Technische Universität München
Fakultät für Informatik

<Bachelor/Master's> Thesis in Informatics

The Title of your Thesis

Your Name

Technische Universität München
Fakultät für Informatik

<Bachelor/Master's> Thesis in Informatics

The Title of your Thesis (english)

The Title of your Thesis (german)

Author: Your Name
Date: 15.MM.20NN
Supervisor: Prof. Bernd Brügge, Ph.D
Advisor: M.Sc. Juan Haladjian

Ich confirm that this <bachelor's/master's> thesis is my own work and I have documented all sources and material used.

München, den 15.MM.20NN, ...
(Your Name)

Acknowledgments

Some Text ...

Contents

Chapter 1

Introduction

1.1 Outline

- **Chapter 2** provides a brief overview of ..... We document the requirements elicitation and the analysis model of the framework by describing the high-level functionality (functional requirements) and requirements that are not directly related to functionality (nonfunctional requirements). The chapter also describes the analysis model in form of use cases, object models, and dynamic models for the system. It contains the complete functional specification and therefore it can be seen as the **Requirement Analysis Document** (RAD).

- **Chapter 3** provides an overview of the software architecture and the design goals and presents a survey of current architectures for similar systems. The purpose of this chapter is to make explicit the background information of the system architecture, their assumptions, and common issues the new system will address. The main part of this chapter documents the system design model with subsystem decomposition, hardware/software mapping, persistent data management, access control and security and global software control. This chapter represents the **System Design Document** (SDD).

- **Chapter 4** describes the detailed decomposition of subsystems into packages and classes. Class interfaces will be introduced and additional classes and interfaces which elaborate the model will be presented. This chapter is organized by packages and is considered as the **Object Design Document** (ODD).

- In **Chapter 5** we will present the prototype implemented during the thesis. After presenting the prototype we will review the requirements and evaluate the implementation. Furthermore we will present informal discussions with developers and their feedback.

- **Chapter 6** will show the conclusion and the preview to future work. We will summarize in this chapter some interesting issues which we encountered during the development process for the future development of the present approach.

Chapter 2

Background Theory

Chapter 3

Requirement Specification

3.1 Related Work

3.2 Scenarios

3.2.1 Problem Scenarios

3.2.2 Visionary Scenario

3.3 Functional Requirements

3.4 Nonfunctional Requirements

3.4.1 Usability

3.4.2 Performance

3.4.3 Supportability

3.4.4 Implementation Requirements

3.5 System Models

3.5.1 Use Case Model

3.5.2 Object Model

3.5.3 Dynamic Model

3.6 User Interface

Chapter 4

System Design

This chapter will show the transformation of the analysis model described in **??** into a system design model. It presents the preliminary architecture along with the design goals that influenced it.

First, in Section 4.1 we discuss the qualities of the system that should be optimized in the development. Section 4.2 will show a survey of the enabling technology for the solution. Then, we describe in sections 4.3 and 4.4 the subsystem decomposition in terms of subsystem responsibilities, dependencies among subsystems and subsystem mapping to hardware. The following next 3 sections 4.6, 4.5 and 4.7 give an outline of the major policy decisions such as persistent data management, access control, security and global software control. Finally, we discuss the boundary conditions of the developed system.

- Design Goals (Section 4.1)
- Enabling Technology (Section 4.2)
- Subsystem Decomposition (Section 4.3)
- Hardware/Software Mapping (Section 4.4 on the next page)
- Persistent Data Management (Section 4.6 on page 9)
- Access Control and Security (Section 4.5 on the next page)
- Global Software Control (Section 4.1 on page 9)

4.1 Design Goals

4.2 Enabling Technology

4.3 Subsystem Decomposition

In this section we introduce the initial decomposition of the system and describe the responsibilities and boundaries of each subsystem. First, based on the functional requirements and the models shown in the previous chapter, we identified N main subsystems: A, B, C and D subsystem. The subsystems are presented in detail in the following sections.

4.3.1 Proposed Software Architecture

In the analysis model described in Section **??** we introduced N main concepts for our target framework. First of all there is .....

4.3.2 A Subsystem

The `A` subsystem consists of ....

Services:

- Service A1
- Service A2

4.3.3 B Subsystem

The `B` subsystem is ...

Services:

- Service B1
- Service B2

4.3.4 C Subsystem

The `C` subsystem provides the

Services:

- Service C1
- Service C2

4.4 Hardware/Software Mapping

4.5 Access Control and Security

This section describes access control and security issues, such as ....

4.6 Persistent Data Management

This section describes the persistent data stored by the framework and the data management infrastructure required for it.

4.7 Global Software Control

This section describes how the global software control is designed. Figure 4.1 illustrates the main control flow involving the subsystems `xxx, xxx, xxx, xxx`.

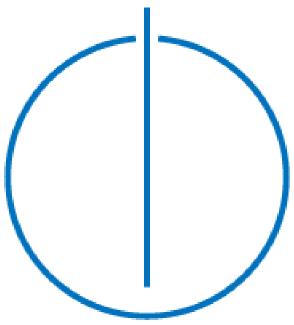

Figure 4.1: Main control flow involving the subsystems xxx, xxx, xxx, xxx (UML sequence diagram).

Chapter 5

Object Design

In Section 4.3 we divided the classes presented in the analysis model into different subsystems. In this section we will focus on packages and present additional classes to refine the model when necessary. Furthermore, attributes and methods are added to the existing classes. The object design is structured according to the subsystem decomposition presented in the system design in Section 4.3. We will show the packages in this chapter organized by subsystems.

- Package A (Section 5.1)
- Package B (Section 5.2)
- Package C (Section 5.3)
- Package D (Section 5.4 on the following page)
- Package E (Section 5.5 on the next page)

The diagrams for each package also show classes related to the package that are not part of the package for readability reasons.

5.1 Package A

Initial starting point for the design of the A package is the ...

5.2 Package B

As already described the aim of this package is ....

5.3 Package C

The aim of this package is ....

5.4 Package D

The aim of this package is ....

5.5 Package E

The aim of this package is ....

Chapter 6

Evaluation

In this chapter we will present the prototype. We will show how this prototype works and we will also present the graphical user interface. Furthermore we will evaluate the implementation by reviewing the requirements. Finally we will present some informal discussions with developers and summarize the feedback we got from these developers.

6.1 Prototypical Implementation

This section presents the implemented prototype with its graphical representation. In the actual version XXX we introduce a XXX application. It is capable of .....

Table 6.1 shows the complexity of the implementation by using a software metric **Source lines of code** (SLOC) which is typically used to estimate programming productivity or effort once the software is produced.

Language	files	blank	comment	code	scale	3rd gen. equiv
Objective C	65	1510	1371	5024	x 2.96	14871.04
C	5	563	782	1965	x 0.77	1513.05
C/C++ Header	71	729	1158	1198	x 1.00	1198.00
SUM:	141	2802	3311	8187	x 2.15	17582.09

Table 6.1: Count of source lines of code (SLOC) for the prototypical implementation.

6.1.1 Some Functionality

The main functionality of the presented framework is ....

6.1.2 More Functionality

Text...

6.2 Achievement of Objectives

In this section we will evaluate the implemented prototype against the requirement specification presented in Chapter 3.

6.2.1 Realized Functionality

Here, we will review the functional requirements described in Section 3.3 and evaluate how they are realized.

6.2.1.1 Functionality A

The main functional implementation of the ....

6.2.1.2 Functionality B

Text. ....

6.2.1.3 Functionality C

Text. ....

6.2.2 Nonfunctional Requirements Accomplishment

In this subsection we will evaluate the nonfunctional requirements.

6.2.2.1 Usability

One of our goals in system design was to design usable interfaces. ...

6.2.2.2 Reliability

We tried to make the framework extensible, but without losing the focus on reliability. In the prototypical implementation we ,,,,

6.2.2.3 Security

Text ....

6.2.2.4 Privacy

Privacy Text....

6.2.2.5 Performance

Performance Text...

6.2.2.6 Supportability

In the requirement specification we have defined supportability as all the actions related to the inherent quality of the target framework required to facilitate detection, isolation and repair of system anomalies. In system and object design we have demonstrated how our target framework can be extended. Furthermore, we have designed the system to support future technologies without major modifications by decomposing the system and using design patterns. Beyond the extensibility to other technologies, we also considered the portability of the system to other platforms......

6.3 Evaluation

Evaluation text....

Chapter 7

Conclusions and Future Work

In the actual version, our framework can be used for ....

7.1 Adaptivity

For our approach, we already have implemented xxx. This could also be automated to allow users to extend yyy.

7.2 Some other Ideas

Assuming that xxxx.

7.3 More ideas

In our approach, the users have to .....

Appendix A

Use Case Descriptions

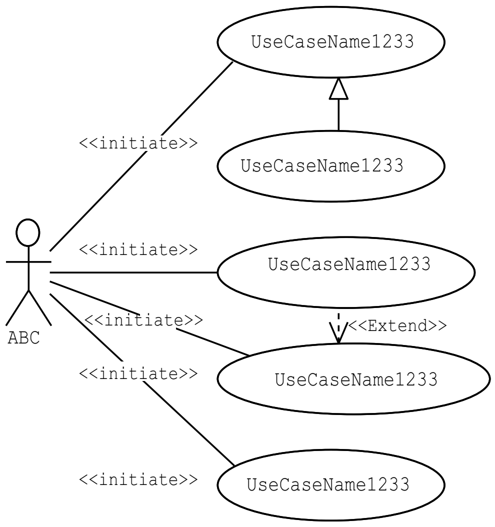

Figure A.1: All use-cases identified from the `User`'s and `Administrator`'s perspective (UML use-case diagram).

Table A.1: Use cases identified from the A's perspective.

Use Case Name	`UseCaseName`
Participating Actors	Initialized by `A`
Flow of Events	1. The `A` decides to xxx; 2. He opens yyy 3. He selects the "ZZZ" tab. 4. The System presents a list of ... 5. After finishing the User closes xxx

Use Case Name	`UseCaseName`
Participating Actors	Initialized by `A`
Flow of Events	1. The `A` decides to xxx; 2. He opens yyy 3. He selects the "ZZZ" tab. 4. The System presents a list of ... 5. After finishing the User closes xxx

Table A.4: Use cases identified from the B's perspective.

Use Case Name	`UseCaseName`
Participating Actors	Initialized by `A`
Flow of Events	1. The `A` decides to xxx; 2. He opens yyy 3. He selects the "ZZZ" tab. 4. The System presents a list of ... 5. After finishing the User closes xxx

Use Case Name	`UseCaseName`
Participating Actors	Initialized by `A`
Flow of Events	1. The `A` decides to xxx; 2. He opens yyy 3. He selects the "ZZZ" tab. 4. The System presents a list of ... 5. After finishing the User closes xxx

Use Case Name	`UseCaseName`
Participating Actors	Initialized by `A`
Flow of Events	1. The `A` decides to xxx; 2. He opens yyy 3. He selects the “ZZZ” tab. 4. The System presents a list of ... 5. After finishing the User closes xxx

Appendix B

Additional Information

Appendix C

Used Software

C.1 Software Development

C.2 Software Resources

Gimp http://www.gimp.org/
Free open source image manipulation software.

Audacity http://audacity.sourceforge.net/
Free open source software for sound recording and edition.

C.3 Authoring this Document

MacTEX http://www.tug.org/
MacTEX is a complete TEX system for Mac OS X, supporting TEX, LATEX, AMSTEX, ConTEXt, XeTEX and many other packages.

LYX 1.6 http://www.lyx.org/
LYX is an open source document processor. LYX is a front-end to the LATEX typesetting system. It allows writing a document based on the structure of the document, not the appearance. This document is written entirely with LYX.

Visual Paradigm for UML http://www.visual-paradigm.com/
Visual Paradigm for UML CASE Tool supporting UML modeling with UML 2.1. All diagrams in this thesis are designed with this tool.

Appendix D

Glossary

Adapter Pattern A design pattern that let objects from unrelated packages collaborate by adapting one interface to another.

Boundary Objects Objects that represent the interface between the system and the actors.

Bridge Pattern A design pattern which decouples an interface from the implementations so that implementations can be substituted.

C A programming language described by the ANSI C standard.

C++ Object oriented programming language.

Composite Pattern A design pattern which represents a hierarchy of different objects.

Design Pattern Defined and repeatable design solution to a common problem. Design patterns are proven concepts for creating the code to solve a given problem. A common description of several design patterns can be found in [1].

Framework An extensible collection of classes providing a standardized interface to a particular service or application which can be then instantiated to build a concrete application.

Graphical User Interface (GUI) Visual interface of any application or tool. It serves as a bridge between the user and an application/tool and allows the user to input information into the application/tool in a graphic method.

Integrated Development Environment (IDE) Computer software that assists developers in developing software.

Object Design Document (ODD) Documents the object design by describing the decomposition of subsystems into packages, classes and class interfaces.

Observer Pattern A design pattern which maintains the consistency across the states of one publisher and many subscribers.

Requirements Analysis Document (RAD) Documents the requirements elicitation and analysis by describing the requirements of the proposed system.

Strategy Pattern A design pattern allowing objects to use different algorithms and change them at runtime.

System Design Document (SDD) Documents the system design by describing the design goals, subsystems, hardware/software mapping, persistent data management, access control, control flow and boundary conditions.

Unified Modeling Language (UML) An Object Management Group (OMG) standard for modelling software artifacts. It is widely-used industry-standard language for the specification, visualization, construction, and documentation of the software system components or software systems. It contains a set of symbols for creating diagrams to model software systems.

Bibliography

[1] Erich Gamma, Richard Helm, John Vlissides, and Ralph E. Johnson. *Design Patterns Elements of Reusable Object-Oriented Software*. Addison-Wesley, Massachusetts, 2000.