for the distributed solar energy system, will be called T400. They will extend the functionality of the T80 to include communication protocols. They will thus be able to communicate with each other and other entities in the system. In this way, vital information could be exchanged to help increase the overall efficiency of the system as well as allow for monetary settlements.

## 1.1 Project Overview

This project is about the investigation and evaluation of the different options for communication between the entities of a distributed solar energy system. The controllers in this system use ARM9 processors with in-built Ethernet, GSM and USB connections. The USB can provide an interface for Bluetooth or WiFi connections. The main constraints in this project arise from security issues. The communication should provide mechanisms for authorization, authentication and validation. If the controllers are connected to the LAN of the household, the communication should be able to bridge firewalls and NATs.

A controller is located at each household and at the solar thermal plant of a communal system. A household may or may not have a solar panel. All households and solar thermal plant are connected to an energy grid which distributes warm water to the households in a community. A simple communal system is as depicted in the figure below;

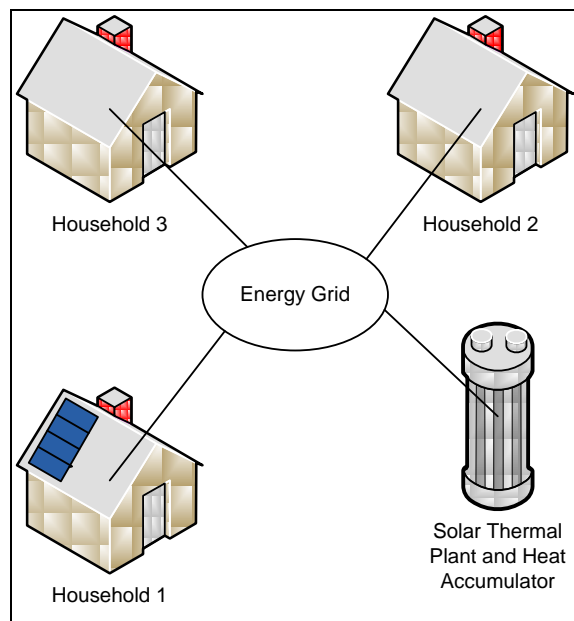


Figure 1-2: A Simple Communal System

A full system analysis and design was done in this project. This involved the identification of communication parties in a distributed solar energy system and the building of a communication infrastructure which enabled these parties to communicate with each other. As will be seen in the chapters that follow, the scope of this project was too much to be done in a single thesis. Even though a complete system analysis was done, only part of the project was actually realized.

## **1.2 Thesis Outline**

The thesis is outlined in such a way that each chapter discusses a different stage in the system development process of this project. Five of the software development steps, as described in [8], are done here. These include the analysis, design, specification, implementation and testing of the system.

Chapter 2 gives a complete requirement analysis of the system. The use cases of the energy grid and T400 energy controller are discussed in this chapter. The communication pairs in the system are identified as well.

Chapter 3 discusses the system design. The network architecture and design constraints are discussed in detail in this chapter.

In chapter 4, the system specification is discussed. Here, only the remote access of the T400 by a user is specified. This is done via a centralized server, which also manages a database. The database model is given and the service and protocol specification of the communication between the centralized server and a T400 device is also discussed.

In chapter 5, the system implementation is discussed. Here, an overview of the development environment is given and thereafter follows the implementation of the database and centralized server. The software architecture of the centralized server is discussed as well.

Chapter 6 discusses the test results. Here, the different web pages generated by the web server are discussed. The user view of each page is shown and it is explained how the user interacts with the different pages to carry out the actions specified in the requirements analysis.

Chapter 7 gives a conclusion about the project and general observations and difficulties encountered while carrying out the tasks involved in the project. It also gives some recommendations for future work.

## 2 Requirements Analysis

The main aim of the project is to design a communication infrastructure which allows for information exchange between the control devices in the system. The control devices in turn use this information to optimize thermal energy storage and consumption. In order to have a feel of the communication parties involved in the system and of what kind of information they may exchange, a simple requirements analysis was done for some key components in the system.

In this chapter a simple requirements analysis is done for the energy grid and the T400 controllers involved in the system. This is done with the help of UML use case diagrams. Finally a detailed requirements analysis is done for the communication infrastructure. Here, the communication between every set of parties involved in the system, is analyzed separately.

### 2.1 The Energy Grid

The energy grid consists of well insulated pipes which circulate warm water to the homes in the community. A solar thermal plant is connected to the grid. The homes in the community and the solar thermal plant may consume thermal energy from the grid or dispatch thermal energy to the grid. Below is a simple use case for the energy grid.

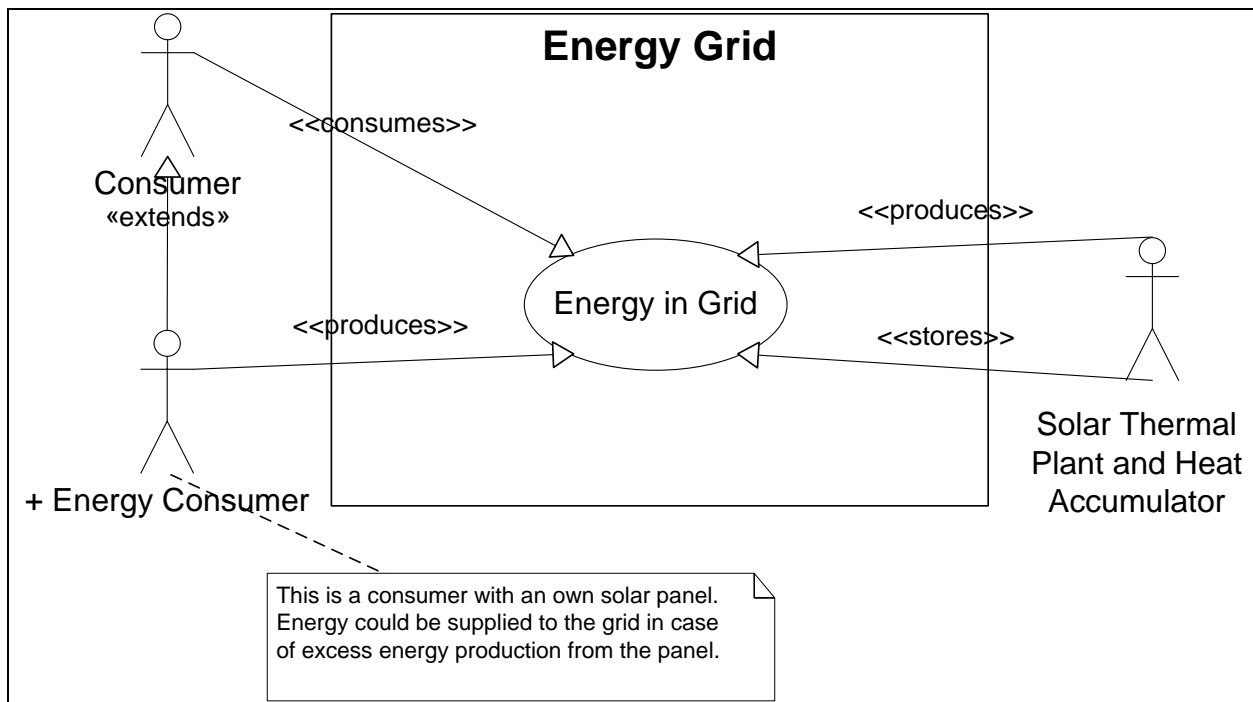


Figure 2-1: Simple Use Case of Energy Grid

A solar thermal plant supplies the grid with hot water from a tank heated up by a shared large solar panel. If there is little or no demand for thermal energy from the grid and there is excess

thermal energy entering the grid, this energy can be stored in the water tank if it has the capacity to absorb more energy. If the tank cannot absorb anymore energy, the solar thermal plant stops production to avoid overheating and signals all plus-energy households to stop dispatching excess thermal energy to the grid.

Thermal energy from the hot water in the grid is consumed by households (referred to as “Consumer” in the use case diagram above). A household pays for the energy it consumes from the grid.

A household may have an own solar panel which produces energy as well (referred to as “+ Energy Consumer” in the above diagram). If its panel does not produce enough energy, it consumes additional energy from the grid and pays for the energy as well. If its panel on the other hand produces more energy than needed, it dispatches the excess energy to the grid. In this case, the household is monetarily rewarded.

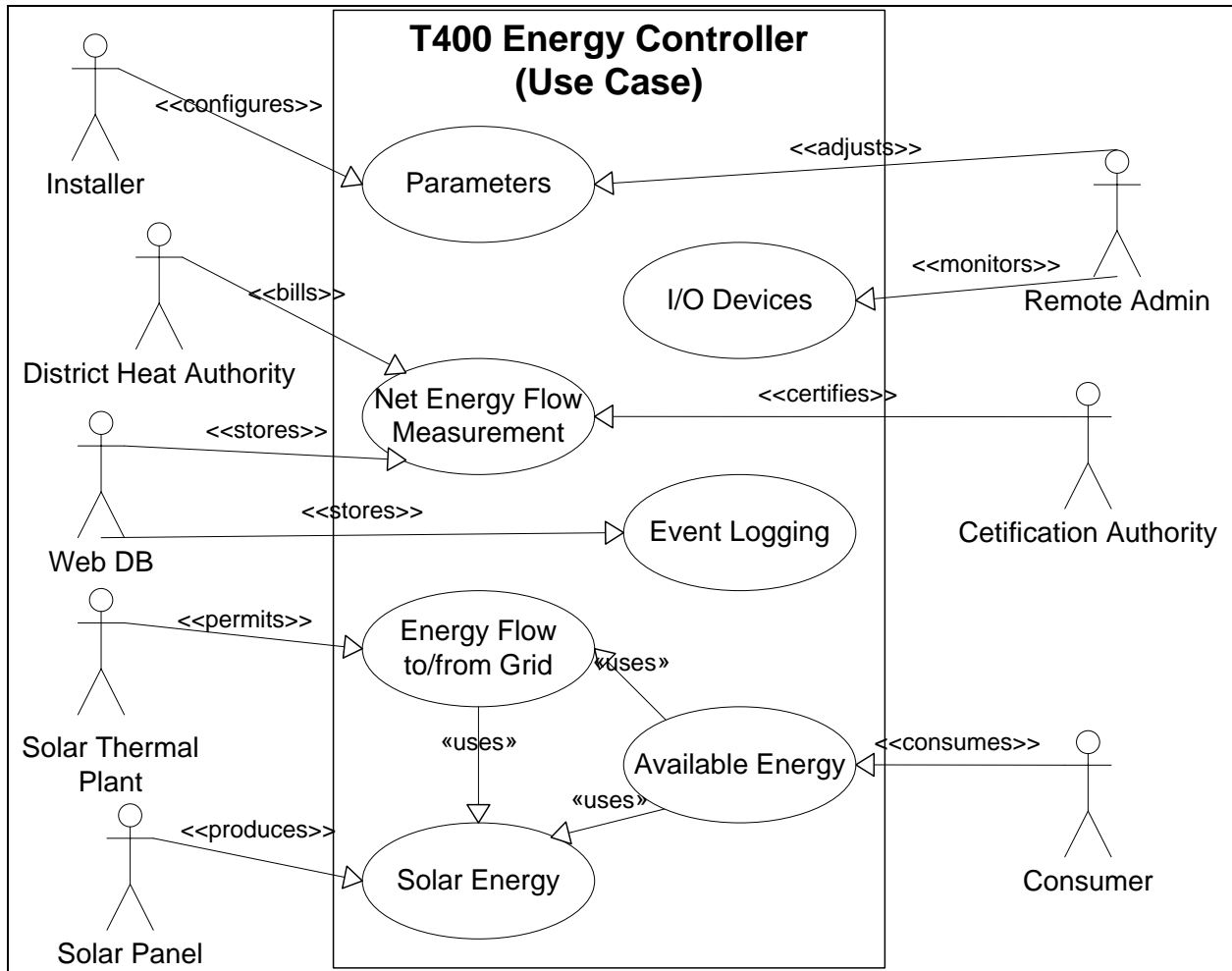
Each household and the solar thermal plant have a T400 controller which switches between energy sources to optimize energy consumption. There is a need for the controllers to communicate with each other to ensure that every household gets a fair share of the energy in the grid.

## **2.2 The T400 Energy Controller**

The T400 energy controller is a small box with an embedded system, located at each household and at the solar thermal plant. All intelligence in the system is programmed in these boxes. A T400 controller at the solar thermal plant has a somewhat different functionality as one at a household. The following two sub-sections discuss the use case and activity diagrams of a T400 at a household.

### **2.2.1 Use Case**

Below is the use case diagram of a T400 energy controller located at a household. It shows several actors who interact in some way with the controller and some of its basic functionality.



**Figure 2-2: Use Case of T400 Energy Controller**

The controller has a set of configuration parameters. These parameters are initially configured by the installer who also sets their limits. A remote administrator (referred to as “Remote Admin” in the above use case diagram) may adjust the parameters and monitor the external inputs and outputs.

The controller measures the net energy flow from and to the grid. This measurement is certified by a certification authority and the district heat authority uses the measurement to bill the customer. The measurement is also stored periodically, together with the log data, in a web database (referred to as “Web DB” in the above use case diagram).

The controller decides in an optimal way the energy source to use (see activity diagram in the following section). It must however be permitted by the solar thermal plant to absorb energy from the grid or dispatch energy to the grid. If the household has an own solar panel, it produces energy if need be. Extra energy from the solar panel may also be dispatched in the grid.

## 2.2.2 Activity Diagram

The following activity diagram shows the behavior of a T400 energy controller on a request for heat energy from a positive energy consumer, i.e. a household with an own solar panel. It shows the different energy sources and the criteria used by the controller to decide which energy source to use.

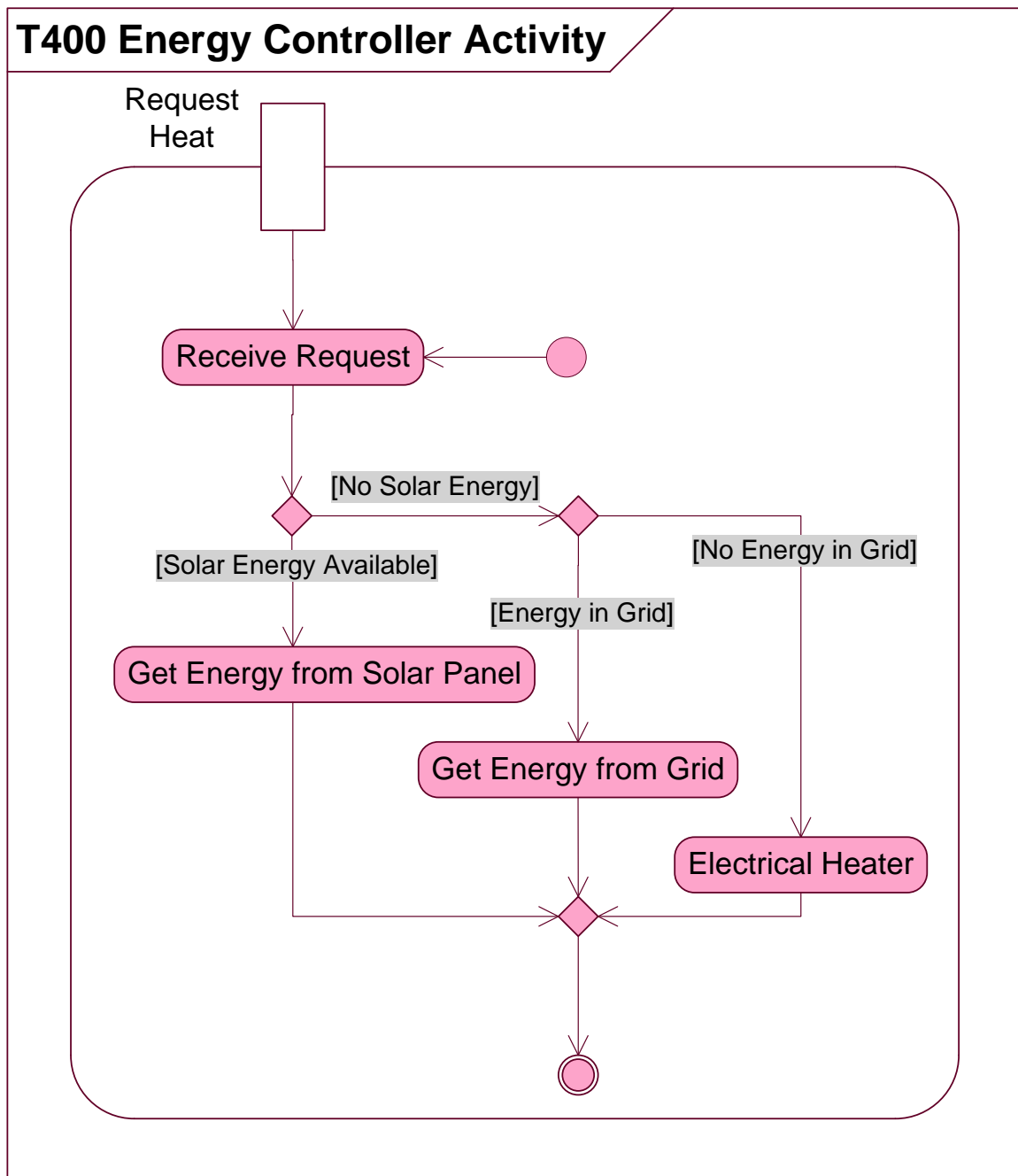


Figure 2-3: Activity Diagram of T400 Energy Controller



The energy sources are prioritized with the solar panel having the highest priority, because it is the cheapest source of energy, and the electric heater having the lowest priority, because it is most expensive. If there is not enough solar energy from the panel, energy is extracted from the grid, if the solar thermal plant permits, else it is taken from the electric heater.

## **2.3 Communication Requirements**

From the requirements analysis so far, it can be observed that the T400 devices do not exclusively communicate among themselves. They also communicate with actors out of the communal system. This will be done over the internet, using the World Wide Web. So it will be preferable to have the T400 devices connected in an IP-based network. This will eliminate the necessity of a gateway to translate the protocol to IP. If the communication breaks down, a T400 device will function as a normal energy controller for a single household.

From the use case of the T400 controller, four different communication pairs can be identified;

- i) T400 – T400
- ii) T400 – District Heat Authority
- iii) T400 – Web DB
- iv) T400 – Remote Administrator

The following sub-sections discuss the requirements for the communication pair of entities listed above. As will be seen, the communication between the first three pairs of entities is rather trivial compared to the fourth pair. Due to time constraints, only the communication of the fourth pair was done in this project, since it is most challenging. More so, SunSil does not permit, for proprietary reasons, the publication of the communication for the first three pairs. As such, the discussion from chapter three is solely based on the communication for the fourth pair.

### **2.3.1 T400 – T400**

All T400 devices in a communal system should have easy and direct access to each other. There is no inter-communication or thermal energy exchange between devices located in different communities, i.e. each communal system is closed and its operation has nothing to do with the others. Since the system deals mostly with temperature regulation, which is a rather slow process, the real time delay of the system could go up to several minutes without affecting the operation of the system.

T400 devices in a communal system exchange their states and possibly some configuration parameters to make them aware of their environment. In this way, they could together coordinate the production and use of energy to optimize the system. The communication protocol should thus be as open as possible to allow for exchange of any kind of information.

Cost minimization and reliability are always important issues in every system. The T400 devices should not rely on the internet connections of the households or their telephone lines. These connections are paid for and controlled by the home owners. They may turn off their modem or fall short of paying a bill leading to their connection being blocked. This will inhibit the devices

from communicating making the system very unreliable. As such, the T400 devices should have their own communication links. On the other hand, wiring the devices together will be expensive, so the devices should be linked preferably by some wireless communication.

The devices in a single communal system should all belong to the same LAN, which should have restricted access. As such, each device will have direct access to the other devices in its community.

### ***2.3.2 T400 – District Heat Authority***

The district heat authority is a web server connected to the internet. It has a public IP-address and a global URL which can be resolved by the DNS. The T400 devices on the other hand are located in a LAN and all have only private IP-addresses. So, only a T400 device can initiate a connection to the web server of the district heat authority. This is done periodically to communicate any necessary information as discussed in section 2.2.1. A reasonable time interval in which the T400 devices communicate with the web server of the district heat authority was not investigated, but is expected to be in the range of daily to weekly. The range is also expected not to pose a burden on energy consumption or availability of the device.

### ***2.3.3 T400 – Web DB***

The communication between these two parties is similar to that discussed in the previous section. The T400 device initiates the connection and information is periodically communicated. The time interval for this communication was also not investigated.

### ***2.3.4 T400 – Remote Administrator***

In contrast to the communication pairs discussed in the previous two sections, the remote administrator initiates the connection for this communication. This makes it much more challenging since the T400 devices have no public IP-address and the connection is externally initiated, i.e. out of the communal LAN of a group of T400 devices.

Several solutions to bridge the above stated problem are discussed in the next chapter. The solution however should provide high accessibility, reliability and restricted access. It should also provide easy access from any computer with a browser and internet connection. A remote administrator should need no special hardware or software in order to access the device.

### 3 System Design

The discussions in this chapter and subsequent ones focus on the communication between a T400 device and a remote administrator. This chapter starts by discussing the network architecture which shows how the communication parties are connected to each other, followed by a discussion of the design constraints and finally a discussion of several solutions taken into consideration. The advantages and disadvantages of each solution are weighed and reasons are given to back the final design solution.

#### 3.1 Network Architecture

The T400 devices in a single community are connected in a wireless Virtual Private Network (VPN). As such, a device has direct access to all other devices in its community. Each VPN has a gateway to the internet which enables its devices to be accessed remotely and to communicate with the district heat authority and the web database (see chapter 2 for the functionality of the district heat authority and web database). The following figure shows the network architecture with two VPNs. Of course, there can be more VPNs depending on how many communities exist.

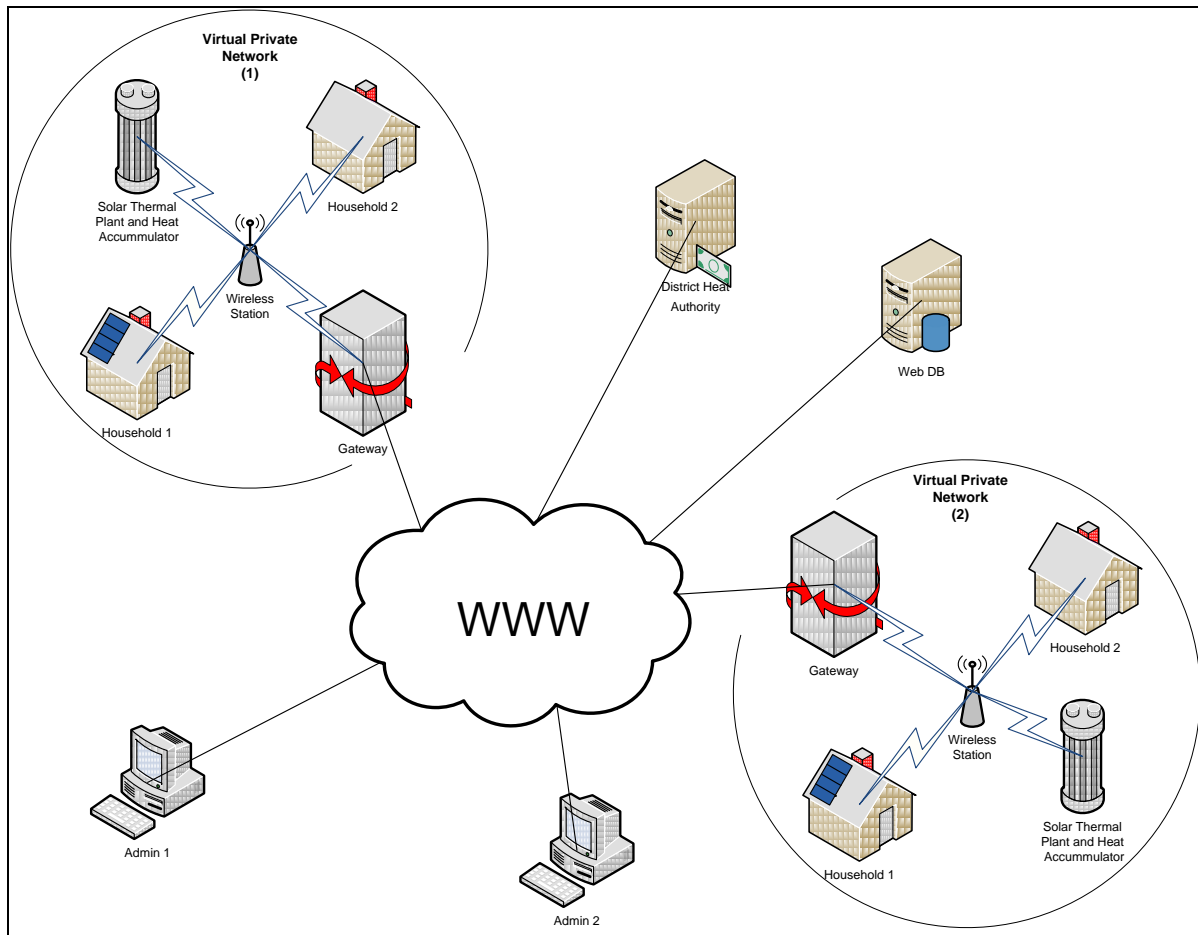


Figure 3-1: Network Architecture for Communal Heating Systems

An energy sharing community is made up of a solar thermal plant and several households, which all have a T400 energy control device. These devices communicate with each other to optimize energy production. They are connected in a VPN provided by a mobile operator. The operator also provides a gateway to the internet. Each device thus has a SIM-card to give them access to the mobile network. They then communicate over GPRS.

A remote administrator, the district heat authority and the web database all communicate with T400 devices over the internet. As mentioned before in chapter 2, devices in different communities do not communicate with each other. So from the above figure, a device in VPN(1) for example does not communicate with one in VPN(2).

### 3.2 Design Constraints

The main design constraint is initiating a connection to a device from without its network, in the case of remote access. These connections are blocked by firewalls and the devices themselves have no public IP-addresses. In the following sub-sections several solutions are discussed to overcome this constraint and their advantages and disadvantages are weighed to get the best solution.

#### 3.2.1 Solution 1: Direct Access

In this case, a remote administrator connects directly from a PC to a T400 device. A TCP connection cannot be initiated from the PC because the PC does not belong to the VPN of the T400 devices, which have only private IP-addresses. The following two figures illustrate two possibilities of overcoming this problem. A discussion of their advantages and disadvantages follow thereafter.

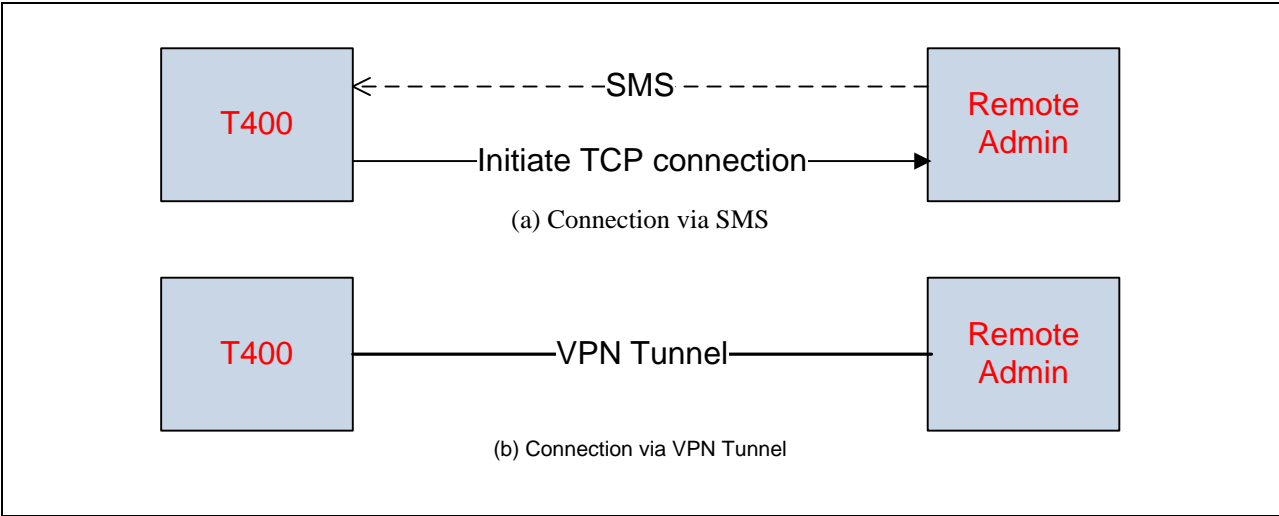


Figure 3-2: Direct Remote Access to T400 Device

In figure 3-2(a), the PC of the remote administrator sends an SMS to the T400 device, telling it to initiate a TCP connection to the PC. This SMS contains the IP and port of the remote administrator's PC. The PC then listens to this port and waits for the connection from the T400 device. After the connection has been established, the remote administrator has to authenticate himself. For this to work, the PC of the remote administrator will need to meet the following requirements;

- A GSM modem for sending SMS
- Software to send the SMS and listen to the appropriate port
- A public IP-address for the TCP connection since the PC is not part of the VPN of the T400
- The firewall of the PC's LAN should be configured to free the port

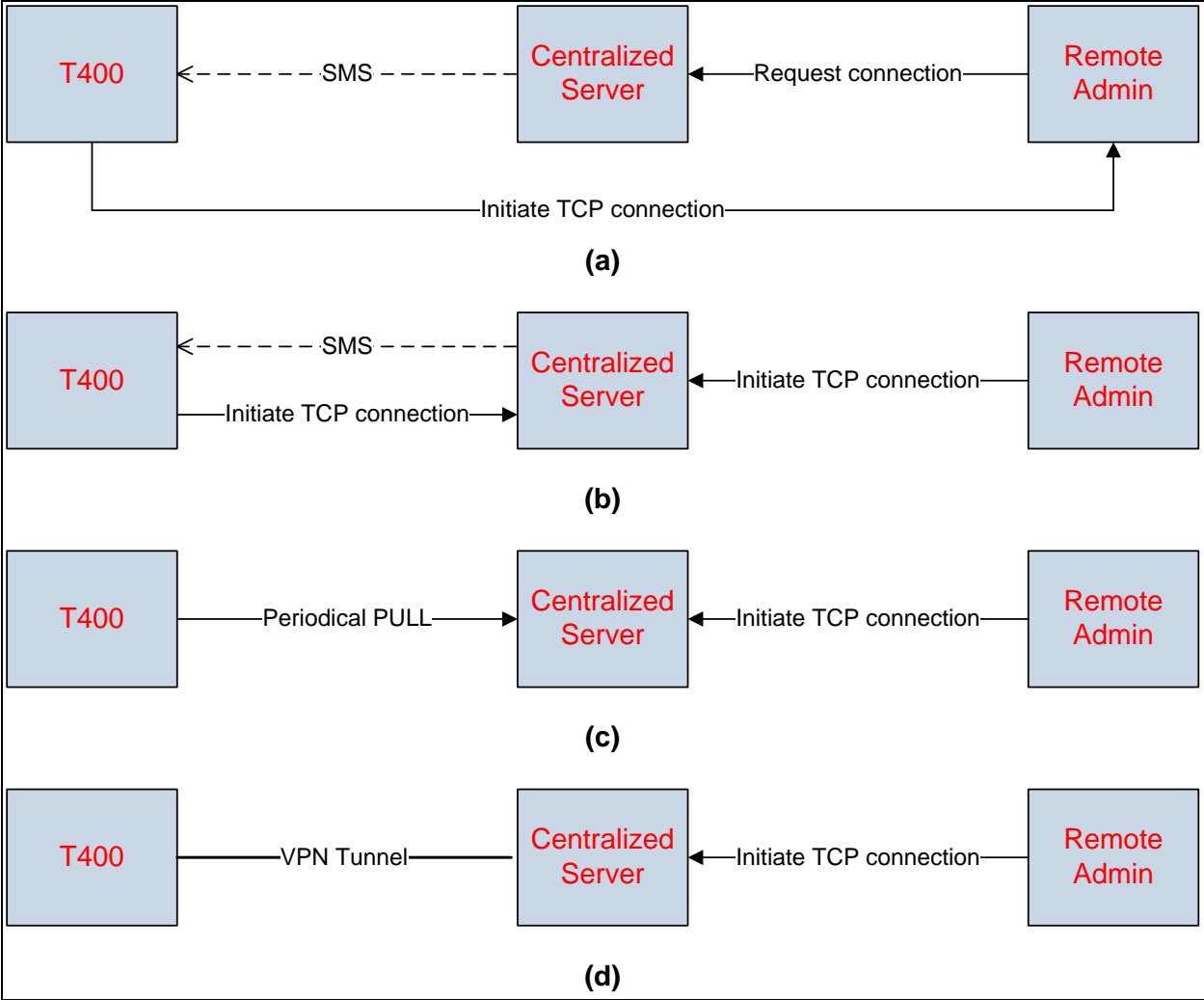
As mentioned in section 2.3.4, high reliability and easy access is required for the remote administrator. SMS is an unreliable and unconfirmed service. There is no guarantee as to when the SMS would arrive and if it would arrive at all. Moreover, the PC of the remote administrator needs special hardware and software, and even a pre-configuration in its LAN, in order to access the T400 device. This results in the devices having no easy remote access.

Figure 3-2(b) shows a much easier way of achieving direct remote access to a T400 device. The PC of the remote administrator builds a VPN tunnel to the network of the T400 device. As such, the PC is virtually part of the device's network and it can initiate a TCP connection to any device in that network. The main problem with this solution is accessibility. Mobile operators offer only a single VPN tunnel to each network they provide. So, only one remote administrator has exclusive access to all devices in a VPN. Others trying to access any device in the VPN will have to wait. Also, the remote administrator has to install the VPN client software on his PC. The T400 device on the other hand could run a small web server, permitting the remote administrator to connect to it using a browser.

In both cases discussed above, the management of users is decentralized, i.e. each T400 device would have to manage the users permitted to access it. If a single user has to be granted access to a group of devices, then this user has to be added to each of the devices. Also, a remote user has to know the telephone number (for solution (a)) or IP-address (for solution (b)) of each device it wants to access.

### ***3.2.2 Solution 2: Access via a Centralized Server***

As will be seen from the discussions that follow, making use of a centralized server, which acts as a relay between the T400 devices and remote administrators, brings a lot of advantages with it. Requirements imposed on the devices and the PC of the remote administrator by the solutions discussed previously could be transferred to the centralized server. This will greatly reduce the amount of resources required for communication on the devices as well as the requirements of the remote administrator's PC. The following figure shows four possible ways of making use of a centralized server;



**Figure 3-3: Remote Access to T400 Device via a Centralized Server**

The requirements for the T400 device and the PC of the remote administrator in figure 3-3(a) above, are same as those in figure 3-2(a) discussed in the previous section, but for the fact that the PC of the remote administrator in this case does not need a GSM modem. The job of sending the SMS to the T400 device is taken over by the centralized server. But it still needs a special software and LAN configurations as discussed in the previous section. It therefore has little improvements over that solution.

In figure 3-3(b), the need of a special software on the PC of the remote administrator and LAN configurations are shifted to the centralized server. This greatly reduces the requirements of the remote administrator’s PC. The centralized server acts as a relay between the T400 device and the remote administrator and there is no direct communication between them. As such the centralized server can provide a web interface to the remote administrator, thus enabling the use of a browser by the remote administrator. Moreover, user management, authentication and authorization can be handled by the centralized server. This greatly reduces the demand for

resources on the T400 device which would otherwise be responsible for that. Furthermore, multiple remote administrators communicating with a single device at once can share a single connection from the device to the centralized server. This further reduces the demand for resources on the T400 device to manage multiple connections. However, the use of SMS still makes it unreliable as discussed in the previous section.

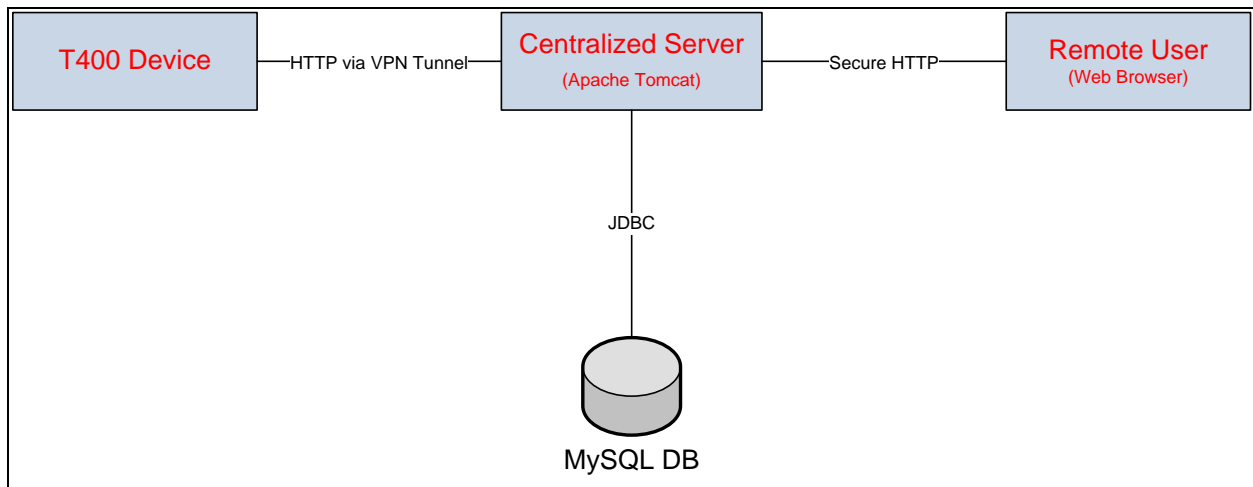
In figure 3-3(c), the T400 device periodically pulls the centralized server to find out if any administrator wants to connect to it. If there is none, the connection is then terminated. On the other hand, a remote administrator, who wants to connect to a device, contacts the centralized server to find out if it has a connection to the required device. If it does not, then the remote administrator will have to wait until the device pulls the server again. This solution causes delayed access on the side of the remote administrator and too much traffic on the side of the device. Generally speaking, mobile networks have very limited bandwidth and pulling is no good practice in mobile networks. It is better to have the server push available information to the device rather than have the device pull periodically to find out if there is available information for it. This brings us to our final solution, which is also the solution used in this project.

Figure 3-3(d) is the solution used in this project. The centralized server has a VPN tunnel to every communal network of the T400 devices. This makes the centralized server part of all the networks. The centralized server provides a web interface to all remote administrators. A remote administrator, who wants to connect to a device, makes a simple URL request to the server and the server initiates a TCP connection to the required device. It then relays information between the device and the administrator. This solution provides highest accessibility and reliability compared to the previous solutions. It also makes use of a push operation rather than a pull. This greatly minimizes the network load. Security is an issue here, in which case users have to authenticate themselves.

## 4 System Specification

This chapter gives a detailed specification of the communication between a T400 device and a remote administrator. They communicate via a centralized server as shown in figure 3-3(d). T400 Device management, user management and the management of access rights are done by the centralized server with the help of a database. Users from here on refer to the administrators.

The system specification is done using a top-down approach. First an overview of the whole system is given in the following figure. Here, all key components in the system and how they interact with each other are shown. Then a specification of each component is given.



**Figure 4-1: System Specification**

The user from a remote PC can connect to the centralized server using a web browser. HTTP is used for the communication and the user has to authenticate. A user may be an administrator, who is given access rights to all or a group of T400 devices, or a normal household owner (may be on holidays) who can access only his T400 device at his home. This is an additional attractive feature of this solution, to improve owner satisfaction. A user, logged on to a device, can adjust parameters on it and can view the I/O values, energy production and log data of the device.

The centralized server is an Apache Tomcat web server with a servlet engine. It uses a MySQL database which stores the credentials and rights of all users as well as the details of all T400 devices. It also provides a web interface for querying and manipulating the database. It communicates with the database using a JDBC driver for MySQL. The specification of the database is discussed in section 4.1.

A user, who wants to access a T400 device, first logs on to the centralized server. The centralized server then displays only devices which the logged on user is allowed to access. The user then selects a device and the centralized server connects to this device through a VPN tunnel. HTTP is also used for the communication between the centralized server and a T400 device. The



centralized server then acts as a relay between the user and the device. Requests from the user are forwarded to the device and the device responses are forwarded to the user.

## 4.1 Database Model

As already mentioned, the MySQL database of figure 4-1 is used by the centralized server to store information necessary for the management of users, devices and access rights. It is also required by the Apache Tomcat servlet container for authentication. The servlet container reads the name, password and roles of a user from the database. Below is the database model;

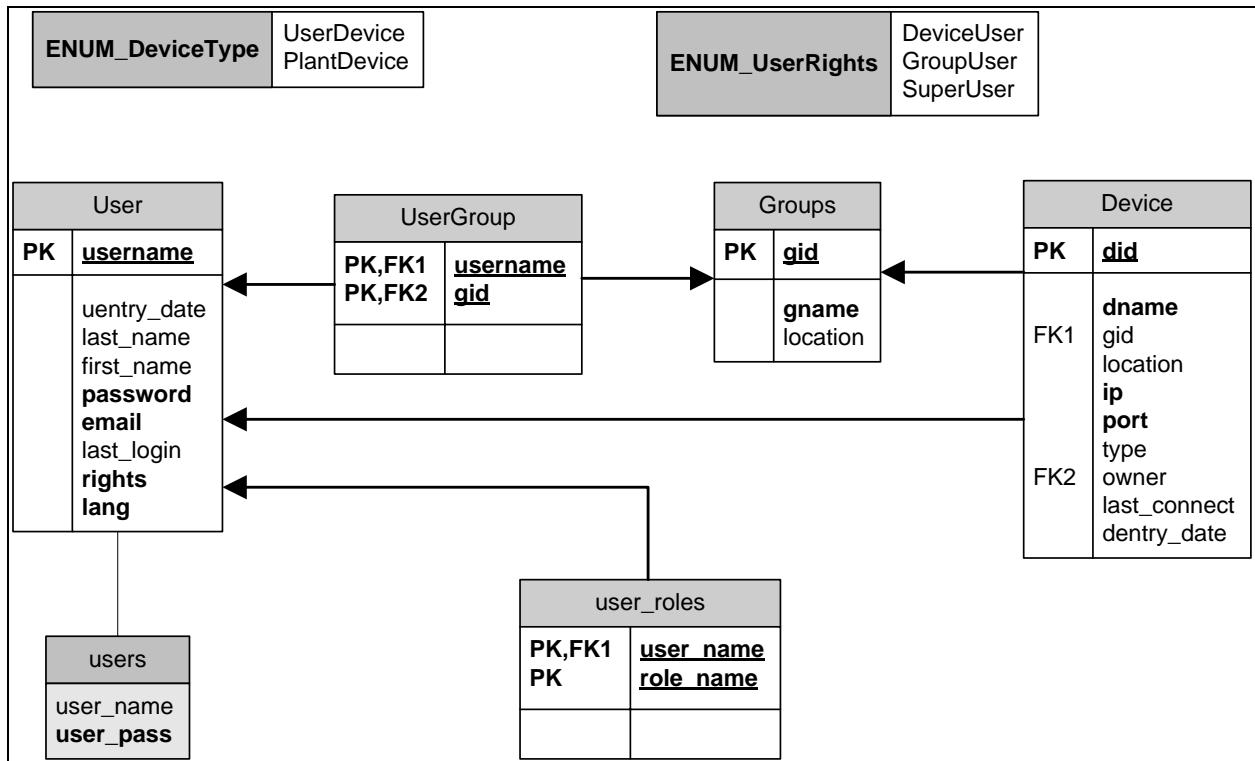


Figure 4-2: Database Model

The *type* column in the *Device* table is an enumeration of *ENUM\_DeviceType* and the *rights* column in the *User* table is an enumeration of *ENUM\_UserRights*. The underlined columns in a table indicate its primary key (PK) while the columns in bold are required. The columns marked with FK are foreign keys. The use of the tables in the above figure and their relationships are explained in the following sub-sections.

### 4.1.1 User Table

This table stores the credentials of all users who would administrate a T400 device. There are three kinds of users; *SuperUser*, *GroupUser* and *DeviceUser*, differentiated by their access rights. A *SuperUser* can add, delete and modify all T400 devices and all other users. He can also

access all devices. A *GroupUser* can add, modify, delete and access only T400 devices in its group and can only add *DeviceUsers*. A *DeviceUser* can access only devices he owns. This is specified by the *owner* column of the *Device* table. All users can edit their profiles but cannot change their rights. Below is an explanation of each column in this table;

<i>username</i>	– PK of this table which uniquely identifies each user
<i>password</i>	– user’s password for authentication
<i>last_name</i>	– user’s last name
<i>first_name</i>	– user’s first name
<i>email</i>	– user’s e-mail address
<i>rights</i>	– user’s access right (may be <i>SuperUser</i> , <i>GroupUser</i> or <i>DeviceUser</i> )
<i>lang</i>	– user’s language (default value of <i>en</i> for English)
<i>last_login</i>	– indicates when a user logged in last
<i>uentry_date</i>	– constant value which indicates when a user was added to the database

When a user profile is modified or added to the database by another user, he is informed per e-mail of the modification and his username and password. The website of the centralized server may be in several languages. The language chosen by the user is stored in the database, so that the same language is used for his next login.

#### 4.1.2 *Group Table*

This table stores each community of T400 devices. Its columns have the following uses;

<i>gid</i>	– group identifier which is an auto-generated primary key
<i>gname</i>	– a unique and meaningful group name to identify each community
<i>location</i>	– the location of each community

#### 4.1.3 *UserGroup Table*

This table associates a *GroupUser* to a group. A *GroupUser* may be in several groups, i.e. be responsible for all devices in several groups. The table has just two columns; *username* and *gid*, which both serve as its unique PK. Both columns are FK’s which respectively point to the *username* of the *GroupUser* and the *gid* of the group the user belongs to.

#### 4.1.4 *Device Table*

This table stores the details of each T400 device. There are two types of devices; *UserDevice* and *PlantDevice*. This is specified by the *type* column. A *UserDevice* is located at a household in the community and has an owner. A *PlantDevice* on the other hand controls the solar thermal plant, as shown in figure 3-1, and has no owner. The columns of this table are as explained below;

<i>did</i>	– device identifier which is an auto-generated primary key
<i>dname</i>	– unique and meaningful device name
<i>gid</i>	– FK which points to the group the device belongs to
<i>location</i>	– the location of the device

<i>ip</i>	– the IP-address of the device
<i>port</i>	– the connection port of the device
<i>type</i>	– the device type (may be PlantDevice or UserDevice)
<i>owner</i>	– the owner of the device
<i>last_connect</i>	– the date and time the server last connected to the device
<i>dentry_date</i>	– the date and time the device was added to the database

The most important columns here are the *ip* and *port*. The centralized server uses these values to connect to the device.

#### 4.1.5 Users View

This view is used exclusively by the Apache Tomcat servlet container for authentication. It is a copy of the *username* and *password* of the *User* table. These columns are renamed to *user\_name* and *user\_pass* in this view.

#### 4.1.6 User\_roles Table

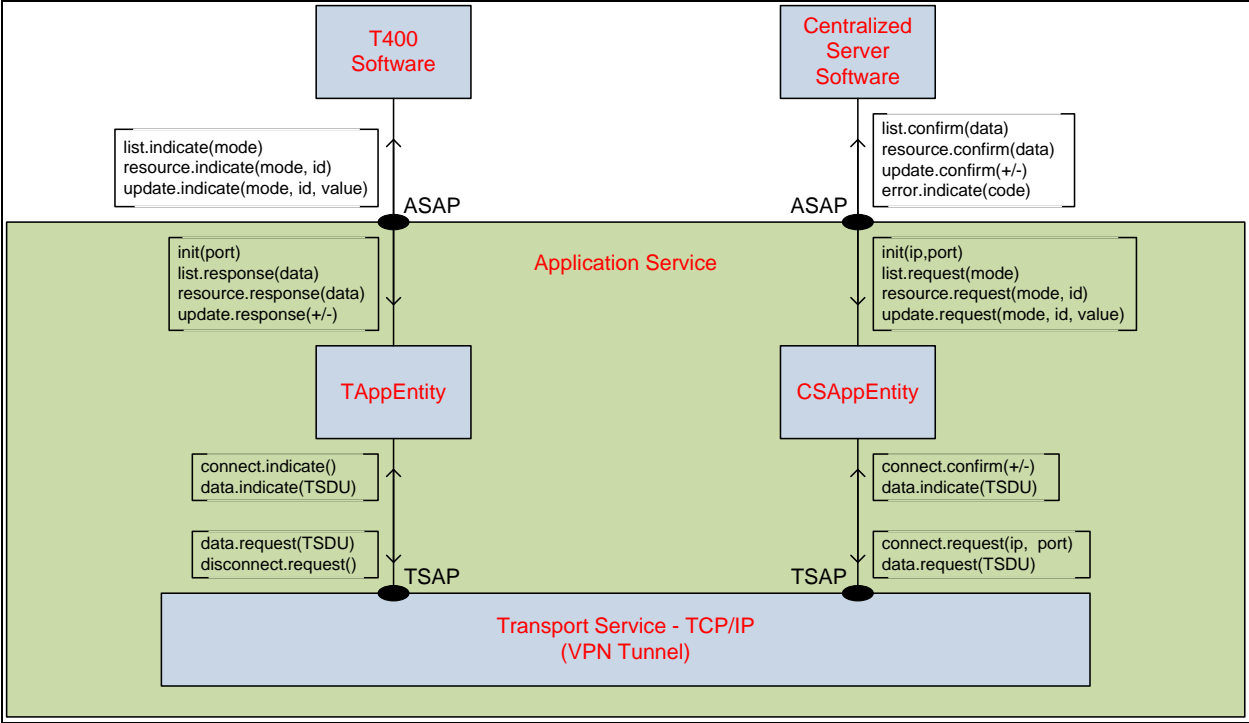
This table is also used exclusively by the Apache Tomcat servlet container for authentication. An Apache Tomcat server can run several applications, each having its own container. Only users with specific roles are allowed to access a container. As such, the Apache Tomcat server uses this table to identify the roles of each user.

This table has only two columns which both serve as PK for the table. The column *user\_name* points to a *username* in the table *User* and the column *user\_role* specifies the role of the user. A single user may have several roles.

## 4.2 Service Specification

The communication between the centralized server and the PC of the remote user is done by the Apache Tomcat web server and the browser, using HTTP. So, well defined standard protocols are used here. It is left just to implement the server to provide the necessary HTML web pages for the remote user.

On the side of the T400 device, the services and protocols still have to be specified. In this section, the service specification for the communication between the T400 device and the centralized server is done using SDL service diagrams (see [1] & [2]). The following figure shows the flow of the various service signals through their Service Access Points (SAP);



**Figure 4-3: Service Specification**

The application service offers the init, update, list, resource and error services to the T400 and centralized server software. The services are discussed in detail in the sub-sections that follow. It has two entities; the T400 Application Entity (*TAppEntity*) and the Centralized Server Application Entity (*CSAppEntity*). They communicate virtually with each other by exchanging Protocol Data Units (PDU) as discussed in the protocol specification (see section 4.3). They communicate with their associated software through the Application SAP (ASAP) using service signals, as shown in the figure above.

The transport service offers the connect, data and disconnect services to the application entities. Service signals are exchange between the application entities and the transport service through the Transport SAP (TSAP).

**4.2.1 Service Signal**

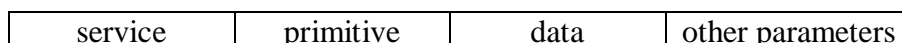
The following table shows a summary of the service signals used in the communication between a T400 device and the centralized server.

Service Access Point (SAP)	Service	Primitive	Source	Destination	Parameters
Application Service Access Point (ASAP)	init		T400 Software	TAppEntity	port
			Centralized Server Software	CSAppEntity	ip, port
	list	request	Centralized Server Software	CSAppEntity	mode
		indicate	TAppEntity	T400 Software	mode
		response	T400 Software	TAppEntity	data
		confirm	CSAppEntity	Centralized Server Software	data
	resource	request	Centralized Server Software	CSAppEntity	mode, id
		indicate	TAppEntity	T400 Software	mode, id
		response	T400 Software	TAppEntity	data
		confirm	CSAppEntity	Centralized Server Software	data
	update	request	Centralized Server Software	CSAppEntity	mode, id, value
		indicate	TAppEntity	T400 Software	mode, id, value
		response	T400 Software	TAppEntity	+/-
		confirm	CSAppEntity	Centralized Server Software	+/-
error	indicate	CSAppEntity	Centralized Server Software	code	
Transport Service Access Point (TSAP)	connect	request	CSAppEntity	Transport Layer	ip, port
		indicate	Transport Layer	TAppEntity	
		confirm	Transport Layer	CSAppEntity	+/-
	data	request	TAppEntity / CSAppEntity	Transport Layer	TSDU
		indicate	Transport Layer	TAppEntity / CSAppEntity	TSDU
	disconnect	request	TAppEntity	Transport Layer	

**Table 4-1: Summary of Service Signals**

Service signals are made up of four parts; the service, the primitive and some parameters. The service specifies the task of the signal. All services are discussed in detail in the following sub-sections. The primitive tells the direction of the signal and its reason. There are four types of primitives; request, indicate, response and confirm. The parameters contain the information exchanged by the peer entities.

A request primitive is sent downwards and is used by a communication entity to invoke a service offered by the layer beneath it. An indicate primitive is sent upwards and is used by a communication entity to notify the layer above it of a service request made by its peer entity. A response primitive is sent downwards and is used by a communication entity to respond to a service indication. Finally, the confirm primitive is sent upwards by a communication entity to notify the layer above it of a service response to a request it made. A service signal has the following structure;



**Figure 4-4: Structure of a Service Signal**

Data in the above structure is a special parameter. It is a CSV-file (Comma Separated Values). This is the format used in resource files obtained from the T400 device.

### 4.2.2 *Init Service*

This is an application service and it creates and initializes instances of the application entities. It has no primitives. The *TAppEntity* is initialized with the *port* it listens to while the *CSAppEntity* is initialized with the *ip* and *port* of the T400 device it should connect to. So in this setting, the centralized server initiates the connection to the T400 device.

### 4.2.3 *List Service*

This is an application service. It is used by the centralized server to get the list of resources from the T400 device. Resources are CSV-files containing some data from the T400. This service is confirmed and has four primitives; *request*, *indicate*, *response* and *confirm*.

The centralized server sends a request with the service signal *list.request(mode)* to the *CSAppEntity*, which communicates the request to the *TAppEntity*. On receipt of this request by the *TAppEntity*, it notifies the T400 software with the service signal *list.indicate(mode)*. The parameter *mode* specifies the access mode, which can either be 0 for user mode or 1 for install mode. The T400 device uses this parameter to restrict access.

The T400 software responds by sending the service signal *list.response(data)* to the *TAppEntity*, which further communicates this to the *CSAppEntity*. The *CSAppEntity* then notifies the centralized server software of the response by sending it the service signal *list.confirm(data)*. The parameter *data* in these signals is a CSV-file containing the available resources on the T400 device. It has two columns, *id* and *title*, which give the identifier and name of the resource.

When a user initially connects to a device, the centralized server invokes this service to get the list of resources the user is allowed to access from the device. It then displays the resource names to the user for selection. On selection, the server invokes the resource service to fetch the selected resource.

### 4.2.4 *Resource Service*

This is an application service and it is used by the centralized server to fetch a resource (i.e. a CSV-file) from the T400 device. Each resource has a unique ID. The service is confirmed and has four primitives namely; *request*, *indicate*, *response* and *confirm*.

The server fetches the resource by sending the service signal *resource.request(mode, id)* to the *CSAppEntity*, which communicates this to the *TAppEntity*. On receiving the request, the *TAppEntity* notifies the T400 software of the request by sending it the service signal *resource.indicate(mode, id)*. The parameter *id* is the unique identifier of the resource and *mode* is used to restrict access as discussed in the list service specification.

The T400 software responds to the server's request by sending the service signal *resource.response(data)* to the *TAppEntity*. The *TAppEntity* then communicates this response to the *CSAppEntity*, which further notifies the centralized server software by sending it the service signal *resource.confirm(data)*. The parameter *data* is a CSV-file of the requested resource.

This service is used for monitoring I/O-devices as shown in the use case of a T400 device in the requirements analysis. A resource may display the details and state of I/O-devices or parameters on the T400 device. A user may decide to change the value of a parameter, in which case the update service is invoked.

#### **4.2.5 Update Service**

This is an application service as well and it is used by the centralized server to change the value of a parameter on the T400 device. Every parameter has a unique id. The service is confirmed and has four primitives; *request*, *indicate*, *response* and *confirm*.

The *request* is made by the centralized server software to the *CSAppEntity* and the parameters *id* and *value* are passed. This is done with the service signal *update.request(mode, id,value)*. The *indicate* is made by the *TAppEntity* to the T400 software, when it receives the *request* from the server. This is done with the service signal *update.indicate(mode, id,value)*. The parameter *id* identifies the parameter to be modified on the T400 device while *value* is the new value of the parameter. The *mode* parameter is as discussed in the list service specification.

The T400 software responses to an update request by sending the service signal *update.response(+/-)* to the *TAppEntity* which further communicates this to its peer, the *CSAppEntity*. The *CSAppEntity* then notifies the centralized server software with the service signal *update.confirm(+/-)*. Positive is sent if the parameter update is successful else negative is sent.

This service is used to adjust parameters on the T400 device as shown on its use case in the requirements analysis. An appropriate message is displayed to the user indicating the outcome of the update.

#### **4.2.6 Error Service**

This is an application service and it is used by the *CSAppEntity* to notify the centralized server software of any errors which occur while processing a request from it. The service has a single primitive *indicate*. The notification is done by sending the service signal *error.indicate(code)* to the centralized server software. The parameter *code* is an error code which uniquely identifies the error which occurred.

#### **4.2.7 Connect Service**

This is a transport service which is used by the *CSAppEntity* to create a TCP/IP connection to the T400 device. It is a confirmed service and has three primitives namely; *request*, *indicate* and *confirm*.

When the *CSAppEntity* is initialized with the *init* service, it sends the service signal *connect.request(ip,port)* to the transport layer. The parameters *ip* and *port* are the IP-address and port of the T400 device to connect to.

The transport layer, on receipt of a connection request, creates a TCP/IP connection to the T400 device. If the connection was successful, the transport layer sends the service signal *connect.indicate()* to the *TAppEntity* and the service signal *connect.confirm(+)* to the *CSAppEntity*. If the connection failed, the transport layer sends the service signal *connect.confirm(-)* to the *CSAppEntity*.

#### **4.2.8 Data Service**

This is also a transport service but it is used by the application entities to exchange data. It is an unconfirmed service and has just two primitives namely; *request* and *indicate*.

An application entity sends data to its peer by sending the service signal *data.request(TSDU)* to the transport layer. The transport layer, on the other hand, notifies an application entity of available data from its peer by sending it the service signal *data.indicate(TSDU)*. The parameter *TSDU* is known as the Transport Service Data Unit, which comprises of an Application Protocol Data Unit (APDU) encoded in the Application Protocol Control Information (APCI). The data units and control information are specified in section 4.3.1.

#### **4.2.9 Disconnect Service**

This is a transport service used by the T400 device to terminate the TCP connection to the centralized server. When a connection is established to a T400 device, the *TAppEntity* sets up a timer for which the connection is kept alive. The timer is reset each time there is data communication. If the set time expires, the connection is terminated by the *TAppEntity*. This service is used exactly for this purpose in which the *TAppEntity* sends the service signal *disconnect.request()* to the transport layer. The service is unconfirmed and has just the single primitive *request*.

### **4.3 Protocol Specification**

Each service signal (except the *init*) from the application software to the application entities generates a PDU, which is communicated between the peer application entities. The sequence diagram below shows the PDU's of the application service;



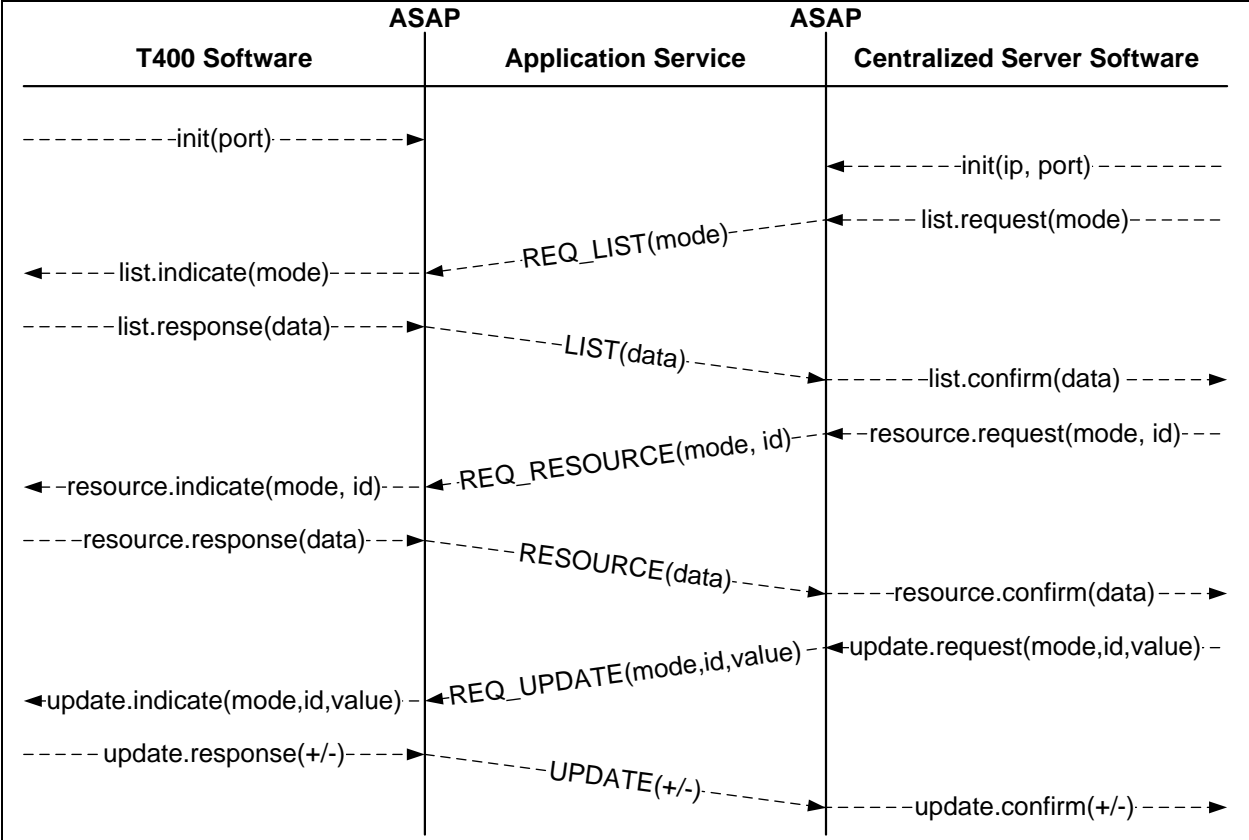


Figure 4-5: Sequence Diagram

The PDU *REQ\_LIST(mode)* communicates the request of the list service by the centralized server software to the T400 software while the PDU *LIST(data)* communicates the response. The PDU *REQ\_RESOURCE(mode, id)* communicates a resource request from the centralized server software to the T400 software while the PDU *RESOURCE(data)* communicates the response. The PDU *REQ\_UPDATE(mode, id, value)* communicates an update service request from the centralized server software to the T400 software while the PDU *UPDATE(+/-)* communicates the response.

**4.3.1 Static Protocol Specification**

The static protocol specification gives the internal structure of the application entities. This specification is done using SDL block diagrams (see [1] & [2]). The entities have processes which communicate with each other by exchanging messages. The *TAppEntity* will be discussed first and then follows the *CSAppEntity*. The figure below shows the internal structure of the *TAppEntity*. It has three processes namely; *Timer*, *TCom* and *Codec*.

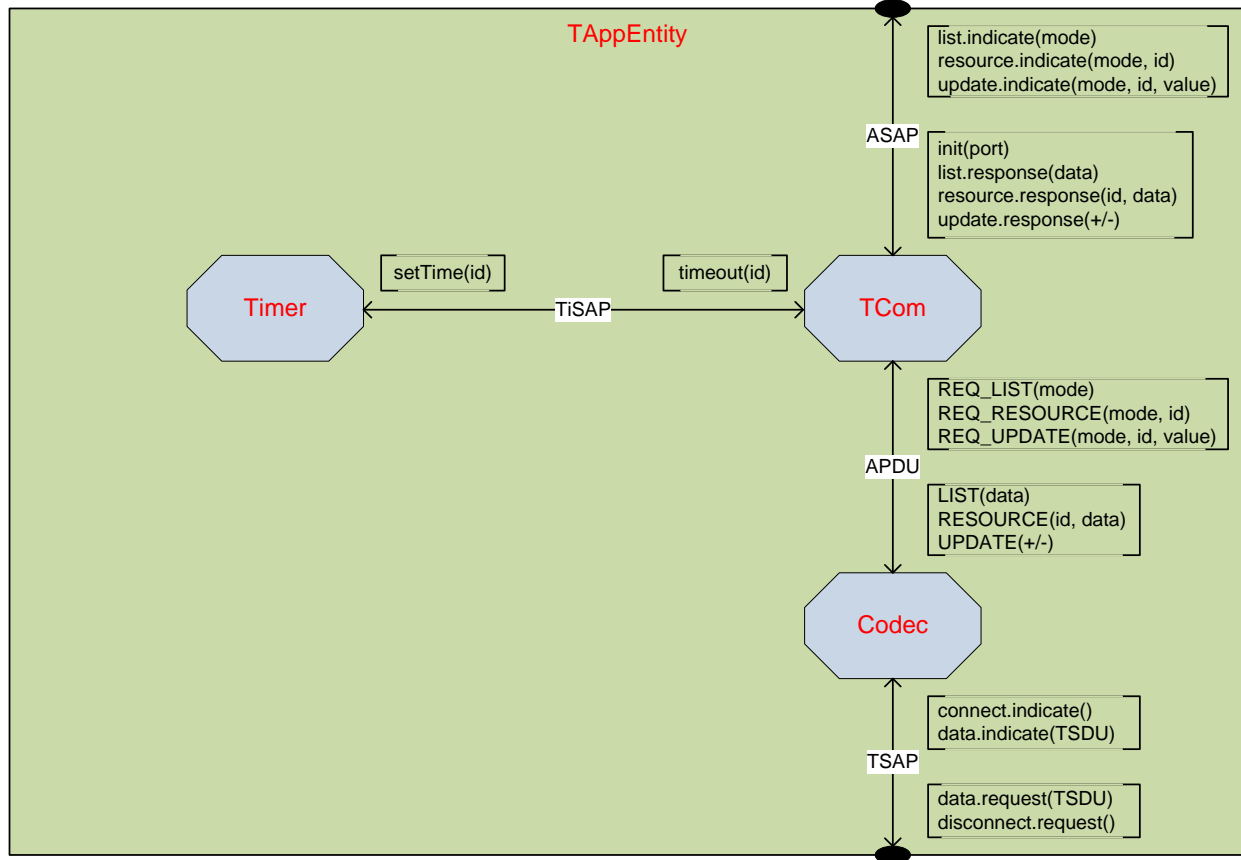


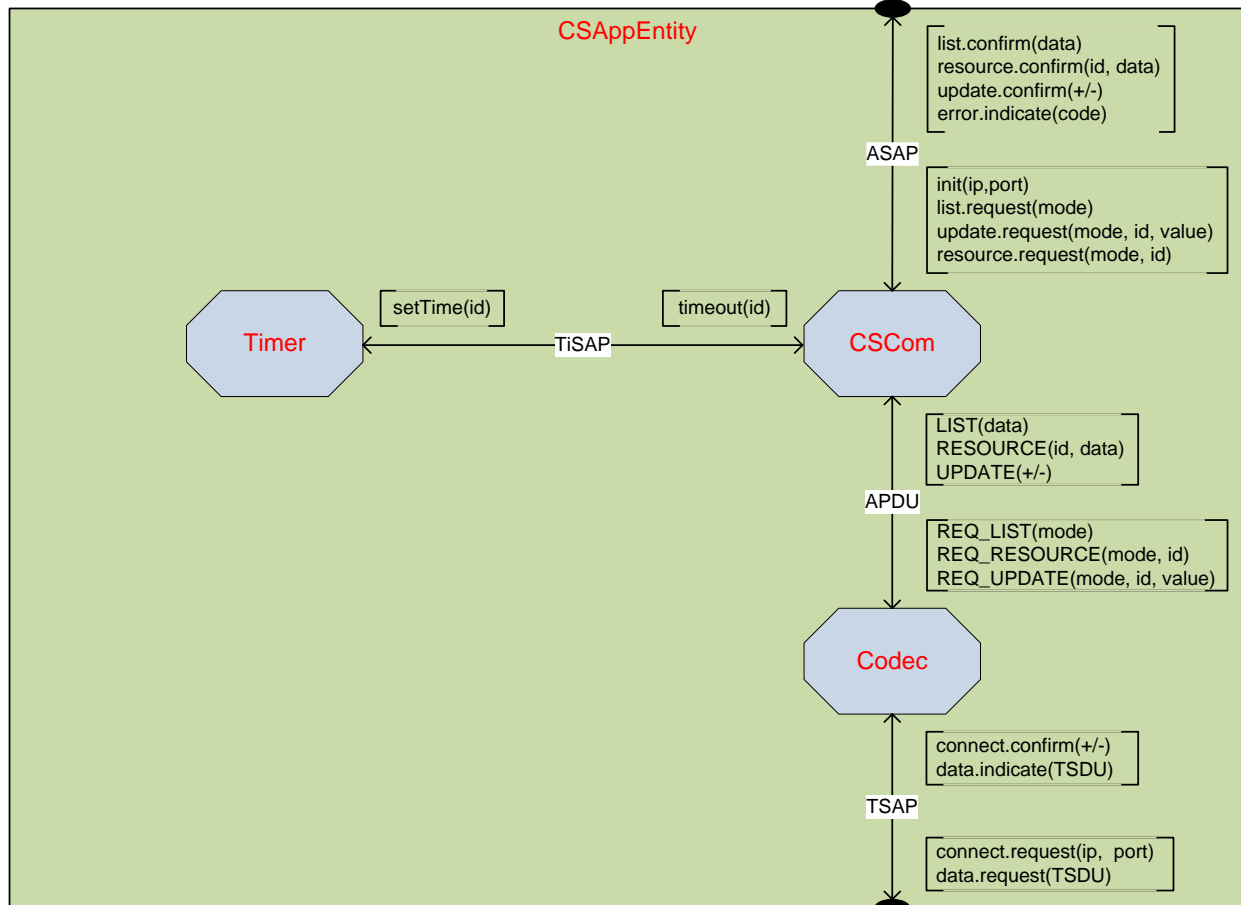
Figure 4-6: Block Specification of TAppEntity

The *Timer* process is used by the *TCom* for timing purposes. The *TCom* sets a timer by sending the signal *setTime(id)* to the *Timer*. When the time set elapses, the signal *timeout(id)* is sent back to the *TCom*. The *TCom* may set multiple timers. As such, the parameter *id* is used to uniquely identify every timer so that the *TCom* knows exactly which time expired. The *TCom* exchanges messages with the *Timer* through the Timer Service Access Point (*TiSAP*).

*TCom* stands for T400 Communication. This process is a Finite State Machine (FSM) and it implements the dynamic and timing behavior of the protocol, as described in the following subsection. It reacts to input service signals from the ASAP and APDU's from the *Codec*. It may also dispatch service signals to the ASAP or APDU's to the *Codec*.

*Codec* stands for code/decode. This process codes APDU's to the TSDU and decodes APDU's from the TSDU. It does this by adding the APCI information to the APDU when coding and removing it when decoding.

Now follows the description of the *CSAppEntity*. It has three processes as well namely; *Timer*, *CSCCom* and *Codec*. The following figure shows its internal structure;



**Figure 4-7: Block Specification of CSAppEntity**

The *Timer* and *Codec* processes here have the same meaning and functionality as their counterparts in the *TAppEntity*. *CSCom* stands for Centralized Server Communication. It plays the same role as the *TCom*.

The APDU's sent by the *CSCom* communicate service requests. They are coded using the HTTP request method GET. The name of the service requested is the resource name of the GET method and the parameters of the APDU are the HTTP request parameters. The table below shows examples of the coding of APDU's for service requests.

APDU	HTTP request
REQ_LIST(0)	GET /LIST?mode=0 HTTP/1.1
REQ_RESOURCE(0, 10)	GET /RESOURCE?mode=0&id=10 HTTP/1.1
UPDATE(1, 200, 50)	GET /UPDATE?mode=1&id=200&value=50 HTTP/1.1

**Table 4-2: HTTP Encoding of APDU's for Service Requests**

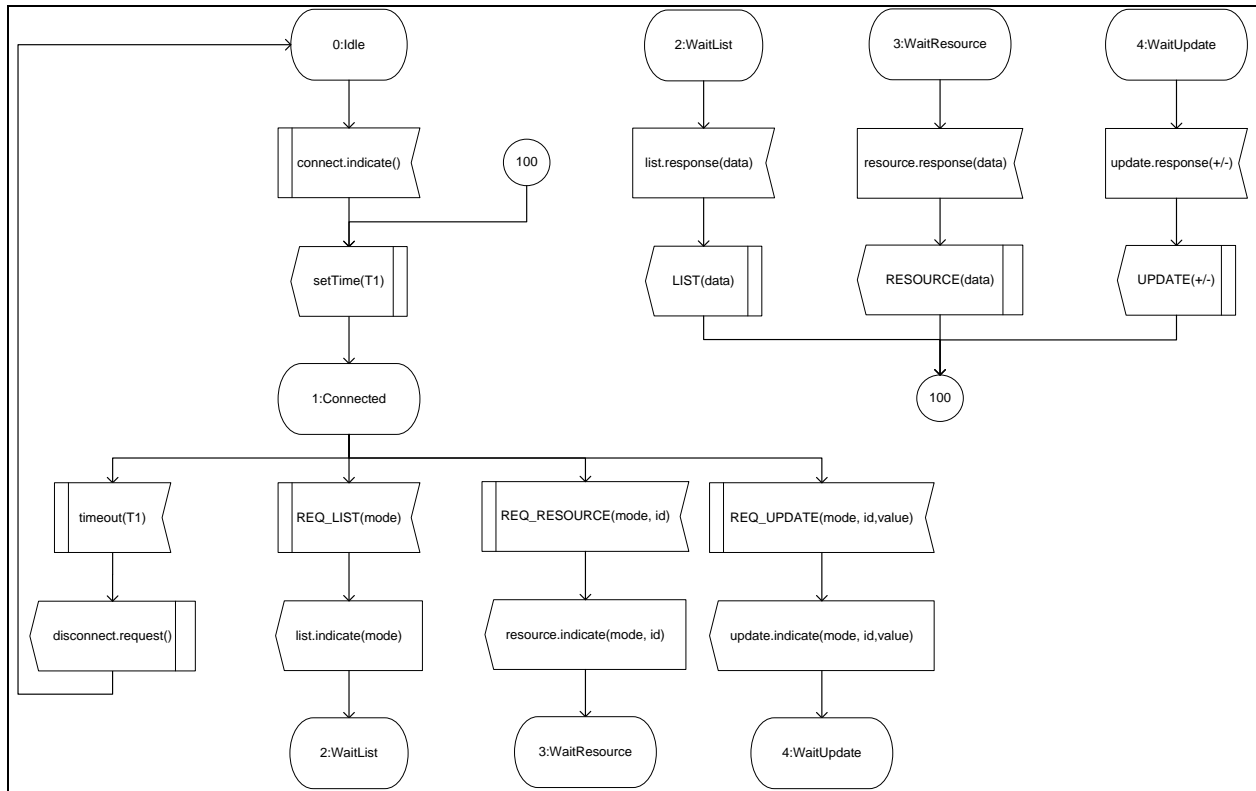
The APDU's sent by the TCom process communicate the service response to a service request. The service responses are encoded as well in HTTP. Here the data parameter of the service response, which is a CSV-file, is the content of the HTTP response. The following table shows examples of the coding of APDU's for service responses.

APDU	HTTP request
LIST(CSV-file)	HTTP/1.1 200 OK <CSV-file content>
RESOURCE(CSV-file)	HTTP/1.1 200 OK <CSV-file content>
UPDATE(+)	HTTP/1.1 200 OK
UPDATE(-)	HTTP/1.1 403 Forbidden

**Table 4-3: HTTP Encoding of APDU's for Service Responses**

### 4.3.2 *Dynamic Protocol Specification*

As already mentioned in the previous sub-section, the dynamic protocol specification specifies the state machine behavior of the communication processes *TCom* and *CSCom*. The timing behavior is specified as well. This specification is done using SDL state diagrams (see [1] & [2]). The dynamic behavior of the *TCom* process will be specified first and thereafter that of the *CSCom*. The following figure presents the state diagram of the *TCom* process.



**Figure 4-8: State Diagram of TCom Process**

The *TCom* state machine has five states namely; *Idle*, *Connected*, *WaitList*, *WaitResource* and *WaitUpdate*.

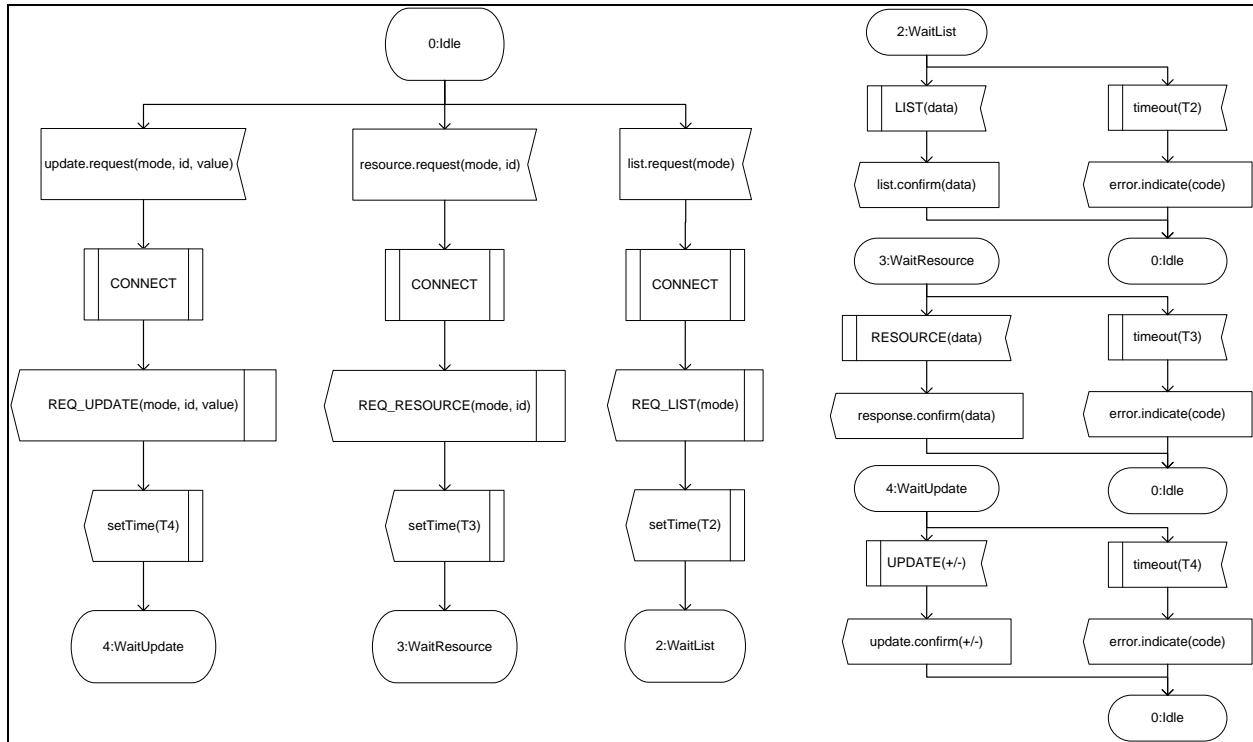
The *Idle* state is the initial state of the machine when it listens to the server port for a connection request from the centralized server. The transport layer notifies the machine of a connection request with the service signal *connect.indicate()*. On receipt of this signal, the machine sends the signal *setTime(T1)* to the *Timer* process and transits to the state *Connected*.

In the *Connected* state, there is an established TCP connection to the centralized server and the machine waits for APDU's from its peer. If the time *T1* runs out, the *Timer* process sends the signal *timeout(T1)* to the machine. The machine then sends the service signal *disconnect.request()* to the transport layer in order to terminate the connection and finally transits back to the *Idle* state. When the APDU's *REQ\_LIST(mode)*, *REQ\_RESOURCE(mode, id)* and *REQ\_UPDATE(mode, id, value)* are received, the machine notifies the T400 software with the service signals *list.indicate(mode)*, *resource.indicate(mode, id)* and *update.indicate(mode, id, value)* respectively. Finally, it transits to the states *WaitList*, *WaitResource* and *WaitUpdate* respectively.

The machine waits for the response of the T400 software in the states *WaitList*, *WaitResource* and *WaitUpdate*. When it receives the service signals *list.response(data)* in the state *WaitList*, *resource.response(data)* in the state *WaitResource* and *update.response(+/-)* in the state

*WaitUpdate*, it dispatches the APDU's *LIST(data)*, *RESOURCE(data)* and *UPDATE(+/-)* respectively to the *Codec* process. It then resets the timer *T1* and goes back to the state *Connected*.

The state diagram of the *CSCom* process is as follows;



**Figure 4-9: State Diagram of CSCom Process**

The *CSCom* process has four states; *Idle*, *WaitResource*, *WaitList* and *WaitUpdate*.

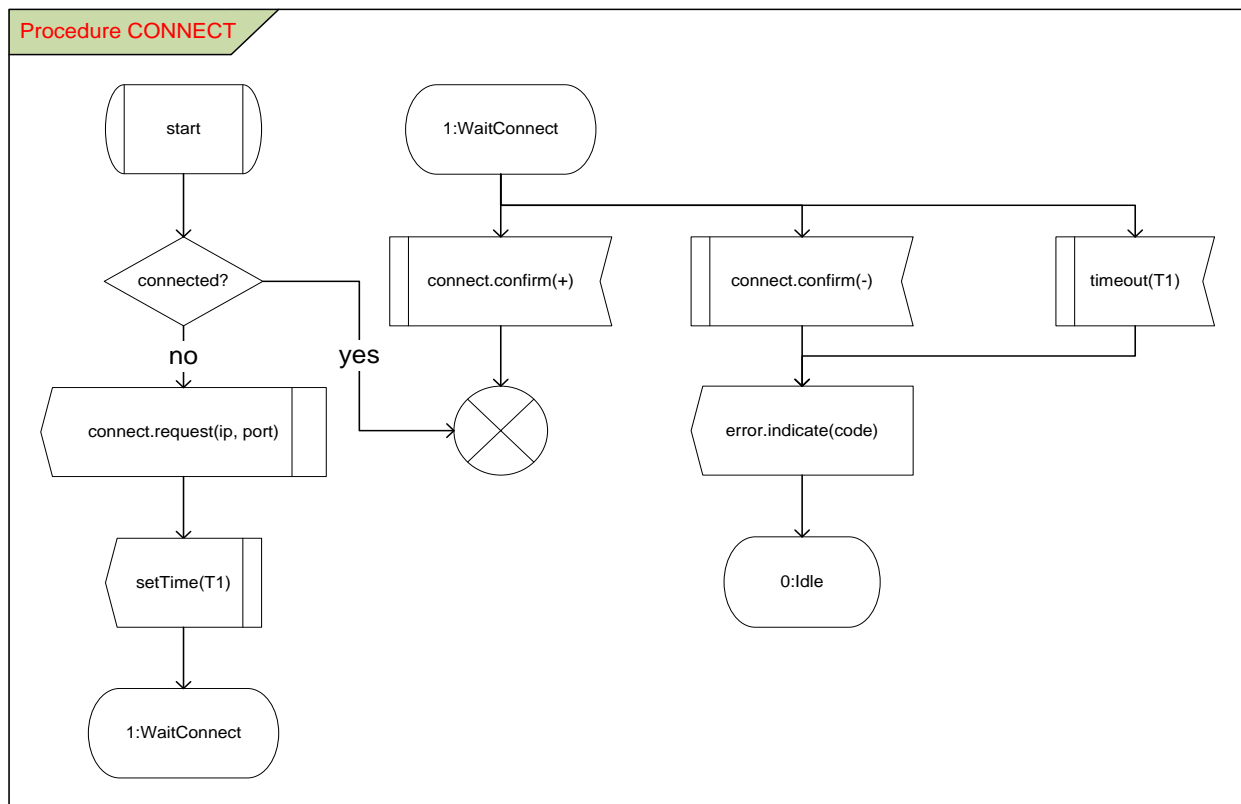
The *Idle* state is the initial state of the *CSCom* state machine. In this state it waits for requests from the centralized server software. If it receives the service signal *update.request(mode, id, value)*, it calls the procedure *CONNECT*. This procedure checks if the TCP connection to the T400 device is still open. If not, it reconnects to the T400 device. The machine then sends the APU *REQ\_UPDATE(mode, id, value)* to the *Codec* process, sets a timer with the signal *setTime(T4)* to the *Timer* process and transits to the state *WaitUpdate*. When the service signal *resource.request(mode, id)* is received in the *Idle* state, the procedure *CONNECT* is also called but the APDU *REQ\_RESOURCE(mode, id)* is sent to the *Codec*. The machine then sends the signal *setTime(T3)* to the *Timer* process and transits to the state *WaitResource*. If in the *Idle* state the service signal *list.request(mode)* is received, the machine still calls the procedure *CONNECT*. Then it sends the APDU *REQ\_LIST(mode)* to the *Codec* and finally the signal *setTime(T2)* to the *Timer*. Thereafter, it makes a transition to the state *WaitList*.

In the state *WaitList*, the *CSCom* state machine waits for the list response from the T400 device. If the time  $T2$  runs out, the *Timer* process sends the signal *timeout(T2)* to the machine. The machine at this state sends the service signal *error.indicate(code)* to the centralized server software and transits to the *Idle* state. If the APDU *LIST(data)* is received in time, the machine notifies the server software with the service signal *list.indicate(data)* and still transits to the *Idle* state.

The *CSCom* state machine, on the other hand, waits for the resource response from the T400 device when in the state *WaitResource*. It notifies the centralized server software with the service signal *resource.indicate(data)* when it receives the APDU *RESOURCE(data)* in time. If not, on receipt of the signal *timeout(T3)*, it sends the service signal *error.indicate(code)* to the centralized server software. In both cases, the machine transits to the *Idle* state.

The *CSCom* state machine in the state *WaitUpdate* waits for the update response from the T400 device. It notifies the centralized server software with the service signal *update.indicate(+/-)* when it receives the APDU *UPDATE(+/-)* in time. If not, on receipt of the signal *timeout(T4)*, it sends the service signal *error.indicate(code)* to the centralized server software. In both cases, the machine transits to the *Idle* state.

Now follows the state diagram of the procedure *CONNECT*;



**Figure 4-10: Procedure CONNECT of CSCom**

The *CONNECT* procedure starts by checking if the connection to the T400 device is still open. If it is, the function simply terminates. If not, it has to reconnect. It does this by sending the service signal *connect.request(ip,port)* to the transport layer. It then sets the timer *T1* by sending the signal *setTime(T1)* to the *Timer* process and transits to the state *WaitConnect*, where it waits for confirmation from the transport layer.

The transport layer confirms a connection request with the service signals *connect.confirm(+)* for a successful connection and *connect.confirm(-)* if the connection failed. If the *CSCom* in the state *WaitConnect* receives a positive confirmation, the procedure terminates normally. On the other hand, if the confirmation is negative or it receives the signal *timeout(T1)* from the *Timer* process, it sends the service signal *error.indicate(code)* to the server software and transits to the *Idle* state.



## 5 System Implementation

Due to time constraints, only the MySQL database and the centralized server of the system specification diagram (see figure 4-1 of the previous chapter) were implemented. More so, SunSil does not allow, for the purpose of secrecy, the publication of any software running on the T400 device. As such, the T400 device was simulated, in order to test the communication between it and the centralized server.

This chapter starts by discussing the development environment of the T400 device, the centralized server and the MySQL database. It also mentions the tools used for implementation. Thereafter, it discusses in detail the implementation of the MySQL database and the centralized server.

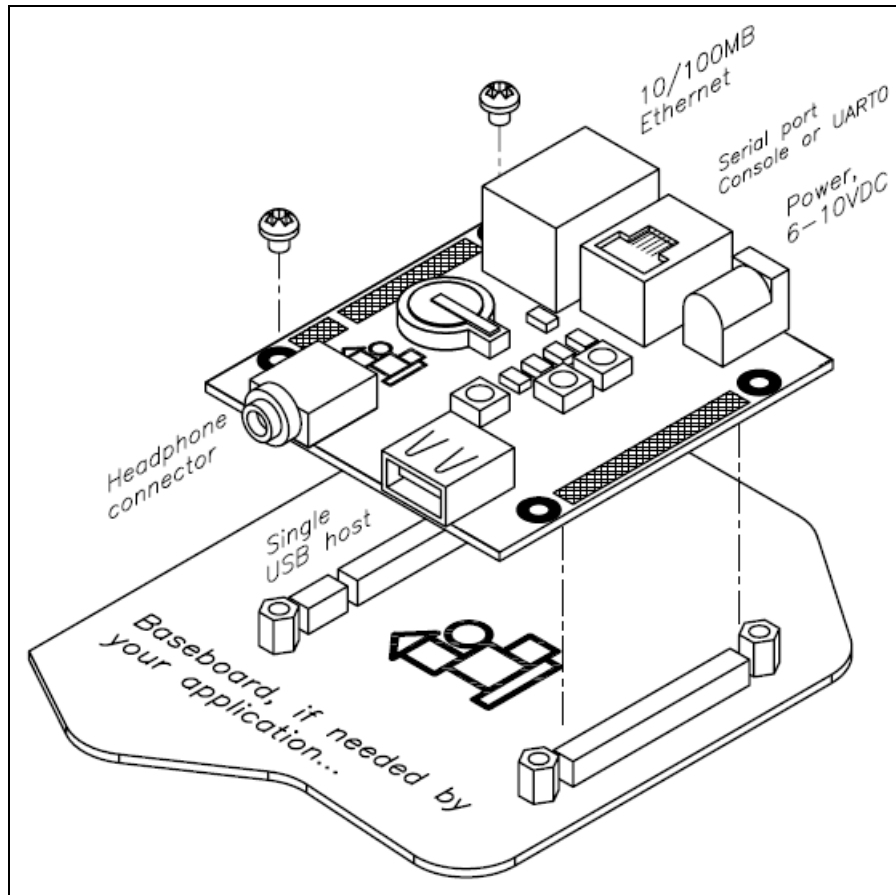
### 5.1 Development Environment

The T400 software and its communication protocol were implemented on a nanoLIAB board while the MySQL database and the centralized server were implemented on a Windows XP PC. The following two subsections give a brief description of these development environments.

#### 5.1.1 *NanoLIAB Board*

LIAB stands for Linux In A Box. This board is based on the ARM9 processor AT91RM9200, which is well suited for the Linux operating system. The ARM was chosen because of its low power consumption and low cost. The version of the Linux kernel running on the board used is the 2.6.16 Linux kernel.

The board has three push buttons, one of which is a reset, and five LED's. It also has an Ethernet interface, a serial port, a USB host and a headphone connector. There is an optional base board which can be connected to the processor board for additional I/O-devices. Below is a diagram showing the top view of the board;



**Figure 5-1: NanoLIAB Board**

For details of the schematics of this board, visit the website at [3]. As already mentioned, the implementation of the communication protocol on this board will not be discussed. Only the simulator used for testing the protocol is discussed in the next chapter.

### **5.1.2 Windows XP PC**

A Windows XP PC was used to implement the MySQL database and the centralized server. The Professional version of Windows XP was used. The PC was manufactured by Dell Inc. and has an Intel Core Duo CPU, each of 2.00 GHz, and a RAM size of 2 GB.

## **5.2 MySQL Database**

In this section, the *CREATE TABLE* statements used to create the tables for the database model of chapter 4 are discussed. For information about downloading, installing and configuring a MySQL database, visit its website at [4]. There is a well-documented step-by-step procedure for getting started with MySQL at the site. The reference manual for MySQL available at this site was also extensively used for this project.

A brief introduction is given into DbVisualizer in Appendix A. This is a tool used in this project to implement the database. It is a very good and simple tool to use for databases. Now follows the discussion of the table implementations of the database.

### 5.2.1 User Table

```
CREATE TABLE User (  
  username CHAR(8) NOT NULL PRIMARY KEY,  
  last_name VARCHAR(32),  
  first_name VARCHAR(32),  
  password CHAR(64) NOT NULL,  
  email VARCHAR(64) NOT NULL,  
  rights ENUM('DeviceUser', 'GroupUser', 'SuperUser') NOT NULL DEFAULT 'DeviceUser',  
  lang VARCHAR(8) NOT NULL DEFAULT 'en',  
  last_login DATETIME,  
  uentry_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Listing 5-1: Definition of User Table

The column *username* is primary key. The columns *last\_name*, *first\_name* and *last\_login* are optional but the rest must be given. The column *rights* is an enumeration which defaults to *DeviceUser* if not specified. The column *lang* defaults to *en* for English. The column *uentry\_date* is a constant timestamp which defaults to the current timestamp if not specified. When inserting a new user to the table, the columns *username*, *password* and *email* must be specified. The rest then take their default values or are set to *NULL*.

### 5.2.2 Groups Table

```
CREATE TABLE Groups (  
  gid INT(8) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  gname VARCHAR(32) NOT NULL UNIQUE,  
  location VARCHAR(256)  
);
```

Listing 5-2: Definition of Groups Table

The column *gid* is primary key and is an automatically incremented unsigned integer. It numbers the groups inserted into the table from one upwards. It is not allowed to be specified when inserting a new group to the table. The column *gname*, on the other hand, must be unique and must be specified during insertion while *location* is optional. It is set to *NULL* if not specified.

### 5.2.3 UserGroup Table

```
CREATE TABLE UserGroup (  
  username CHAR(8) NOT NULL REFERENCES User(username) ON DELETE CASCADE,  
  gid INT(8) UNSIGNED ZEROFILL NOT NULL REFERENCES Groups(gid) ON DELETE CASCADE,  
  PRIMARY KEY(username, gid)  
);
```

Listing 5-3: Definition of UserGroup Table

The column *username* of this table references the same column in the *User* table and the column *gid* also references the same column in the *Groups* table. If a user or a group is deleted from their tables, the corresponding entries in this table are automatically deleted. The columns of this table act jointly as primary key and they must both be specified when inserting a row to this table.

### 5.2.4 Device Table

```
CREATE TABLE Device (  
  did INT(16) UNSIGNED ZEROFILL NOT NULL AUTO_INCREMENT PRIMARY KEY,  
  dname VARCHAR(32) NOT NULL UNIQUE,  
  gid INT(8) UNSIGNED ZEROFILL REFERENCES Groups(gid) ON DELETE SET NULL,  
  location VARCHAR(256),  
  ip VARCHAR(16) NOT NULL UNIQUE,  
  port INT(8) NOT NULL DEFAULT 8192,  
  type ENUM('UserDevice', 'PlantDevice') NOT NULL DEFAULT 'UserDevice',  
  owner CHAR(8) REFERENCES User(username) ON DELETE SET NULL,  
  last_connect DATETIME,  
  dentry_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP  
);
```

Listing 5-4: Definition of Device Table

The column *did* is primary key and is an automatically incremented unsigned integer. It counts the devices inserted into this table from one upwards. It is not allowed to be specified during insertion. The column *dname* is unique and must be specified when inserting a device. The column *gid* references the same column of the *Groups* table. It is set to *NULL* when its corresponding group is deleted from the *Groups* table. The column *ip* is as well unique and must be specified during insertion meanwhile the column *port* may not be specified, in which case it is set to its default value 8192. The column *type* is an enumeration which defaults to the value *UserDevice* if not specified during insertion. The column *owner* references the column *username* of the *User* table. When the corresponding user is deleted from the *User* table, its value is automatically set to *NULL* by the database. The column *last\_connect* may or may not be

specified during insertion. The column *dentry\_date* is a constant timestamp which is set by default to the current time if not specified when inserting a device.

### 5.2.5 Users View

```
CREATE VIEW users AS SELECT
    username AS user_name,
    password AS user_pass
FROM User;
```

Listing 5-5: Definition of users View

The column *user\_name* of this view represents the column *username* of the *User* table while the column *user\_pass* represents the *password* column of the *User* table.

### 5.2.6 User\_roles Table

```
CREATE TABLE user_roles (
    user_name CHAR(8) NOT NULL REFERENCES users(user_name) ON DELETE CASCADE,
    role_name VARCHAR(8) NOT NULL,
    PRIMARY KEY(user_name, role_name)
);
```

Listing 5-6: Definition of user\_roles Table

The column *user\_name* references the column *user\_name* of the *users* view which in turn represents the column *username* of the *User* table. On deleting a user from the *User* table, the corresponding entry in this table is deleted as well by the database. Both columns of this table are primary key and must be specified when inserting a row to this table.

## 5.3 Centralized Server

The centralized server as shown in the system specification of figure 4-1 is an Apache Tomcat web server with a servlet container. Servlets are implemented in Java. Details on downloading, installing and configuring a tomcat server are well documented on its website at [6] and will not be discussed here. However, some vital configurations of the server will be mentioned, in the course of the discussions here, to explain certain behaviours of the server or why a functionality is implemented in a particular way. The documentation at the tomcat website at [6] was also extensively used throughout this project.

The centralized server constitutes the centralized server software and the centralized server application entity (CSAppEntity) as shown in the service specification of figure 4-3. This section

discusses the software architecture and packages of the centralized server. The architecture used here is generic and can be used for any kind of server application. The actual web pages generated by the centralized server and how they are interlinked are discussed in the next chapter. The actions carried out by the user to perform the tasks specified in the requirements analysis are as well discussed in the next chapter.

Before discussing the implementation of the centralized server application, it is worth mentioning that the important configuration files for this application, which are needed by the Apache Tomcat, are shown in Appendix B. Also, the Apache Ant is a tool used to build web applications for the Apache Tomcat server and automatically install them on the server. This tool was used extensively in this project. More about this tool can be found in [7].

### 5.3.1 Software Architecture

The centralized server has a simple three-tier architecture as described in [8]. The tiers consist of the Presentation Tier, the Logic Tier and the Data Tier. Below is a diagram which illustrates the tiers, their software components and how the components interact with each other.

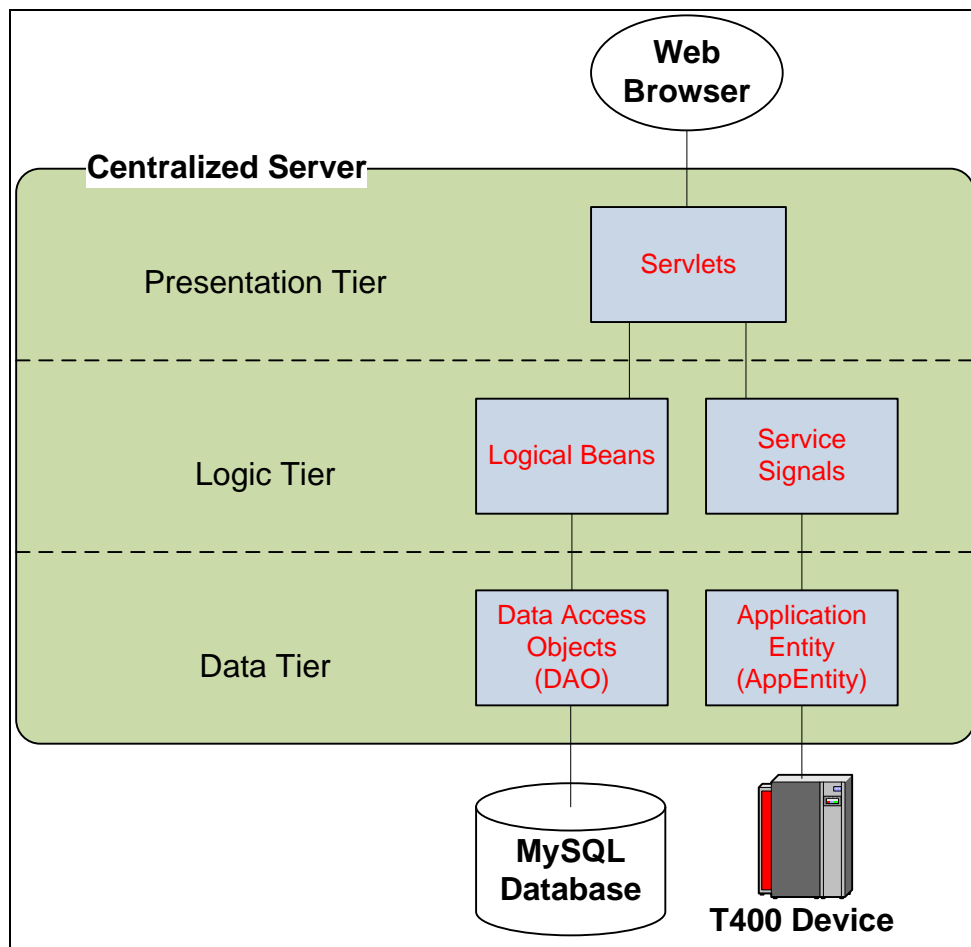


Figure 5-2: Software Architecture of Centralized Server

The *Presentation Tier* is in charge of the user view, i.e. what the browser displays to the user. It is made up of the servlets component. Servlets are exclusively used at the backend of the web interface while HTML and JavaScript are used at the frontend. The servlets dynamically generate HTML code on request from the web browser.

The *Logical Tier* acts as an interface between the presentation and data tier. It abstracts the raw data representation in the data tier from the presentation tier. This makes it possible to change the MySQL database, for example, to an LDAP server without changing or recompiling the code of the presentation and logical tiers. All that needs to be done is re-implement the Data Access Object (DAO) while keeping its interface constant. The DAO is loaded at runtime. The logical tier consists of two software components; logical beans and service signals. The logical beans are objects with GET- and SET-functions for all their attributes. They describe the tables in the database and their attributes correspond to table columns. They also enforce user rights. The service signals are as described in table 4-1.

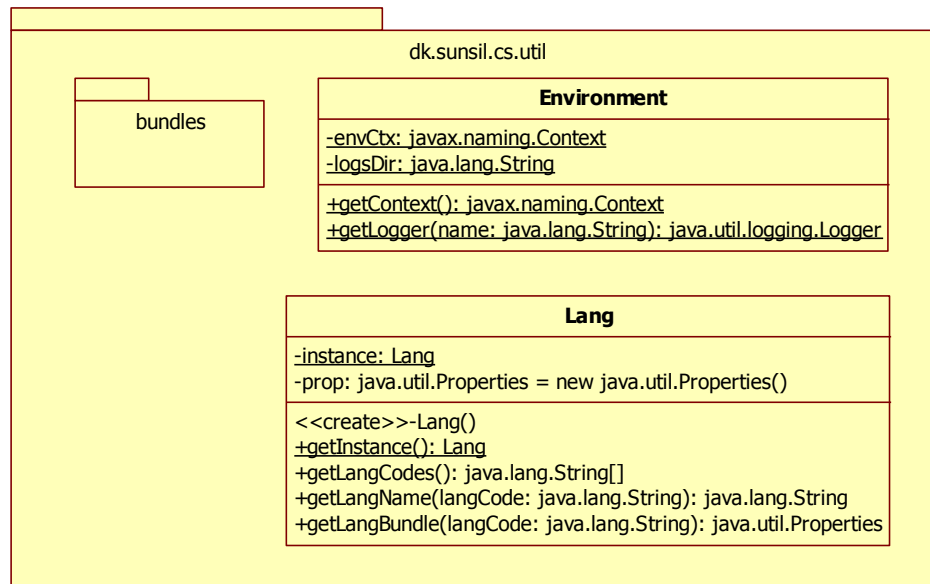
The *Data Tier* is in charge of saving and retrieving data from external actors - the MySQL database and T400 device in this case. It converts the raw data representation to some logical form used by the presentation tier. This gives rise to the advantage that the application logic is decoupled from the underlying data storage. So, changing the form of data storage will require only this tier to be re-implemented. The *Data Tier* is made up of two software components; the Data Access Object (DAO) and the Application Entity (AppEntity). The DAO saves and retrieves data from the MySQL database while the AppEntity communicates with the T400 devices.

The software components of the different tiers shown in figure 5-2 were all implemented as Java packages. An additional software component – utilities, not shown in figure 5-2, implements general utility functionalities like logging, language handling, etc. Below is a table summarizing all software components and their corresponding Java packages;

<b>Software Component</b>	<b>Java Package</b>
Utilities	dk.sunsil.cs.util
Logical Beans	dk.sunsil.cs.bean
Data Access Objects (DAO)	dk.sunsil.cs.dao
Service Signals	dk.sunsil.cs.signal
Application Entity (AppEntity)	dk.sunsil.cs.appentity
Servlets	dk.sunsil.cs.servlet

**Table 5-1: Software Components of Centralized Server**

### 5.3.2 Package *dk.sunsil.cs.util*



**Figure 5-3: Class Diagram of Package *dk.sunsil.cs.util***

The class *Environment* is static and it provides the environment context and handles logging. The private static attribute *envCtx* is a reference to the single environment context object provided by the Apache Tomcat server. It is initialized by a static block in the class *Environment* as follows;

```
Context initCtx = new InitialContext();
envCtx = (Context) initCtx.lookup("java:comp/env");
```

**Listing 5-7: Lookup of Environment Context**

The two lines of code above is a standard way of looking up the environment context of a web application in the Apache Tomcat server. This environment context is initialized by the Apache Tomcat server using the *context.xml* file located in the *META-INF* directory of the web application (see Appendix B). The public static function *getContext()* simple returns the attribute *envCtx*.

The private static attribute *logsDir* specifies the path to the directory where the log files are saved. It is initialized in the same static block as the attribute *envCtx*. The public static function *getLogger(name)* creates and initializes a logger with the specified name on the first call. Subsequent calls with the same name simply return the logger. The loggers are initialized, using the attribute *logsDir*, such that all information written to this logger is saved in the file with the following path;



`<tomcat-home>/logs/sunsi/<name>.yyyy-mm-dd.log`

where

*tomcat-home* : home directory of the Apache Tomcat server, e.g. C:\Programs\Tomcat.  
*name* : parameter passed to the function, eg. dk.sunsi.cs.servlet.HomeServlet.  
*yyyy-mm-dd* : year, month and day when the logging was done, e.g. 2009-08-01.

The class *Lang* has a singleton pattern as described in [8]. The static private attribute *instance* is a reference to the single instance of this class and the private attribute *prop* stores the properties of this class in a hash table. The class has a single default constructor *Lang()* which is private. This means only this class can create instances of itself. This is done by the public static function *getInstance()* which creates the single instance of this class on its first call and returns that instance on subsequent calls. The function *getLangCodes()* returns in a string array the codes of the available languages in the application while the function *getLangName(langCode)* returns the full name of a given language code. The table below shows an example of three language codes and their full names;

Language Code	Language Name
en	English
de	Deutsch
dk	Dansk

**Table 5-2: Language Codes and Names**

The function *getLangBundle(langCode)* returns a list of textual translations into the language with the specified code.

The sub-package *bundles* contains properties-files used to initialize the *Lang* instance. It contains the files *lang.properties*, *lang\_en.properties* and *lang\_de.properties*. The file *lang.properties* defines the codes of all available languages and their full names. The language codes are the keys while the values are the language names. The files *lang\_en.properties* and *lang\_de.properties* define textual translations in English and German respectively. Below is a listing of the file *lang.properties*;

```
en = English
de = Deutsch
```

**Listing 5-8: File lang.properties**

So, to add a new language to the application, add the key-value pair `<code> = <name>` for the language in the *lang.properties* file as shown above and create the file *lang\_<code>.properties*,

which contain all required textual translations in that language. Then reload the application and it will automatically display the language for selection.

### 5.3.3 Package *dk.sunsil.cs.bean*

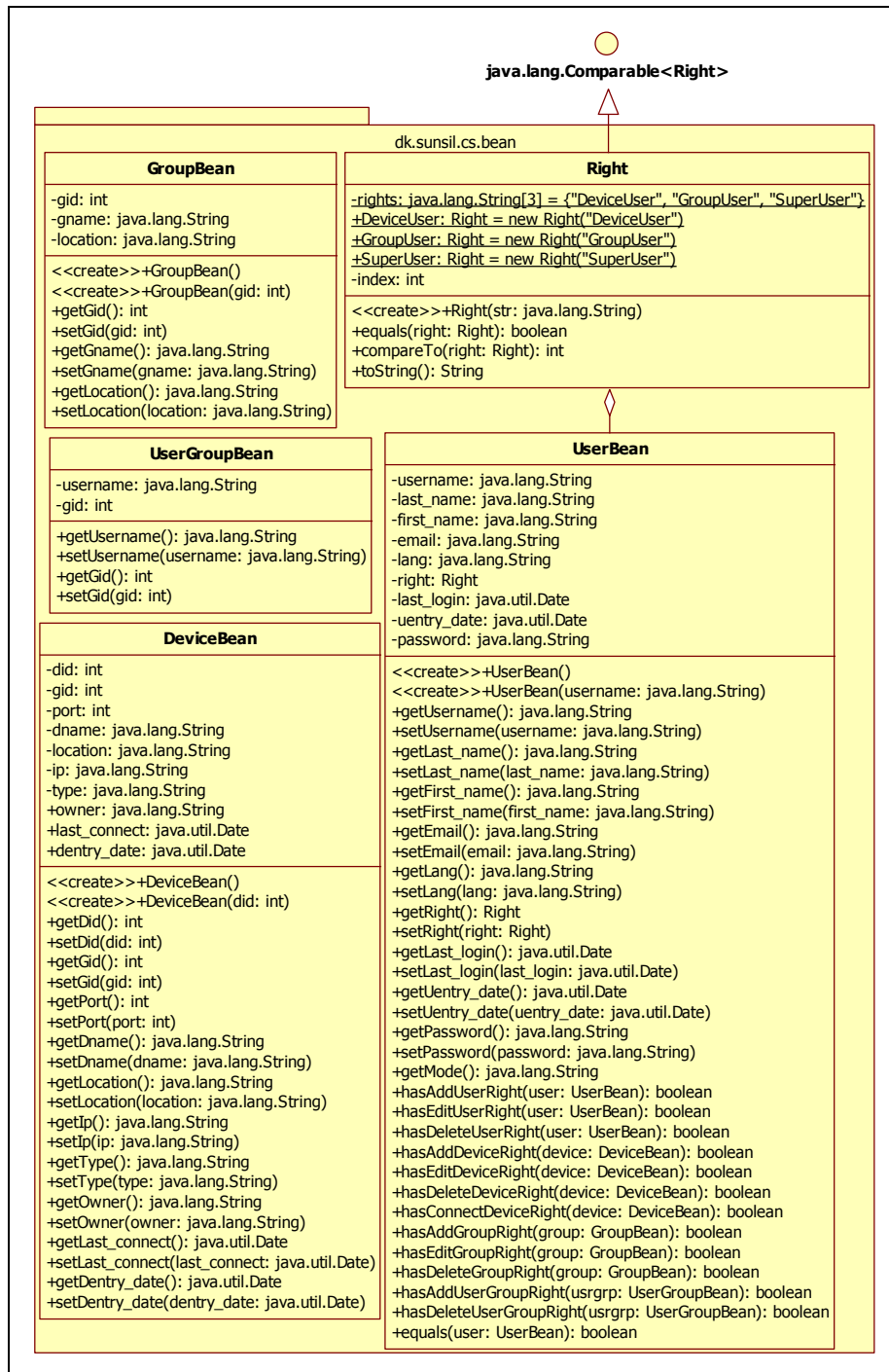
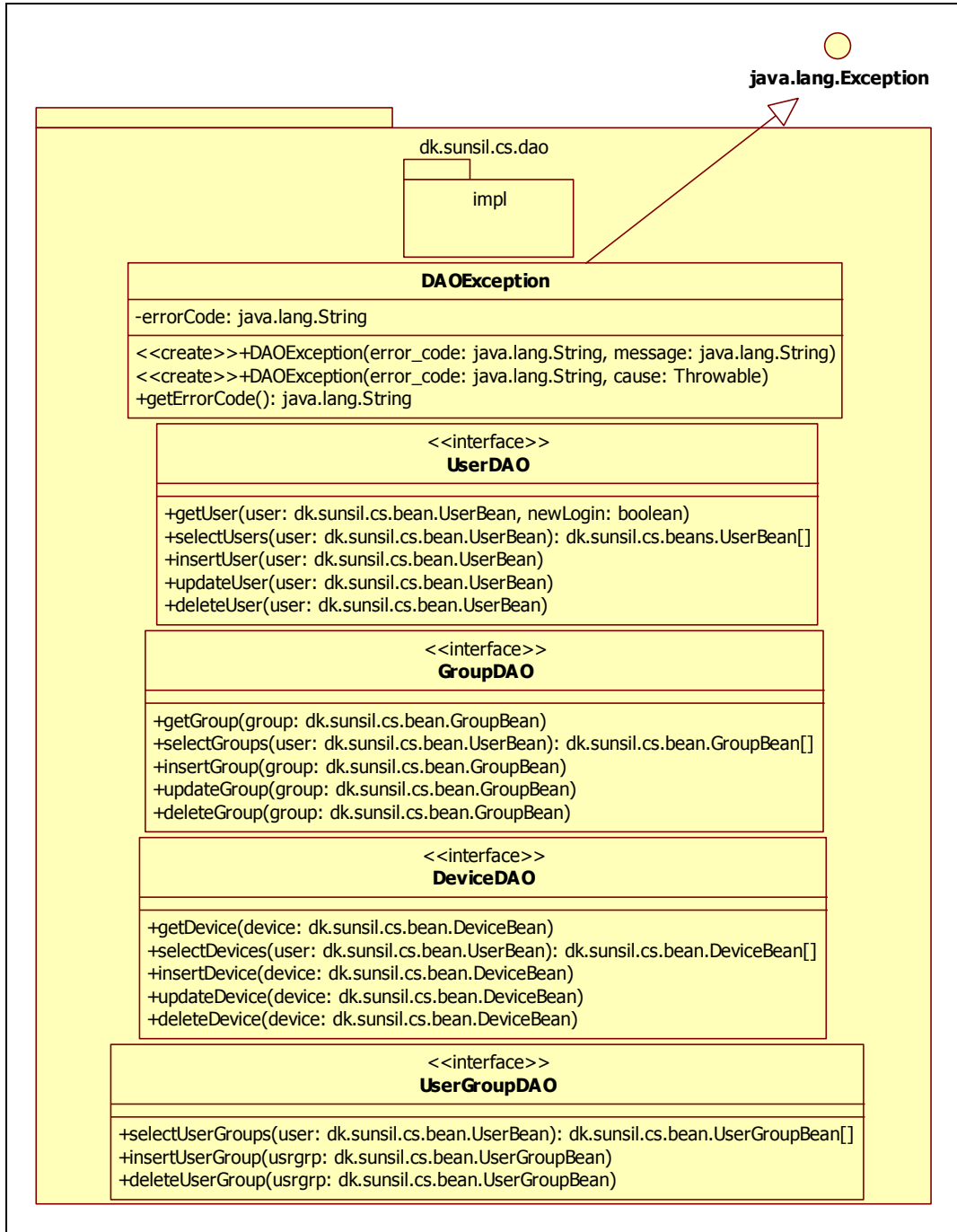


Figure 5-4: Class Diagram of Package *dk.sunsil.cs.bean*

The class *Right* describes the user rights as discussed in section 4.1.1 of the system specification. It implements the interface *java.lang.Comparable<Right>*. As such, it has to override the function *compareTo(right)* of this interface. This function compares the current *Right* object with that passed as parameter and returns an integer. It returns zero if both objects are equal else it returns a negative number if the current object is less than the parameter object and a positive number otherwise. The private static attribute *rights* is constant and used in the constructor to initialize the private attribute *index* and in the *toString()* function to get the string value of an instance. The public static attributes *DeviceUser*, *GroupUser* and *SuperUser* are constant and can be used as reference objects for comparison. The function *equals(right)* compares two *Right* objects and returns true if they are equal and false otherwise.

The classes *UserBean*, *GroupBean*, *UserGroupBean* and *DeviceBean* describe a user, group, usergroup and device respectively. All their attributes have GET- and SET-functions. The HAS-RIGHT functions of the class *UserBean* are used to check if a user has the right to carry out the given operation on the object passed as parameter. These functions are called by the servlets before a user can carry out any of the given operations on the specified object. The *getMode()* function of the class *UserBean* gets the device access mode of the user. The mode 0 specifies the user mode and is returned for device users while the mode 1 specifies the install mode which is returned for group and super users. The function *equals(user)* of the class *UserBean* returns true if the *username* of both *UserBean* instances compared are equal else it returns false.

### 5.3.4 Package *dk.sunsil.cs.dao*



**Figure 5-5: Class Diagram of Package *dk.sunsil.cs.dao***

The interface *DeviceDAO* defines the interface to the Data Access Object (DAO) used to access the devices in the database. The functions *getDevice(device)*, *insertDevice(device)*,

*updateDevice(device)* and *deleteDevice(device)* respectively fetch, add, edit and remove the specified device. The function *selectDevices(user)* fetches only those devices which the specified user is allowed to manipulate.

The interface *UserDAO* defines the interface to DAO used to access the users in the database. The functions *getUser(user, newLogin)*, *insertUser(user)*, *updateUser(user)* and *deleteUser(user)* respectively fetch, add, edit and remove the specified user. The parameter *newLogin*, if true, tells the DAO to set the last login of the specified user to the current time. This is done by the servlets if the user has a new session. The function *selectUsers(user)* fetches only those users which the specified user is allowed to manipulate.

The interface *GroupDAO* defines the interface to the DAO used to access the groups in the database. The functions *getGroup(group)*, *insertGroup(group)*, *updateGroup(group)* and *deleteGroup(group)* respectively fetch, add, edit and remove the specified group. The function *selectGroups(user)* fetches only those groups which the specified user is allowed to manipulate.

The interface *UserGroupDAO* defines the interface to the DAO used to access the usergroups in the database. The functions *insertUserGroup(usrgrp)* and *deleteUserGroup(usrgrp)* respectively add and remove the specified usergroup. The function *selectUserGroups(user)* fetches only those usergroups which the specified user is allowed to manipulate.

The class *DAOException* inherits from the class *java.lang.Exception*. Its single private attribute *errorCode* is set with the parameter *error\_code* of the constructors. The parameters *message* and *cause* of the constructors are passed to the base constructor. The only public method of this class is the GET-function of the *errorCode*. All functions of the interfaces in this package throw a *DAOException* if an error occurs during execution.

The sub-package *impl* is discussed in the following sub-section. It contains the implementations of the DAO interfaces discussed above.

### 5.3.5 Package *dk.sunsil.cs.dao.impl*

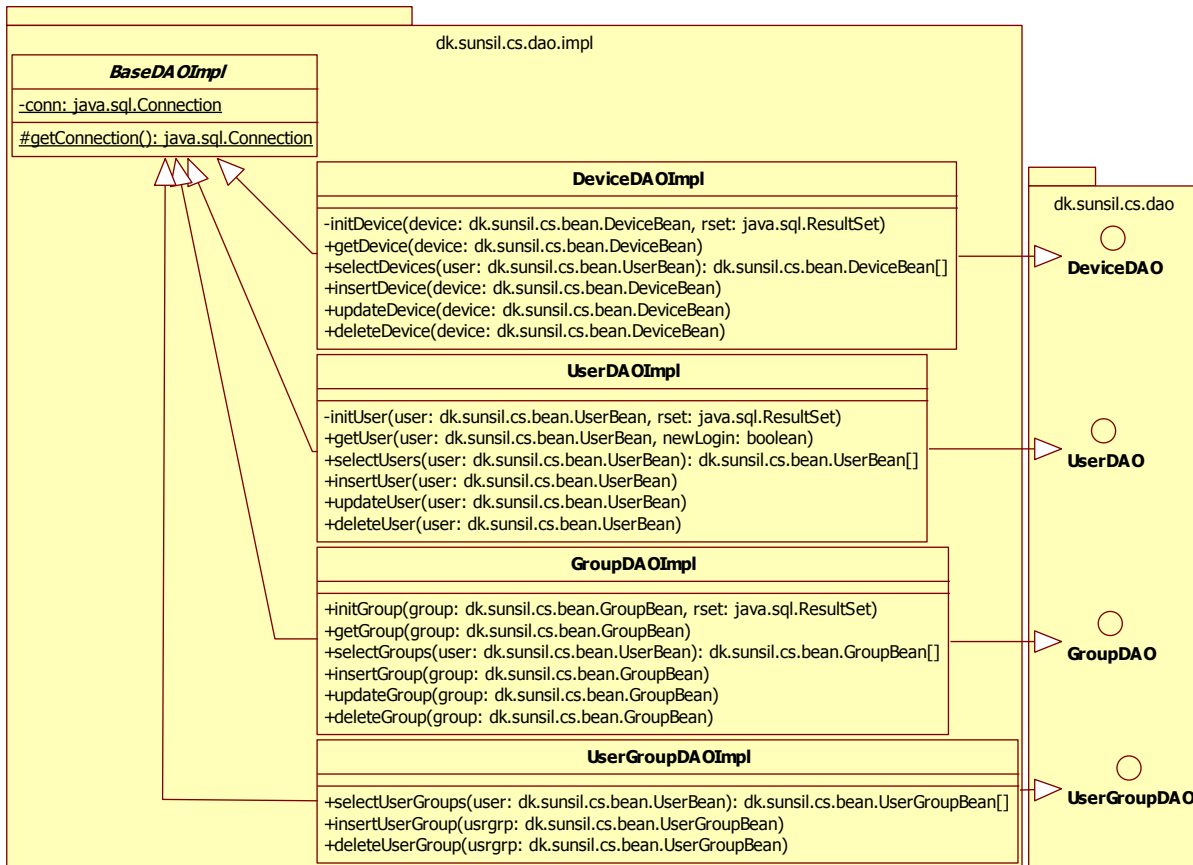


Figure 5-6: Class Diagram of Package *dk.sunsil.cs.dao.impl*

The abstract class *BaseDAOImpl* provides a single SQL connection to the DB for all DAO implementations. It is more efficient to recycle and reuse already existing connections than to open new ones according to the documentation at [6]. This uses up less resources and prevents DB connection pool leaks, where unclosed connections become unavailable for reuse. The protected static function *getConnection()* creates a new connection at its first call or if the existing connection has been closed and reuses this connection for subsequent calls. The connection is created from the JDBC data source defined in the context configuration file of the web application (see Appendix B).

The above resource can be accessed through the environment context (see listing 5-7) using its name. The Java code used to access the above resource and create a connection is as follows;

```
// Look up data source
DataSource ds = (DataSource)Environment.getContext().lookup("jdbc/SunSilDB");
// Allocate a connection from the pool
conn = ds.getConnection();
```

#### Listing 5-9: Creating a Connection from a Data Source

The class *DeviceDAOImpl* inherits from the class *BaseDAOImpl*, to make use of the connection it provides, and implements the interface *dk.sunsil.cs.dao.DeviceDAO*. It therefore implements all functions of this interface. The additional function *initDevice(device, rset)* initializes the specified device with values from a row of the given result set, which is a representation of a database table.

The class *UserDAOImpl* inherits from the class *BaseDAOImpl*, to make use of the connection it provides, and implements the interface *dk.sunsil.cs.dao.UserDAO*. It therefore implements all functions of this interface. The additional function *initUser(user, rset)* initializes the specified user with values from a row of the given result set, which is a representation of a database table.

The class *GroupDAOImpl* inherits from the class *BaseDAOImpl*, to make use of the connection it provides, and implements the interface *dk.sunsil.cs.dao.GroupDAO*. It therefore implements all functions of this interface. The additional function *initGroup(group, rset)* initializes the specified group with values from a row of the given result set, which is a representation of a database table.

The class *UserGroupDAOImpl* inherits from the class *BaseDAOImpl*, to make use of the connection it provides, and implements the interface *dk.sunsil.cs.dao.UserGroupDAO*. It therefore implements all functions of this interface.

All DAO implementation classes are as well defined in the context configuration file of the web application (see Appendix B). The DAO implementations are loaded at runtime and casted to their corresponding interfaces. This makes it possible to re-implement them without recompiling the application by specifying their new class paths for the *type* attribute. Below is a Java code listing for loading and casting the DAO implementations;

```
Context envCtx = Environment.getContext();
DeviceDAO device_dao = (DeviceDAO) envCtx.lookup("dao/device");
GroupDAO group_dao = (GroupDAO) envCtx.lookup("dao/group");
UserGroupDAO usrgrp_dao = (UserGroupDAO) envCtx.lookup("dao/usergroup");
UserDAO user_dao = (UserDAO) envCtx.lookup("dao/user");
```

#### Listing 5-10: Loading DAO Implementations

### 5.3.6 Package *dk.sunsil.cs.signal*

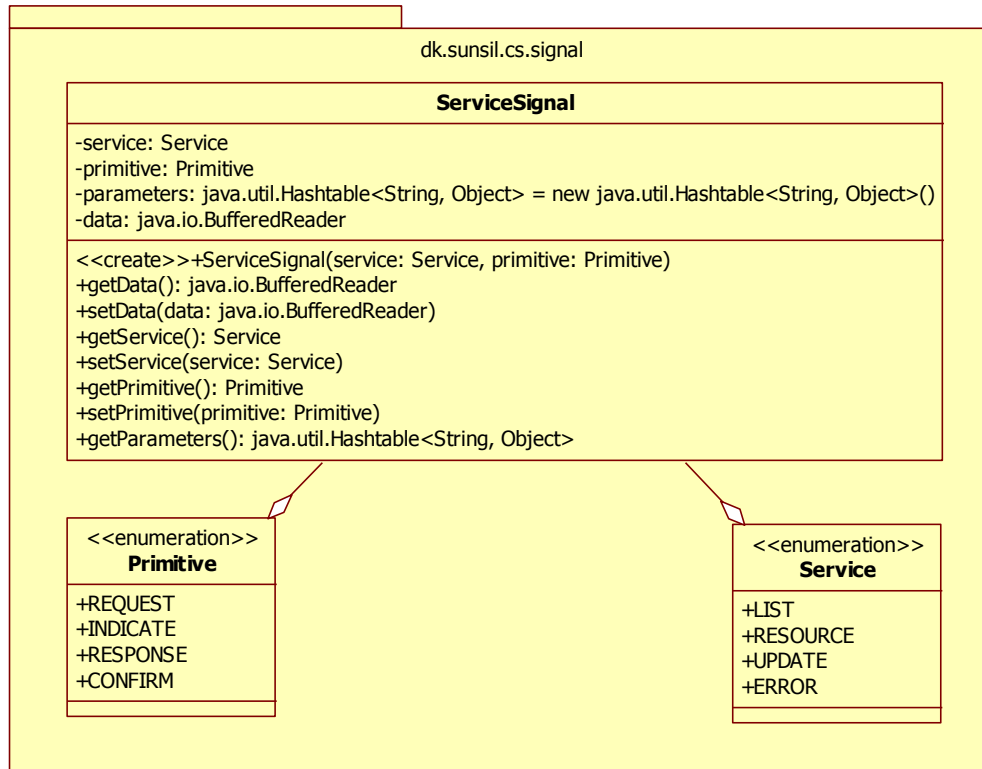


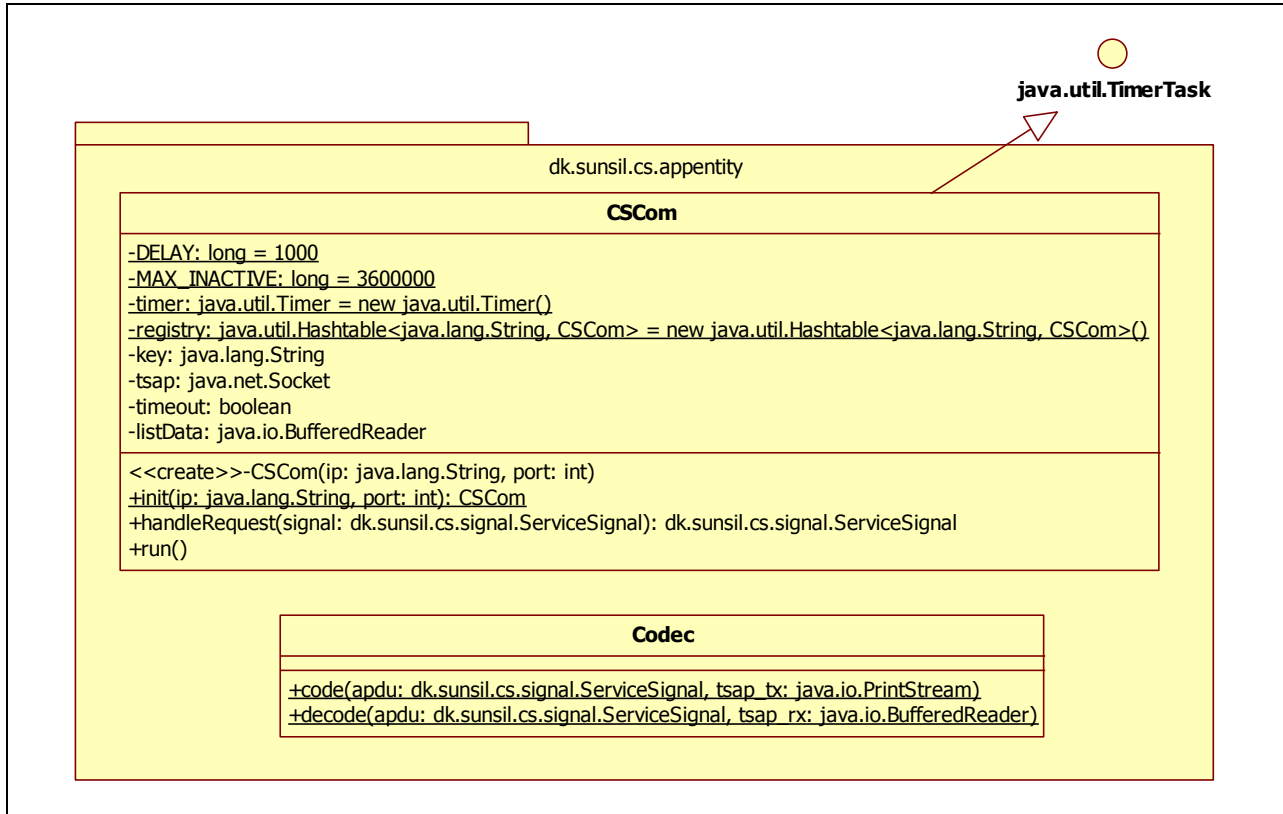
Figure 5-7: Class Diagram of Package *dk.sunsil.cs.signal*

The class *ServiceSignal* describes a service signal as presented in the system specification. Its private attributes correspond to the four parts of a service signal namely; service, primitive, parameters and data. The *parameters* attribute is a hash table in which all parameters are saved. It has only a GET-function used to get and add or read parameters from the hash table. All other attributes have GET- and SET-functions. The single public constructor initializes the *service* and *primitive* attributes.

The enumerations *Service* and *Primitive* simply enumerate the services and primitives required by the service signal.



### 5.3.7 Package *dk.sunsil.cs.appentity*



**Figure 5-8: Class Diagram of Package *dk.sunsil.cs.appentity***

The class *CSCCom* implements the *CSCCom* state machine as discussed in the specification. It inherits from the class *java.util.TimerTask* and overrides the *run()* function of this class. The private static attribute *DELAY* is constant and it specifies, in ms, how long the machine should wait for a response from the T400 device after it sends a request. The private static attribute *MAX\_INACTIVE* is also constant and given in ms but it specifies how long a *CSCCom* object is allowed to stay inactive, i.e. there is no communication. The private static attribute *timer* represents the *Timer* process as discussed in the specification of the *CSAppEntity*. The private static attribute *registry* stores all the *CSCCom* objects in use. The private attribute *key* is the key used to save the *CSCCom* object in the *registry*. The private attribute *tsap* is the Transport Service Access Point. It is simply a socket which connects to a T400 device. The private attribute *timeout* is used by the timer to send a timeout signal to the *CSCCom* object. The private attribute *listData* stores the data of the list service on the first list request and simply returns it on subsequent list requests since it does not change. The single constructor is private, meaning only the class can create instances of itself. The *ip* and *port* parameters of the constructor specify the IP-address and port of the T400 device to communicate with. The public static function *init(ip, port)* implements the init service as discussed in the specification. It first checks if there is a *CSCCom* object in the *registry* which is connected to the given *ip* and *port*. If there is, it simply returns this

object else it creates a new object with this *ip* and *port* and registers it in the *registry* before returning it. As such, only one *CSCom* object is connected to a T400 device at a time, meaning multiple users simultaneously communicating with a T400 device all share the same TCP connection from the centralized server to the T400 device. This is more efficient as it saves resources for multiple connections on both the server and the device and reduces the communication overhead to initiate TCP connections. The *init* function also schedules newly created *CSCom* objects as timer tasks in the *timer*. The *timer* then calls the *run()* function every *MAX\_INACTIVE* ms. The *run* function checks if the connection to the T400 device has been closed. If it has, the *CSCom* object has been inactive for too long and it is removed from the *registry*. The public function *handleRequest(signal)* handles the request for all other services. It is synchronized to prevent simultaneous execution by multiple threads. This avoids a thread requesting a service from reading the response to the request of another thread.

The class *Codec* implements the *Codec* process as discussed in the specification. The static public function *code(apdu, tsap\_tx)* codes an Application Protocol Data Unit (APDU) into an HTTP request and writes it into the transmission buffer of the socket which is specified by the parameter *tsap\_tx*. The static public function *decode(apdu, tsap\_rx)* does the opposite. It decodes an HTTP response from the T400 device into an APDU. The response is read from the receive buffer of the socket which is specified by the parameter *tsap\_rx*.

### 5.3.8 Package *dk.sunsil.cs.servlet*

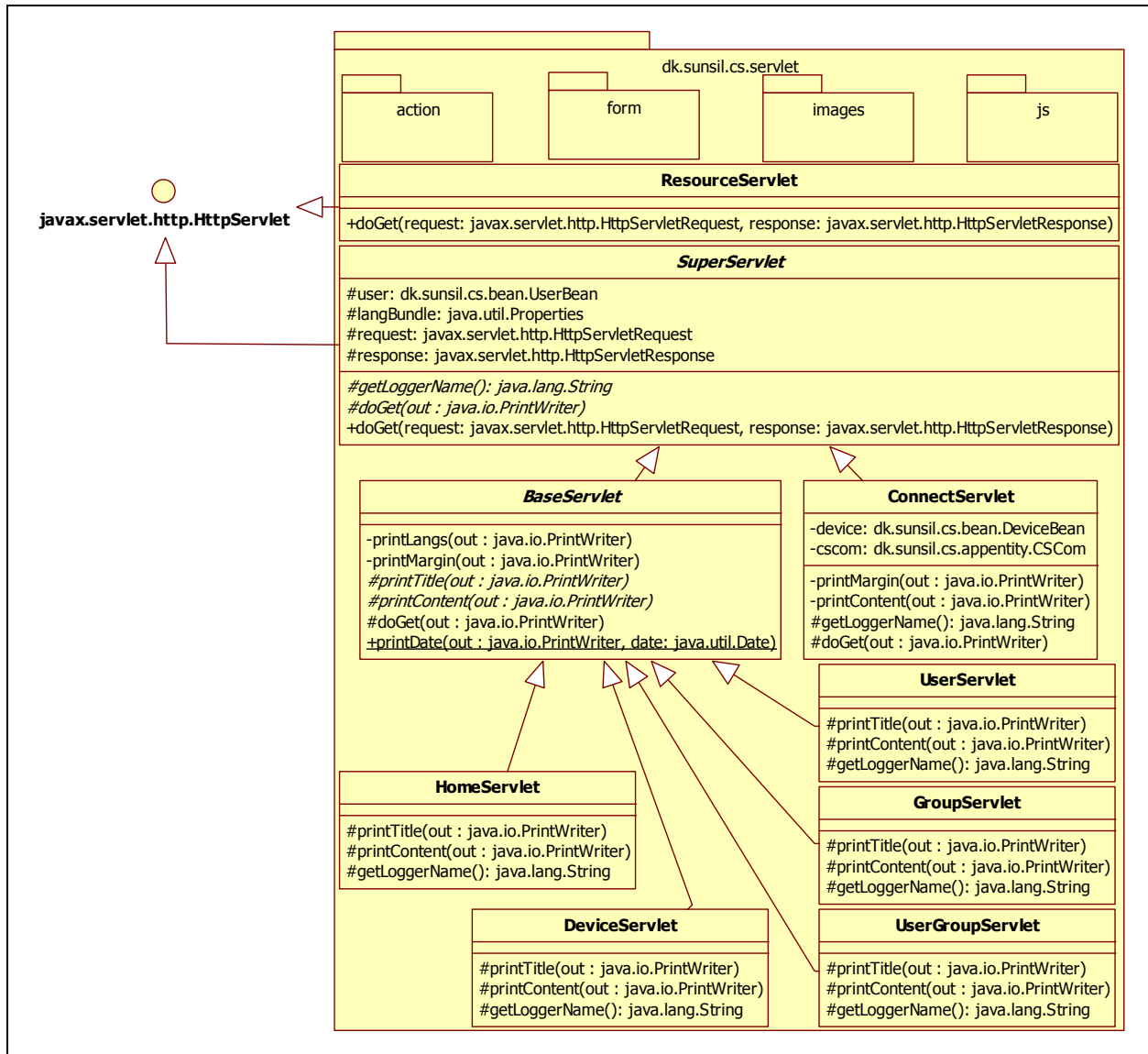


Figure 5-9: Class Diagram of Package `dk.sunsil.cs.servlet`

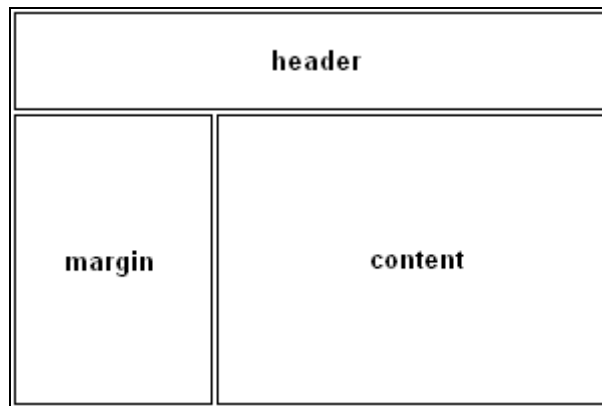
The sub-package *form* contains servlets which generate HTML-forms for adding or editing objects in the database while the sub-package *action* contains servlets used to carry out actions in the database. They are both discussed in detail in the next two sub-sections. The sub-packages *images* and *js* contain JPEG-images and javascrpts respectively, which are used by the application.

Servlets are exclusively used at the backend, i.e. the server side, of the web interface. All servlets inherit from the class `javax.servlet.http.HttpServlet` and override its public function

*doGet(request, response)*. This is the function called for the HTTP GET-method. Servlets have only default constructors. So, all servlet attributes are initialized in the *doGet* function. Both static and dynamic resources are used in the web application. All HTML-pages are dynamically generated at request. The static resources constitute the images and javascripts stored in the sub-packages *images* and *js* respectively. The class *ResourceServlet* is used to allocate the static resources.

The abstract class *SuperServlet* provides common attributes to all its subclasses. All its attributes are initialized in the function *doGet(request, response)* which is final, meaning no subclass can override this function. The private attribute *user* is the user bean for the currently logged in user. The private attribute *langBundle* is the language bundle which contains the textual translations of the user's chosen language. The private attributes *request* and *response* are the *request* and *response* parameters of the function *doGet(request, response)*. After the attributes have been initialized, the protected abstract function *doGet(out)* is called to generate the HTML code. In case of any exceptions, the protected abstract function *getLoggerName()* is called to get the logger name of a concrete subclass. The class name of the concrete subclass is returned as the logger name.

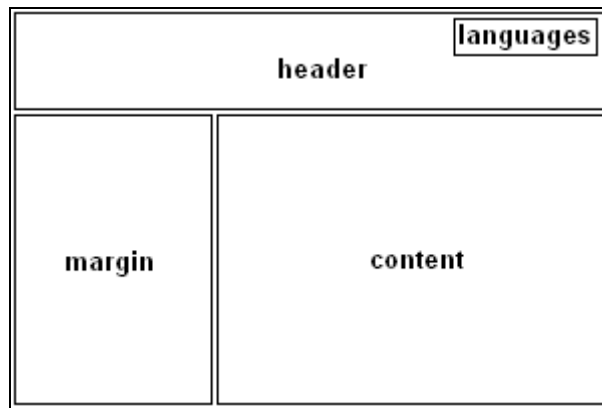
The class *ConnectServlet* inherits from the class *SuperServlet* and relays information between the currently logged in user and a T400 device. The private attribute *device* is the device bean for the device the user communicates with. The private attribute *cscom* is the *CSCom* instance which communicates with the given device. This servlet has the following template;



**Figure 5-10: ConnectServlet Template**

The class *ConnectServlet* generates the HTML-code such that it has the above template. The private function *printMargin(out)* is called such that what it prints is displayed in the **margin** section of the above template. It prints the list of resources available on the T400 device for selection by the user. Likewise, the private function *printContent(out)* is called such that what it prints is displayed in the **content** section of the above template. It prints the response of the service requested. The functions *getLoggerName()* and *doGet(out)* implement the abstract functions of the base class.

The abstract class *BaseServlet* inherits from the class *SuperServlet* and provides a common template to all its subclasses. This template is as follows;



**Figure 5-11: BaseServlet Template**

The *BaseServlet* also generates HTML-code such that it displays information according to the above template. The private function *printLangs(out)* is called such that it prints the other languages available in the web application in the **languages** section of the template above. The private function *printMargin(out)* is called such that it prints a list of the database tables, which the logged in user is allowed to view, in the **margin** section of the above template. The user can then select a table and its contents will be displayed in the **content** section of the above template. The protected abstract function *printTitle(out)* is called such that it prints the title of the HTML page. The protected abstract function *printContent(out)* is called such that it prints the contents of a database table in the **content** region of the above template. The protected function *doGet(out)* implements the abstract function of the base class and is final, meaning no subclass can override it. The public static function *printDate(out, date)* prints the date in following format;

dd.MM.yyyy HH:mm

where

- dd: 2-digit day of the month
- MM: 2-digit month
- yyyy: 4-digit year
- HH: 2-digit hour from 00 – 23 hours
- mm: 2-digit minute from 00 – 59 minutes

The classes *HomeServlet*, *UserServlet*, *DeviceServlet*, *GroupServlet* and *UserGroupServlet* all inherit from the class *BaseServlet* and implement all its abstract functions. They display a welcome message, the *User* table, the *Device* table, the *Groups* table and the *UserGroup* table respectively. All servlets and their relative paths are specified in the Web Application Deployment Descriptor, i.e. the *web.xml* file (see Appendix B).

### 5.3.9 Package *dk.sunil.cs.servlet.form*

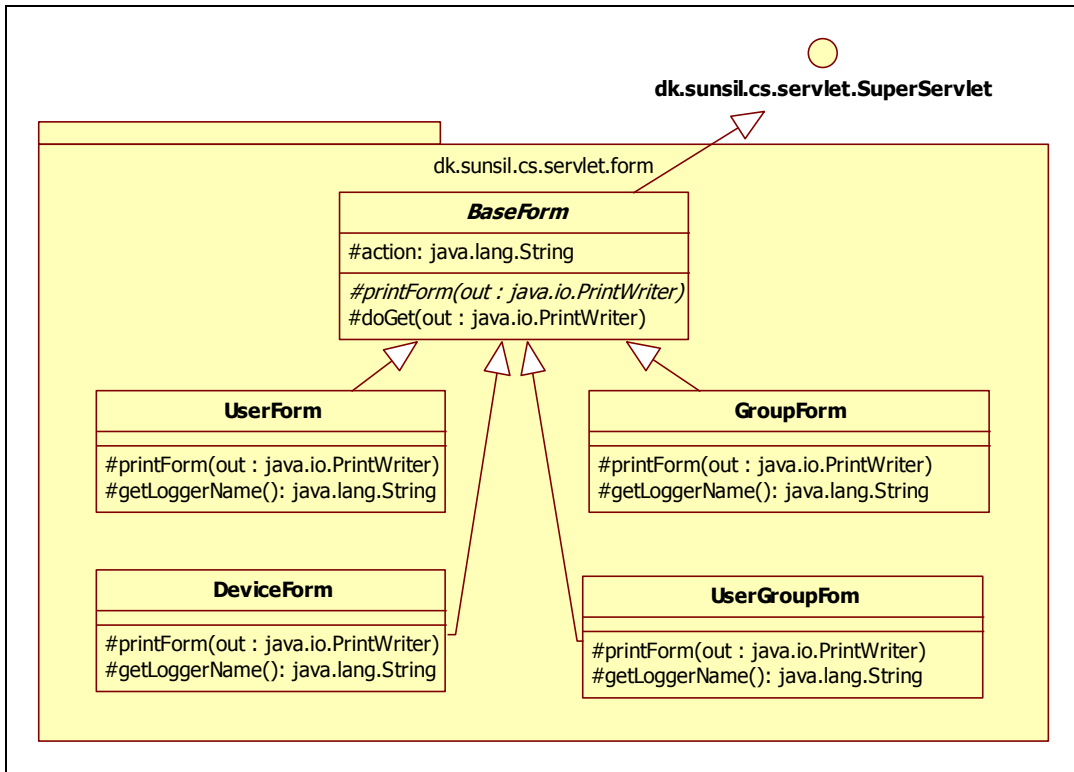


Figure 5-12: Class Diagram of Package *dk.sunil.cs.servlet.form*

The abstract class *BaseForm* inherits from the class *dk.sunil.cs.servlet.SuperServlet* and it is the base class for the classes which generate HTML forms for adding and editing objects in the database. The protected attribute *action* is a URL parameter which specifies whether an object should be added to the database or edited. The protected abstract function *printForm(out)* is called to print the required HTML form tag. The protected function *doGet(out)* implements the abstract function of its base class and it is final. Therefore, no subclass can override it.

The classes *UserForm*, *GroupForm*, *DeviceForm* and *UserGroupForm* generate the HTML forms for adding and editing users, groups, devices and usergroups respectively. They simply implement the abstract functions of their base class. Their definitions and mappings in the Web Application Deployment Descriptor are shown in Appendix B.

### 5.3.10 Package *dk.sunsil.cs.servlet.action*

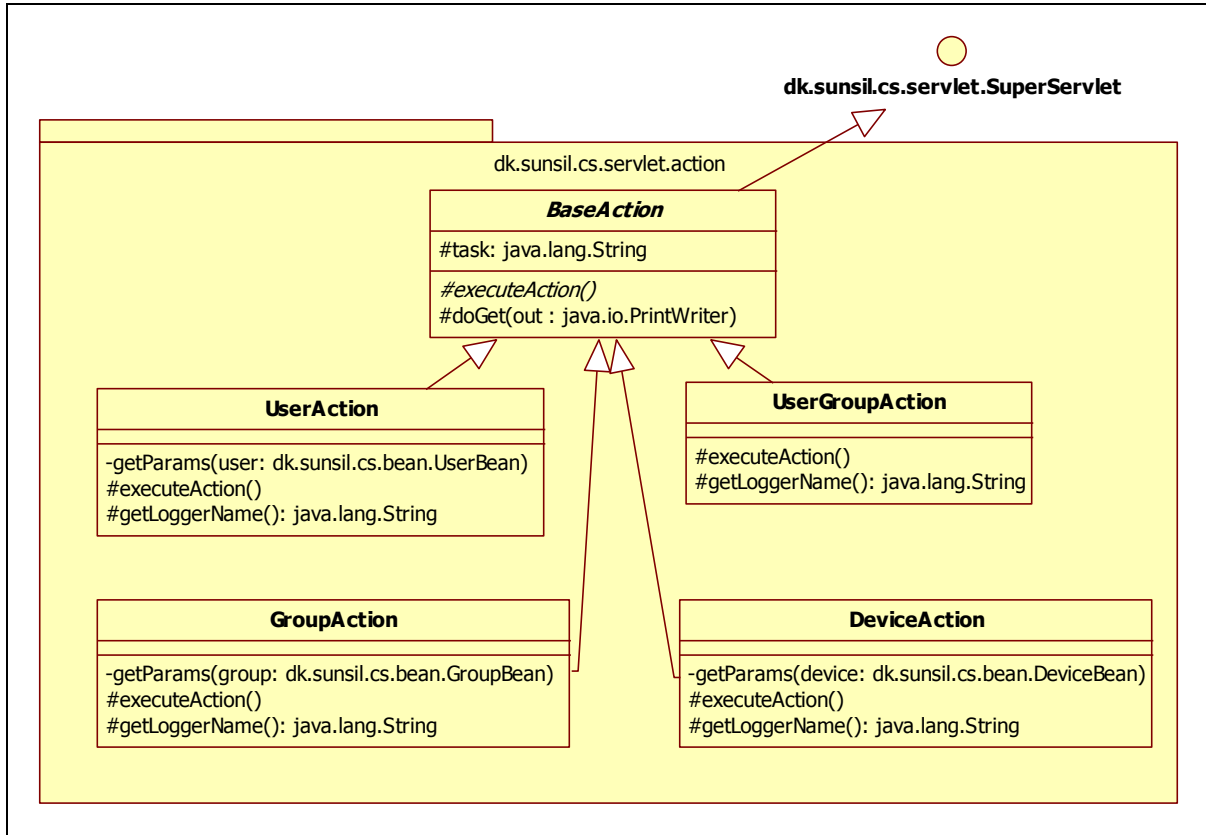


Figure 5-13: Class Diagram of Package *dk.sunsil.cs.servlet.action*

The abstract class *BaseAction* inherits from the class *dk.sunsil.cs.servlet.SuperServlet* and is the base class for the servlets which insert, update and delete objects from the database. The private attribute *task* is a URL parameter which specifies the action to be carried out. The protected abstract function *executeAction()* carries out the requested task and throws a *dk.sunsil.cs.dao.DAOException* if an error occurs. The function *doGet(out)* implements the abstract function of the base class and is final. It prints a success message if the requested task was performed successfully and a failure message otherwise.

The classes *DeviceAction*, *UserAction*, *GroupAction* and *UserGroupAction* all inherit from the class *BaseAction* and implement all its abstract functions. They insert, update and delete objects from the database tables *Device*, *User*, *Groups* and *UserGroup* respectively. The private functions *getParams* initialize their bean objects with URL parameters. They are defined and mapped in the Web Application Deployment Descriptor as given in Appendix B.

### ***5.3.11 Summary***

In sum, the centralized server has a three-tier architecture. Each tier has a set of software packages which implement specific functionalities of the tier. The centralized server provides a web interface to the user. This interface uses HTML and JavaScript on the frontend and servlets on the backend. The servlets belong to the uppermost tier and they dynamically generate the HTML code for a give web page on request. Data presented in the web pages are gotten either from the database or a T400 device. In the lowermost tier, Data Access Objects (DAO's) are used by the servlet to access the database while the Centralized Server Application Entity (CSAppEntity) is used to access a T400 device. The servlets communicate with the DAO's using logical beans and with the CSAppEntity using service signals, both of which belong to the middle tier.



## 6 System Testing

This chapter demonstrates some test scenarios of the whole system. A lot of unit tests were carried out during the implementation phase of the project. Details about all these tests will not be elaborated here. The test scenarios simply show what the browser displays to the user. The content of the web pages are shown and explained in the following sections.

The tests were carried out on the development PC, which has an installation of the centralized server. The centralized server was accessed via the localhost. The server listened to the port 8080, which is the default port of the Apache Tomcat. The centralized server web application is named sunsil. Therefore, it was access by the URL <http://localhost:8080/sunsil>.

### 6.1 Authentication

Users must authenticate with a username and password before being granted access to the centralized server web application. The following dialog box is shown for authentication;

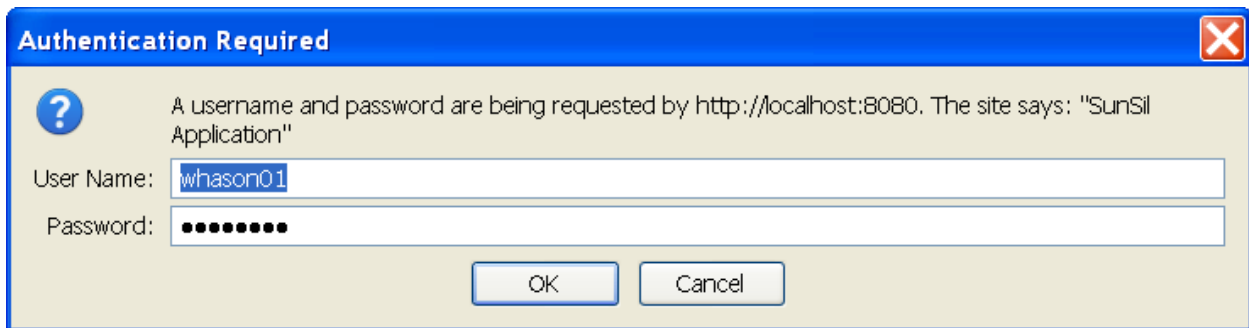


Figure 6-1: User Authentication Prompt

### 6.2 The Homepage

The homepage is displayed after a user successfully authenticates. It has the BaseServlet Template as discussed in the previous chapter (see Figure 5-11). It shows the name and last login date of the user. Below is the view of the homepage for the super user *whason01*.



**Figure 6-2: The Homepage**

The link *Deutsch* gives a German translation of this page. The link *Home* is as well the current page. The links *Device*, *User*, *Groups* and *UserGroup* displays the contents of their respective database tables. The *Groups* and *UserGroup* links are not shown for group users, since they are not allowed to view or modify these tables. The *User* link is in addition not shown for device users. The link *Edit my profile* results in a popup window containing a form with which the user can edit his personal details. Below is such a popup window for the current user.

### Edit my profile

Username:

Password\*:

Last name:

First name:

E-mail\*:

\* Required fields

**Figure 6-3: Edit Profile Form**

The user cannot change his username. The fields labeled with the asterisk are required. On sumitting the form, a JavaScript function is called to check if these fields are empty. If they are, a corresponding error message is shown indicating the empty field which must be filled.

### 6.3 The Device Page

The device page displays the devices the logged on user is allowed to view. It also enables the user to add, modify, delete or remotely connect to the device. The user is permitted to do just as much as his access rights requires, as discussed in the system specification. Below is a view of the device page for the super user *whason01*.

| [[Deutsch](#)]

## SunSil T400 Remote Access

Username: whason01 | E-mail: wedndah@yahoo.com | [[Edit my profile](#)] | [[Home](#)]

Database Tables	Device Table														
<a href="#">Device</a> <a href="#">User</a> <a href="#">Groups</a> <a href="#">UserGroup</a>					Action	did	dname	gid	location	ip	port	type	owner	last_connect	dentry_date
					1	SN0000	1		10.0.1.1	32121	UserDevice	fgkahn01		08.07.2009 14:06	
					2	SN0001	2		10.0.2.1	32121	UserDevice	cnmols01		08.07.2009 14:06	
					3	SN0002	3		10.0.3.1	32121	UserDevice	whason03		08.07.2009 14:06	
					4	SN0003	1		10.0.1.2	32121	UserDevice	cnande01		08.07.2009 14:06	
					5	SN0004	2		10.0.2.2	32121	UserDevice	jsfisc01		08.07.2009 14:06	
					6	SN0005	3		10.0.3.2	32121	UserDevice	sehans01		08.07.2009 14:06	
					7	SN0006	1		10.0.1.3	32121	UserDevice	ekhans01		08.07.2009 14:06	
					8	SN0007	2		10.0.2.3	32121	UserDevice	whason02		08.07.2009 14:06	
					9	SN0008	3		localhost	8192	UserDevice	whason01	03.10.2009 15:20	08.07.2009 14:06	
					10	SN0010	1		10.0.1.4	32121	PlantDevice			29.07.2009 12:18	

Figure 6-4: The Device Page

The icons , , and add, connect to, edit and delete a device respectively. Clicking on any of them, results in a popup window which enables the user to carry out the given task. The connect icon calls the connect page as described later in this chapter. The add and edit icons display forms which enables the user to add and edit a device respectively. The delete icon results in a dialog box which prompts the user to confirm or decline the deletion of the specified device. If a user is not permitted to carry out any given task, the icon for the task is not displayed. The add and edit forms as well as the delete prompt are as follows;

### Add row to Device table

dname\*:

gid:

location:

ip\*:

port\*:

type: UserDevice

owner:

\* Required fields

**Figure 6-5: Add Device Form**

### Edit row in Device table

did: 1

dname\*: SN0000

gid: 1

location:

ip\*: 10.0.1.1

port\*: 32121

type: UserDevice

owner: fgkahn01

\* Required fields

**Figure 6-6: Edit Device Form**

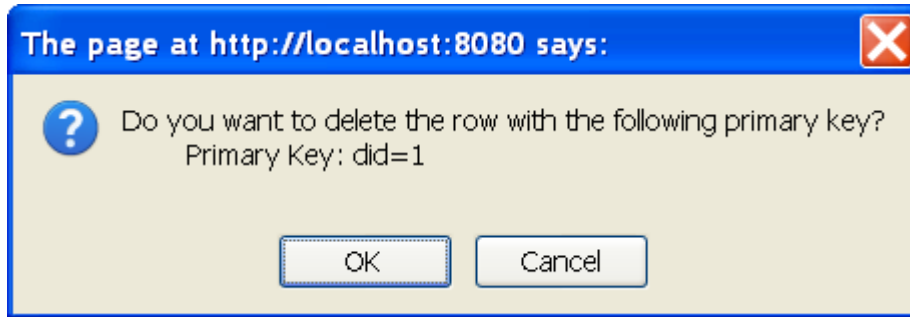


Figure 6-7: Delete Device Prompt

## 6.4 The User Page

The user page displays only those users, whom the currently logged on user is allowed to see. It can be accessed only by super or group users. It also enables the logged on user to add, modify or delete other users. The user is permitted to do just as much as his access rights allows, as discussed in the system specification. Below is a view of the user page for the super user *whason01*.

[ [Deutsch](#) ]

### SunSil T400 Remote Access

Username: whason01 | E-mail: wedndah@yahoo.com | [ [Edit my profile](#) ] | [ [Home](#) ]




Database Tables

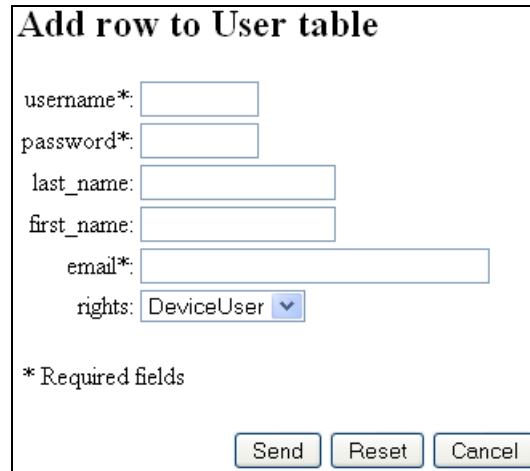
- [Device](#)
- [User](#)
- [Groups](#)
- [UserGroup](#)

**User Table**

Action	username	last_name	first_name	email	rights	last_login	uentry_date	lang
	cnande01	Anderson	Carsten	ca@sunsil.dk	DeviceUser		07.07.2009 18:03	en
	cnmols01	Mols	Carsten	cnmols@sunsil.dk	DeviceUser		07.07.2009 18:03	en
	ekhans01	Hansen	Erik	eh@sunsil.dk	DeviceUser	05.10.2009 19:14	07.07.2009 18:03	en
	fgkahn01	Kahn	Fleming	fk@sunsil.dk	GroupUser	06.10.2009 16:35	07.07.2009 18:03	en
	jsfisc01	Fischer	Jens	jf@sunsil.dk	GroupUser		07.07.2009 18:03	en
	sehans01	Hansen	Simone	sh@sunsil.dk	DeviceUser	09.07.2009 23:04	07.07.2009 18:03	en
	whason01	Asong	Wedndah	wedndah@yahoo.com	SuperUser	26.10.2009 17:11	18.06.2009 15:35	en
	whason02	Asong	Wedndah	wedndah_asong@yahoo.de	SuperUser		18.06.2009 15:35	en
	whason03	Asong	Wedndah	waa@sunsil.dk	SuperUser		07.07.2009 18:03	en

Figure 6-8: The User Page

The icons ,  and  add, edit and delete a user respectively. Clicking on any of them, results in a popup window which enables the logged on user to carry out the given task. The add and edit icons display forms which enables the logged on user to add and edit another user respectively. The delete icon results in a dialog box which prompts the logged on user to confirm or decline the deletion of the specified user. If the logged on user is not permitted to carry out any given task, the icon for the task is not displayed. The add and edit forms as well as the delete prompt are as follows;



**Add row to User table**

username\*:

password\*:

last\_name:

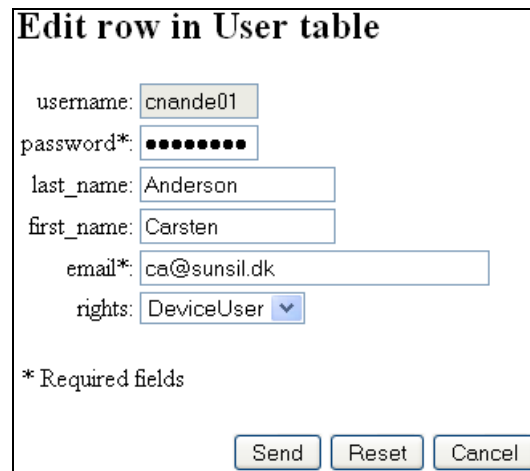
first\_name:

email\*:

rights: DeviceUser

\* Required fields

**Figure 6-9: Add User Form**



**Edit row in User table**

username:

password\*:

last\_name:

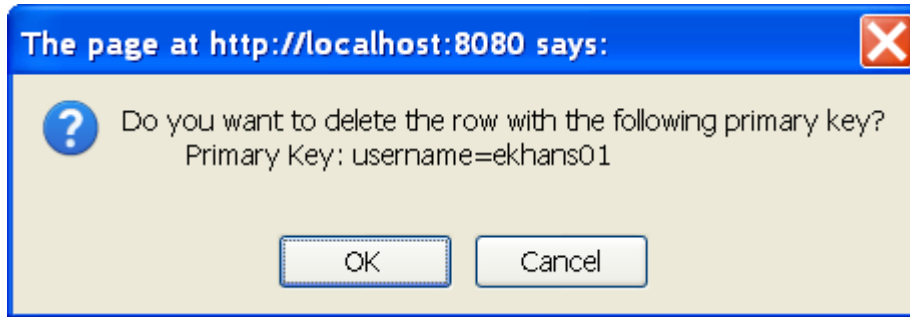
first\_name:

email\*:

rights: DeviceUser

\* Required fields

**Figure 6-10: Edit User Form**



**Figure 6-11: Delete User Prompt**

## 6.5 The Groups Page

The groups page displays all groups in the database. This page can be accessed only by super users. It also enables the user to add, modify or delete groups. Below is a view of the device page for the super user *whason01*.

| [[Deutsch](#)]

## SunSil T400 Remote Access

Username: whason01 | E-mail: wedndah@yahoo.com | [[Edit my profile](#)] | [[Home](#)]

Database Tables	Groups Table			
<a href="#">Device</a> <a href="#">User</a> <a href="#">Groups</a> <a href="#">UserGroup</a>	+			
	Action	gid	gname	location
		1	Toftlund	null
		2	Flensburg	
		3	Hamburg	

**Figure 6-12: The Groups Page**

The icons , and add, edit and delete a group respectively. Clicking on any of them, results in a popup window which enables the user to carry out the given task. The add and edit icons display forms which enables the user to add and edit a group respectively. The delete icon results in a dialog box which prompts the user to confirm or decline the deletion of the specified group. The add and edit forms as well as the delete prompt are as follows;

### Add row to Groups table

gname\*:

location:

\* Required fields

**Figure 6-13: Add Group Form**

### Edit row in Groups table

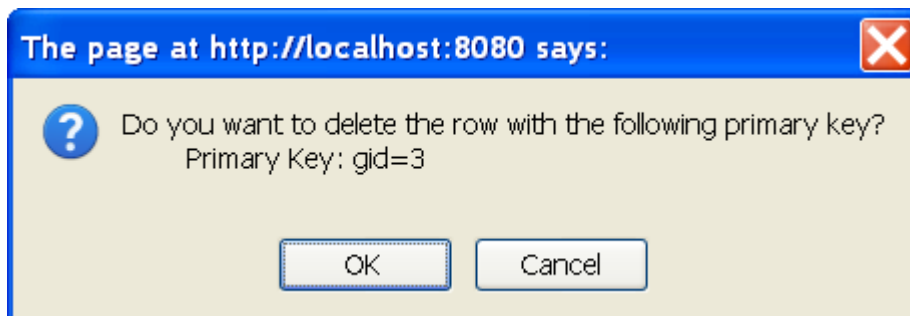
gid:

gname\*:

location:

\* Required fields

**Figure 6-14: Edit Group Form**

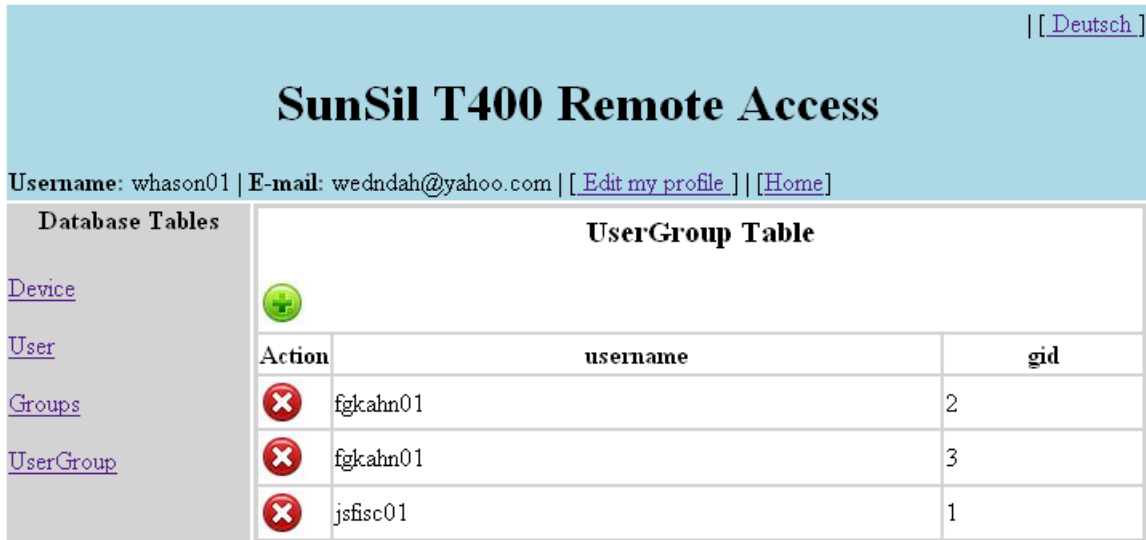


**Figure 6-15: Delete Group Prompt**



## 6.6 The UserGroup Page

The usergroup page displays all user groups in the database. This page can be accessed only by super users. It also enables the user to add or delete user groups. User groups cannot be edited. Below is a view of the usergroup page for the super user *whason01*.



The screenshot shows a web interface for SunSil T400 Remote Access. At the top right, there is a language selector link: [\[ \[Deutsch\] \]](#). The main header is "SunSil T400 Remote Access". Below the header, the user's login information is displayed: "Username: whason01 | E-mail: wedndah@yahoo.com | [[Edit my profile](#)] | [[Home](#)]". On the left side, there is a "Database Tables" menu with links for "Device", "User", "Groups", and "UserGroup". The "UserGroup" link is selected. The main content area is titled "UserGroup Table" and contains a table with three columns: "Action", "username", and "gid". The table has three rows of data, each with a delete icon (a red circle with a white 'X') in the "Action" column. Above the table, there is a green plus icon in a circle, which is used to add new user groups.






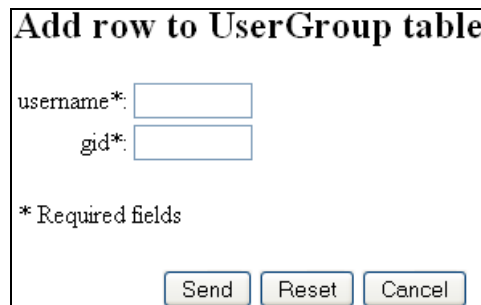
Action	username	gid
	fgkahn01	2
	fgkahn01	3
	jsfisc01	1

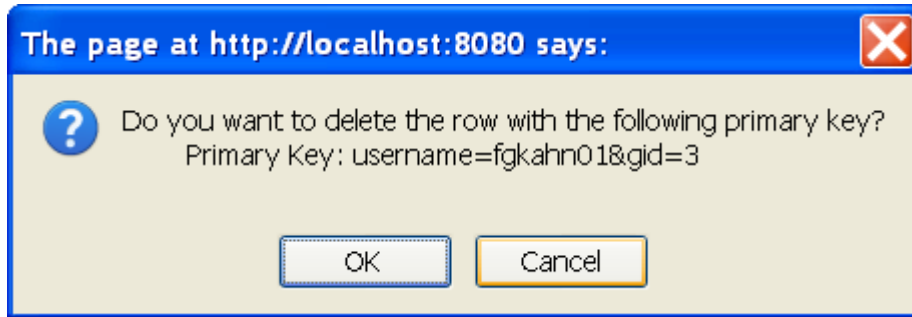
Figure 6-16: The UserGroup Page

The icons  and  add and delete a user group respectively. Clicking on any of them, results in a popup window which enables the user to carry out the given task. The add icon displays a form which enables the user to add a user group. The delete icon results in a dialog box which prompts the user to confirm or decline the deletion of the specified user group. The add form as well as the delete prompt are as follows;



The form is titled "Add row to UserGroup table". It contains two input fields: "username\*" and "gid\*", both marked as required fields. Below the input fields, there is a note: "\* Required fields". At the bottom of the form, there are three buttons: "Send", "Reset", and "Cancel".

Figure 6-17: Add User Group Form



**Figure 6-18: Delete User Group Prompt**

## 6.7 The Connect Page

So far, all pages accessed the database to make some changes. The connect page is the only page which provides an interface to communicate with a T400 device. This page is called from the device page when the user clicks the connect button of a device. The following figure shows the connect page for the device with device ID, did = 9.

Device Communication	
Device ID: 9   Device Name: SN0008	
Resources	Device Properties
<a href="#">Parameters</a> <a href="#">IO-Devices</a>	<b>Group ID:</b> 3 <b>Location:</b> <b>IP-Address:</b> localhost <b>Port Number:</b> 8192 <b>Type:</b> UserDevice <b>Owner:</b> whason01 <b>Last Connection:</b> 03.10.2009 15:20 <b>Entry Date:</b> 08.07.2009 14:06

**Figure 6-19: The Connect Page – Device Properties**

The links Parameters and IO-Devices is gotten from the list of resources provided by the device. They display the parameters and I/O-devices for the T400 device respectively. This list is retrieved using the list service. The list may differ from device to device. The following figure shows the result of clicking on the link Parameters;












Device Communication						
Device ID: 9   Device Name: SN0008						
Resources	Parameters					
<a href="#">Parameters</a>	Action	id	value	iConfig mnemonic	type	LangText
<a href="#">IO-Devices</a>		175	4	PRM_AUXH_T_START	TYPE_PRM_DEGREE_C	Start Temp
		176	4	PRM_AUXH_T_STOP	TYPE_PRM_DEGREE_C	Stop Temp
		177	4	PRM_SOLAR_T_START_1	TYPE_PRM_DEGREE_C	Dt Start
		178	4	PRM_SOLAR_T_STOP_1	TYPE_PRM_DEGREE_C	Dt Stop
		179	4	PRM_SOLAR_T_START_2	TYPE_PRM_DEGREE_C	Dt Start
		180	4	PRM_SOLAR_T_STOP_2	TYPE_PRM_DEGREE_C	Dt Stop
		181	4	PRM_SOLAR_OP_MODE	TYPE_PRM_VALUE	Op. Mode
		182	4	PRM_SOLAR_T_SET	TYPE_PRM_DEGREE_C	Set Temp
		183	4	PRM_SHR_T_START	TYPE_PRM_DEGREE_C	Start Temp
		184	4	PRM_SHR_T_STOP	TYPE_PRM_DEGREE_C	Stop Temp

Figure 6-20: The Connect Page - Parameters

The icon  is used to change the value of a parameter. On clicking the icon for a given parameter, a form is displayed which enables the user to enter a new value for the chosen parameter. The following form is shown for the parameter with id = 182;

Device Communication	
Device ID: 9   Device Name: SN0008	
Resources	Edit Form
<a href="#">Parameters</a>	id: <input type="text" value="182"/>
<a href="#">IO-Devices</a>	value*: <input type="text" value="4"/>
	* Required fields
	<input data-bbox="500 1625 634 1656" type="button" value=" &lt;&lt;&lt; Cancel "/> <input data-bbox="651 1625 743 1656" type="button" value=" Reset "/> <input data-bbox="760 1625 868 1656" type="button" value=" Send &gt;&gt;&gt; "/>

Figure 6-21: The Connect Page - Edit Form

## 7 Conclusion

A communication infrastructure for a distributed solar energy system was designed in this project. A complete system analysis was done. Due to time constraints, however, only parts of the system were designed, specified, implemented and tested. Four communication pairs were identified in the system, of which, only one of these pairs were further discussed. This leaves still a lot to be done in this project. However, the communication protocols developed here could as well be extended or modified for reuse by the other communication parties. An investigation of how this could be done will be of interest.

There was no real implementation of the communication protocol on a T400 device because this device is not yet developed beyond the conceptual stage. Instead, this was simulated, for testing purposes, on a computer located in the LAN of the centralized server. This is OK since the devices and the centralized server will share the same LAN, anyway. Setting up a real test environment with the T400 located in a mobile network and building a VPN tunnel to the centralized server will be of interest.

Setting up a web server for the District Heat Authority (DHA) and the Web DB entities of the system still has to be done. Also, an investigation on how to carry out the energy measurements required by the DHA is still to be done. The billing of the households still has to be specified.

### *Future work*

If the solar thermal plant is a combined heat and power plant, The T400 should be able to receive price signals from the plant, and then make a decision to consume heat or consume electricity for heating the water tank in the household. Or to sell as much thermal energy as possible to the grid, allowing the household water temperature to drop to a set minimum, to optimize earnings.

The T400 could also receive a price signal to help cooling the combined heat and power plant during night time operations, in case there is a shortage of electricity but no need or room for thermal energy storage available. Then the T400 will operate the solar thermal panel in reverse, until morning.

## References

- [1] Peter Gerdson & Peter Kröger (1994) Kommunikationssysteme 1 – Theorie, Entwurf & Messtechnik
- [2] Peter Gerdson & Peter Kröger (1994) Kommunikationssysteme 2 – Anleitung zum praktischen Entwurf (SDL)
- [3] LIAB – Linux in a Box (August 2009) <<http://www.liab.dk>>
- [4] MySQL (August 2009) <<http://www.mysql.com>>
- [5] DbVisualiser download (May 2009)  
<<http://www.minq.se/products/dbvis/download/index.jsp>>
- [6] Apache Tomcat (August 2009) <<http://tomcat.apache.org>>
- [7] Apache Ant (August 2009) <<http://ant.apache.org>>
- [8] Wikipedia, the free encyclopedia (August 2009) <<http://www.en.wikipedia.org>>

## Abbreviations

ARM	Advanced RISC Machine
CPU	Central Processing Unit
CSV	Comma Separated Values
DB	Database
FK	Foreign Key
FSM	Finite State Machine
GPRS	General Packet Radio Service
GSM	Global System for Mobile communications
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifier
JDBC	Java Database Connectivity
JPEG	Joint Photographic Experts Group
IP	Internet Protocol
LAN	Local Area Network
LED	Light-Emitting Diode
PC	Personal Computer
PCI	Protocol Control Information
PDU	Protocol Data Unit
PK	Primary Key
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SAP	Service Access Point
SDL	Specification and Description Language
SDU	Service Data Unit
SMS	Short Message Service
TCP	Transmission Control Protocol
URL	Universal Resource Locator
VPN	Virtual Private Network

# Appendix

## A The DbVisualiser Tool

The tool DbVisualizer is a universal database visualization tool, available for free download at [5]. It gives a complete overview of all tables, views and procedures in a database. It has in addition an SQL editor which shows SQL statements in coloured text. With this tool, SQL statements can also be sent to the database for execution and the results sent back and displayed in another view.

In order to view and manipulate a database with DbVisualizer, a connection to the database must first be created. The tool provides a wizard which walks you through the creation of a database connection. The information shown in the following figure is needed to create a connection to a database;

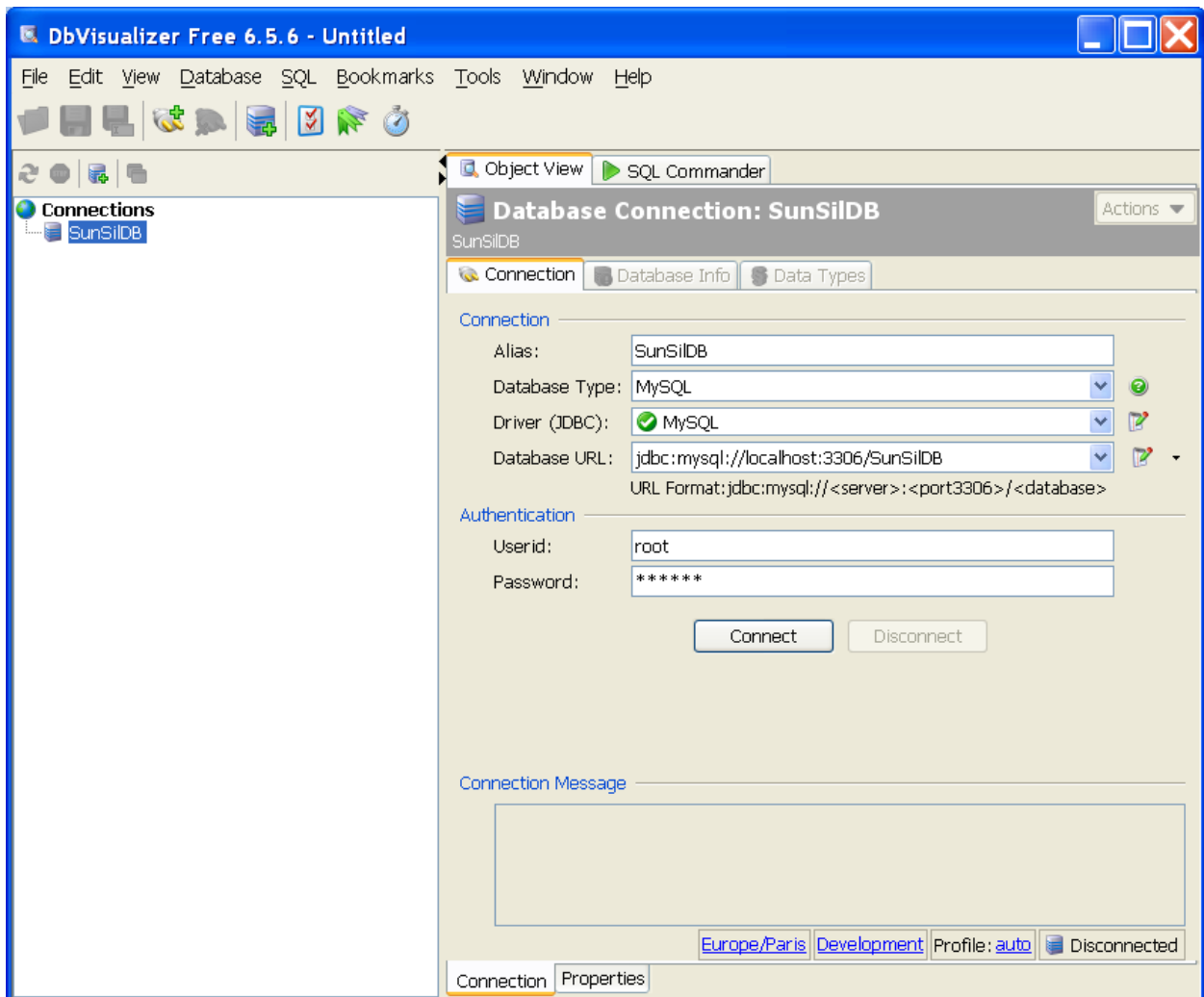
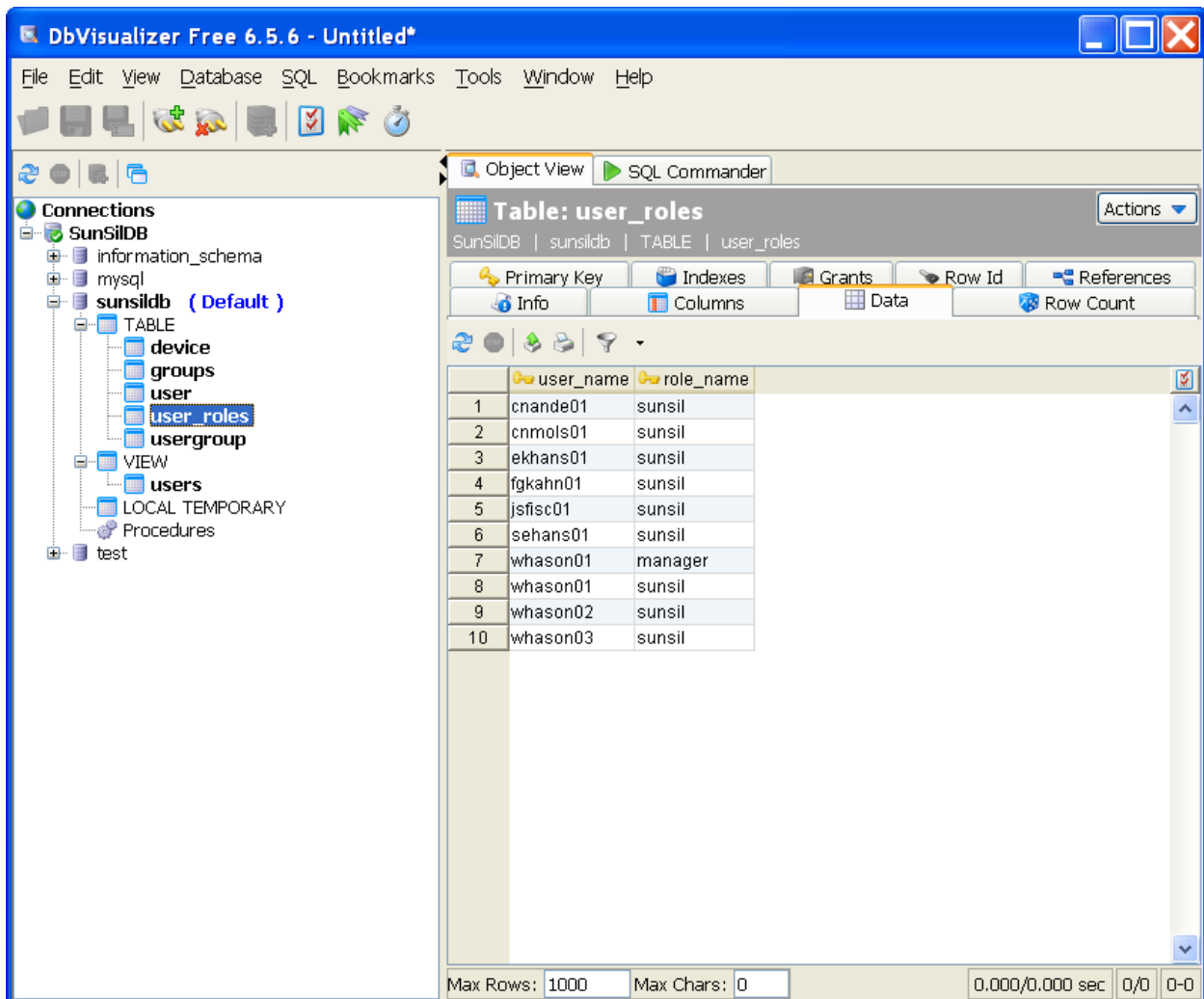


Figure A-1: DbVisualizer - Database Connection View

Below is a view of the data in the *user\_roles* table after the creation of tables and insertion of data;



**Figure A-2: DbVisualizer - Data View of user\_roles Table**

As can be seen from the above figure, there are different views of a single table. A view can be chosen by simply clicking on the required tab to the right of the above figure. The tables, views and procedures in a database can be navigated on the left panel of the above figure. Clicking the tab *SQL Commander* in the above figure brings you to the SQL editor where SQL statements can be typed and run.



## B Apache Tomcat

The two most important files used by the Apache Tomcat to configure a server application are the *context.xml* and *web.xml* files. They are as follows;

```
<?xml version="1.0" encoding="UTF-8"?>
<Context>

  <!--##### Realm Declaration #####-->
  <!-- Specify JDBC realm used for user authentication. Every user must
have the roll sunsil -->
  <Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
    driverName="com.mysql.jdbc.Driver"
    connectionURL=
      "jdbc:mysql://localhost:3306/test?user=root&password=321645"
    userTable="users" userNameCol="user_name" userCredCol="user_pass"
    userRoleTable="user_roles" roleNameCol="role_name"/>

  <!--##### Resource Declarations #####-->
  <!-- Specify JDBC resource used to access MySQL database -->
  <Resource name="jdbc/SunSILDB"
    auth="Application"
    type="javax.sql.DataSource"
    username="root"
    password="321645"
    driverClassName="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost:3306/test"
    maxActive="1"
    maxIdle="1"
    maxWait="3000"
    removeAbandoned="true"
    removeAbandonedTimeout="200"
    logAbandoned="true"/>

  <!-- Specify DAO Resources -->
  <Resource name="dao/user" auth="Application"
    type="dk.sunsil.cs.dao.impl.UserDAOImpl"
    factory="org.apache.naming.factory.BeanFactory"/>
  <Resource name="dao/device" auth="Application"
    type="dk.sunsil.cs.dao.impl.DeviceDAOImpl"
    factory="org.apache.naming.factory.BeanFactory"/>
  <Resource name="dao/group" auth="Application"
    type="dk.sunsil.cs.dao.impl.GroupDAOImpl"
    factory="org.apache.naming.factory.BeanFactory"/>
  <Resource name="dao/usergroup" auth="Application"
    type="dk.sunsil.cs.dao.impl.UserGroupDAOImpl"
    factory="org.apache.naming.factory.BeanFactory"/>

</Context>
```

**Listing B-1: context.xml**

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
  "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <display-name>SunSil Centralized Server</display-name>

  <context-param>
    <param-name>webmaster</param-name>
    <param-value>waa@sunsil.dk</param-value>
    <description>
      The EMAIL address of the administrator to whom questions
      and comments about this application should be addressed.
    </description>
  </context-param>

  <servlet>
    <servlet-name>home</servlet-name>
    <description> Homepage </description>
    <servlet-class>dk.sunsil.cs.servlet.HomeServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>connect</servlet-name>
    <description> Connects to a T400 device and relays info </description>
    <servlet-class>dk.sunsil.cs.servlet.ConnectServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>user</servlet-name>
    <description> Display User table </description>
    <servlet-class>dk.sunsil.cs.servlet.UserServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>device</servlet-name>
    <description> Display Device table </description>
    <servlet-class>dk.sunsil.cs.servlet.DeviceServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>

  <servlet>
    <servlet-name>group</servlet-name>
    <description> Display Groups table </description>
    <servlet-class>dk.sunsil.cs.servlet.GroupServlet</servlet-class>
    <load-on-startup>5</load-on-startup>
  </servlet>

```

```

<servlet>
  <servlet-name>usergroup</servlet-name>
  <description> Display UserGroup table </description>
  <servlet-class>dk.sunsil.cs.servlet.UserGroupServlet</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>resource</servlet-name>
  <description>
    Resource allocator for static resources like js, images, etc
  </description>
  <servlet-class>dk.sunsil.cs.servlet.ResourceServlet</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>UserForm</servlet-name>
  <description>
    Form for adding/editing a row in the User table
  </description>
  <servlet-class>dk.sunsil.cs.servlet.form.UserForm</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>DeviceForm</servlet-name>
  <description>
    Form for adding/editing a row in the Device table
  </description>
  <servlet-class>dk.sunsil.cs.servlet.form.DeviceForm</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>GroupForm</servlet-name>
  <description>
    Form for adding/editing a row in the Groups table
  </description>
  <servlet-class>dk.sunsil.cs.servlet.form.GroupForm</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>UserGroupForm</servlet-name>
  <description>
    Form for adding a row in the UserGroup table
  </description>
  <servlet-class>dk.sunsil.cs.servlet.form.UserGroupForm</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>UserAction</servlet-name>
  <description> Add/Edit/Delete a row in the User table </description>
  <servlet-class>dk.sunsil.cs.servlet.action.UserAction</servlet-class>
  <load-on-startup>5</load-on-startup>

```

```

</servlet>

<servlet>
  <servlet-name>DeviceAction</servlet-name>
  <description> Add/Edit/Delete a row in the Device table </description>
  <servlet-class>dk.sunsil.cs.servlet.action.DeviceAction</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>GroupAction</servlet-name>
  <description> Add/Edit/Delete a row in the Groups table </description>
  <servlet-class>dk.sunsil.cs.servlet.action.GroupAction</servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet>
  <servlet-name>UserGroupAction</servlet-name>
  <description> Add/Delete a row in the UserGroup table </description>
  <servlet-class>
    dk.sunsil.cs.servlet.action.UserGroupAction
  </servlet-class>
  <load-on-startup>5</load-on-startup>
</servlet>

<servlet-mapping>
  <servlet-name>home</servlet-name>
  <url-pattern>/home</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>home</servlet-name>
  <url-pattern>/</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>connect</servlet-name>
  <url-pattern>/connect</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>user</servlet-name>
  <url-pattern>/user</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>device</servlet-name>
  <url-pattern>/device</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>group</servlet-name>
  <url-pattern>/group</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>usergroup</servlet-name>

```

```
<url-pattern>/usergroup</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>resource</servlet-name>
  <url-pattern>/js/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>resource</servlet-name>
  <url-pattern>/images/*</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>GroupForm</servlet-name>
  <url-pattern>/form/group</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>UserGroupForm</servlet-name>
  <url-pattern>/form/usergroup</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>UserForm</servlet-name>
  <url-pattern>/form/user</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>DeviceForm</servlet-name>
  <url-pattern>/form/device</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>GroupAction</servlet-name>
  <url-pattern>/action/group</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>UserAction</servlet-name>
  <url-pattern>/action/user</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>UserGroupAction</servlet-name>
  <url-pattern>/action/usergroup</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>DeviceAction</servlet-name>
  <url-pattern>/action/device</url-pattern>
</servlet-mapping>

<session-config>
  <session-timeout>30</session-timeout>    <!-- 30 minutes -->
</session-config>
```

```
<!-- Declaration of Resource Requirements -->

<!-- Define a Security Constraint on this Application -->
<security-constraint>
  <web-resource-collection>
    <web-resource-name>Centralized Server</web-resource-name>
    <url-pattern>/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>sunsil</role-name>
  </auth-constraint>
</security-constraint>

<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>SunSil Application</realm-name>
</login-config>

<!-- Security roles referenced by this web application -->
<security-role>
  <description>
    The role that is required to log in to the SunSil Application
  </description>
  <role-name>sunsil</role-name>
</security-role>

</web-app>
```

**Listing B-2: Web Application Deployment Descriptor (web.xml)**

## **Acknowledgement**

I would like to thank everybody who contributed in one way or another to my master project and thesis. I am extremely grateful to my supervisors Prof. Dr. Wolfgang Fohl and Mr. Jens Fischer for the time they devoted to my thesis work. Their contributions were very valuable to my work. I want to also express my gratitude to all my colleagues at SunSil A/S for providing a good working atmosphere for my project. Special thanks go to Mr. Erik Hansen, Mr. Carsten Mols and Mrs. Lise Nielson, who helped do some brain storming for the requirements analysis of the project. Finally, I want to thank my family and friends, without whom my studies and consequently this thesis would not have been possible. I highly appreciate their moral support and words of encouragement which kept me going.

## Declaration

**I declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Master Thesis has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.**

*Hamburg, 10.12.2009*

---

**City, Date and Signature**