Architecture for the E^4 Essential Enterprise Environment

Team 5

January 14, 2008

Contents

1 Introduction **7**

1.1 Background 7

1.2 The E^4 system 7

1.3 This architecture document 8

2 Requirements **9**

2.1 Functional requirements 9

2.1.1 Browsing 9

2.1.2 Editing 9

2.1.3 Analysis 10

2.1.4 Code documenting 10

2.1.5 Issue tracking 10

2.1.6 Meeting 10

2.1.7 Export 10

2.1.8 Extension 11

2.1.9 User management 11

2.2 Non-functional requirements 11

2.2.1 Maintainability 11

2.2.2 Security 11

2.2.3 Portability 11

2.2.4 Responsiveness 12

2.2.5 Consistency 12

2.3 Constraints 12

2.3.1 Subversion repository 12

3 Stakeholders **13**
3.1 End users . . . 13
3.1.1 Programmers . . . 14
3.1.2 Code reviewers . . . 14
3.1.3 Designers . . . 14
3.1.4 Researchers . . . 14
3.1.5 Extenders . . . 14
3.2 System administrators . . . 14
3.3 Managers . . . 15
3.4 Developers . . . 15

4 Use cases **17**
4.1 Login to the E^4 system . . . 18
4.2 View an artefact . . . 19
4.3 Edit an artefact . . . 20
4.4 Run an existing analysis . . . 21
4.5 Create a new analysis . . . 22
4.6 View an issue . . . 23
4.7 Create an issue . . . 24
4.8 Respond to an issue . . . 25
4.9 View a meeting . . . 26
4.10 Start a meeting . . . 27
4.11 Respond to a meeting . . . 28
4.12 Perform an export . . . 29
4.13 Create an export plan . . . 30
4.14 Manage users . . . 31
4.15 Update permissions . . . 32

5 Logical View **33**
5.1 Concepts . . . 33
5.1.1 Artefact . . . 34
5.1.2 Files . . . 34
5.1.3 Chunks . . . 34

5.1.4 Export rules 34
5.1.5 Types 34
5.1.6 Artefact groups 34
5.1.7 Document 34
5.1.8 Property 35
5.1.9 Users 35
5.1.10 User groups 35
5.1.11 Tools 35
5.1.12 Issue 35
5.1.13 Comment 35
5.2 Relationships 35

6 Development View **39**
6.1 Development Architecture 39
6.2 Development team and costs 40

7 Process View **43**
7.1 Overview 43
7.2 Tiers 44
7.2.1 Services 44
7.2.2 Business logic 45
7.2.3 Data tier 45
7.3 Rationale 46

8 Physical View **47**
8.1 Rationale 47
8.2 Quality 48
8.2.1 Availability 48
8.2.2 Reliability 49
8.2.3 Scalability 49
8.2.4 Performance 50

9 Implementation View **53**
9.1 Rationale 53
9.2 Hardware 53
9.2.1 Application Server 53
9.2.2 Firewall 54
9.2.3 Main switch 54
9.2.4 Backup server 54
9.2.5 Uninterruptible Power Supply (UPS) 55
9.2.6 Hardware costs 55
9.3 Software 55

10 Security View **59**
10.1 Authentication and authorization 59
10.2 User management 60
10.3 Logging 60
10.4 Hardware security 60

11 Conclusion **63**

Bibliography **65**

Chapter 1

Introduction

1.1 Background

Large programming projects require an infrastructure to support them. Contributors are often distributed geographically, work on different software platforms and have different roles within the project. Large amounts of code, documentation and other artefacts have to be organized and kept manageable. Large projects also often develop their own particular techniques, which can benefit from special tool support.

This document describes the architecture of a distributed collaboration environment for the development of the Essential Haskell Compiler (EHC) [2]. This project to develop a compiler for the Haskell programming language satisfies most of the conditions above. In particular, the use of specific techniques and technologies is very developed in this project. The architecture of the system described in this document addresses this by providing tool support for these techniques.

1.2 The E^4 system

We call the system described in this document E^4 or "E^4 Essential Enterprise Environment", in the good tradition of recursive acronyms like GNU (GNU's Not Unix) [3] and PHP (PHP: Hypertext Preprocessor) [4]. Although the "Essential" in the name is a reference to the EHC, the system can also be used for other (large) programming projects.

1.3 This architecture document

The architecture in this document is described according to the IEEE 1471 standard [8]. Furthermore, within this standard the viewpoints from the "4+1" view model [9] are used, and the Unified Modeling Language (UML) [6] is used as an architecture description language (ADL).

The "4+1" view model describes four viewpoints, which provide different ways of looking at the system:

Logical view This viewpoint is concerned with objects from the problem domain, and mapping functional requirements to those objects. The logical view on the system is described in chapter 5.

Process view This viewpoint addresses concurrency and distribution of the objects from the logical view to threads of control. The process view on the system is described in chapter 7.

Development view This viewpoint focuses on the organization of the actual software packages and modules the system is divided into. It is described in chapter 6.

Physical view This viewpoint describes the mapping of the software to physical nodes it will run on. It is described in chapter 8.

Use cases This view describes a set of scenarios or use cases that tie the other views together. It is the "+1" in the "4+1" view model. It is described in chapter 4.

In this document, we present two additional viewpoints to further clarify the architecture described:

Security This viewpoint describes the security aspects of the system. It shows how the system architecture prevents users from accessing files and performing actions when that is not allowed. It is described in chapter 10.

Implementation This viewpoint presents a proposed implementation, in terms of actual hardware. It is described in chapter 9.

Chapter 2

Requirements

In this chapter, we list all the requirements for the system. These are extracted from [7].

2.1 Functional requirements

This section describes the functional requirements, by listing the available functions of the system. Larger, more expanded explanations in the form of use cases are described later in chapter 4.

2.1.1 Browsing

Artefacts stored of the project being managed by E^4 can be browsed and viewed. Files can be located by browsing the file system as it is stored in the subversion repository. The file itself can be viewed either as plain text, or pretty printed according to its type. Within a file, chunks can also be individually selected and viewed, again as plain text or pretty printed according to their type. If a chunk includes another chunk, this is shown hierarchically.

If the type of the artefact allows references to other artefacts, references will be used to allow the user to navigate to the referenced artefact(s). Examples of such references are source files that import other files, makefiles that reference source files, etc.

2.1.2 Editing

Artefacts in the project can be edited. The artefact can be selected by browsing as described in 2.1.1. The type of the artefact being edited is used

to offer type specific functionality: completion of language keywords and constructs, completion of identifiers, and syntax checking.

2.1.3 Analysis

Reports can be generated, based on the properties of artefacts in the project. These properties include, but are not limited to, the number of lines in an artefact, and the number of words in an artefact. Users can query these properties to generate report about the project being managed.

2.1.4 Code documenting

Documentation for the project can be edited as described in 2.1.2. In addition, documentation can be linked to the artefact or group of artefacts it describes.

2.1.5 Issue tracking

Bug reports and feature requests (issues) for the project can be stored and managed. An issue can hold at least the following information: at least one comment, describing the bug or feature, followed by other comments documenting the progress being made; links to all relevant artefacts; attachments of documents related to the issue. The issue as classified as a bug or a feature, and requirements and limitations on the number of comments, attached artefacts, etc. can be imposed depending on this classification.

2.1.6 Meeting

Users have the ability to hold meetings online. During this meeting, they can exchange information by typing, talking (possibly supported by video), and using files. They can also link to issues entered into the system. This information is shown on a 'blackboard', so that all users have it at their disposal.

2.1.7 Export

A source distribution or documentation for the project can be exported. Since these are derived artefacts, the system will first build them according to a specification written by the users of the system. They will then be made available to the user who requested the export.

2.1.8 Extension

Users can add support for new artefact types to the system. They fill in the required information about the type, as well as source code for the functions required from the type (such as those needed for syntax highlighting). After the type has been added, all functionality that uses type information will be able to handle the new type.

2.1.9 User management

Using user management, security related settings can be changed. Users can be created, modified, and added to user groups. Users or groups can be given permission to read, or modify, groups of artefacts; they can also be given permission to perform certain actions, as described in this chapter.

2.2 Non-functional requirements

In this section, several non-functional requirements that are important in E^4 will be listed. These specify not functionality that the system will have, but properties of the system as a whole. Each non-functional requirement will be shortly explained.

2.2.1 Maintainability

It should be easy to maintain the E^4 system, and to add new functionality to the system. This does not include extending the system with new types, for which you are referred to 2.1.8.

2.2.2 Security

The system should be secure from unauthorized access to the system or artefacts. Users who have no permissions to modify an artefacts will not be able to do so. Persons without a user account for the project will not be able to access the project artefacts, unless they are made available outside the E^4 system.

2.2.3 Portability

There will be a web based interface to perform all actions, allowing it to be used cross platform. There will also be platform native interface for performing each action, to provide a more integrated experience for the users.

2.2.4 Responsiveness

The common activities – browsing, editing, documenting, issue tracking, user management – shall be responsive, giving a reaction to user input within 0.5 s for the native interface, or 5 s for the web-based interface. Other activities – analysis, export, extension – may take longer. If they do, progress and estimated time left will be indicated.

2.2.5 Consistency

The repository of files and the data used for analysis might not always be consistent. There will be an indication to the user of the system about this inconsistency. It will also indicate how long it will take the system to become consistent.

2.3 Constraints

This section described constraints upon the E^4 system, such as technologies that are required to be used.

2.3.1 Subversion repository

The files of the project will be available in a Subversion [5] repository. This will allow the users to check out, modify and commit the files as they see fit. All system functionality accessing files also access this repository.

Chapter 3

Stakeholders

This chapter lists all stakeholders. These are the people or organizations that have an interest in the development, deployment, maintenance and/or use of the E^4 system. Note that a single actual person can be categorized as multiple stakeholders, for example; a person might be both a programmer and code reviewer.

We recognize the following groups of stakeholders:

End users These include all of the following stakeholders:

- **Programmers** Day-to-day programmers of the EHC.
- **Code reviewers** People who, on a regular basis, review the source code of the EHC.
- **Designers** People who design the global architecture of the EHC.
- **Researchers** Scientists who use the EHC to do research.
- **Extenders** People who extend the EHC with new artefact types.

System administrators People responsible for the deployment and day-to-day maintenance of the E^4 system.

Managers The people who acquire the E^4 system and are responsible for managing the previous stakeholders.

Developers The programmers building the E^4 system. In other words, the people who will be using this architecture document to develop E^4 itself.

3.1 End users

The end users are the people who use the E^4 system and thus are the stakeholders with functional requirements.

3.1.1 Programmers

Programmers use the E^4 system to develop the EHC. They will edit code and maintain the appropriate documentation. They will collaborate using the bug tracking, feature request and meeting tools.

3.1.2 Code reviewers

Code reviewers use the E^4 system to review the code of the EHC. They will use the code browsing tool. They will use the bug tracking, feature request and meeting tools to collaborate with each other and with the programmers of the EHC.

3.1.3 Designers

Designers use the E^4 system to design the architecture of the EHC. They will use the code browsing, code analysis and documentation tools. They might also use the meeting tool to collaborate.

3.1.4 Researchers

Researches use the E^4 system to do research in the area of compiler construction. They will primarily use the coding tools: code editing, code analysis and documentation.

3.1.5 Extenders

Extenders use the E^4 system to extend the EHC system with extra functionality. They will most likely also be part of another user group (e.g. programmers or researchers). They are responsible for adding and maintaining artefact types such as .hs, .ag or .rul.

3.2 System administrators

System administrators are responsible for the installation and maintenance of the E^4 system. They will install the software on servers and apply updates to it.

3.3 Managers

Managers are responsible for managing the users and system administrators. They will also decide whether or not to purchase the E^4 system. Finally, during deployment and maintenance of the system, they are also responsible for managing the users and their respective permissions.

3.4 Developers

The developers will develop the E^4 system. They will do so using the latest version of this architecture document. The final developed product will meet the agreed upon requirements. They can also provide bug fixes or build extra functionality into E^4 when contracted to do so.

Chapter 4

Use cases

This chapter describes all the use cases that will be fulfilled by the E^4 system. These use cases describe all the functionality provided by the system. They do so by listing the interaction between the 'user' and the 'system'.

Here, the term 'user' refers to one of the five types of end users, namely: Programmer, Code reviewer, Designer, Researcher and Extender (see chapter 3 Stakeholders). The term 'system' refers to the E^4 system.

4.1 Login to the E^4 system

Primary actor:	All users
Stakeholders:	Users and Managers
Requirements:	User management 2.1.9
Preconditions:	None
Description:	A user wants to login to the E^4 system. If she provides correct credentials the system authenticates her and allows the user to enter other use cases.

Basic flow	
Actor action:	**System responsibility**
1. User requests a service from the system.	2. System checks to see if service is protected.
	3. System requests credentials.
4. User provides the requested credentials	5. System compares the provided credentials with known credentials.
	6. System continues processing the original request for a services.
Alternative flows	
	3a. System finds that the requested service is not protected. Continue with step 6.
	5b. System finds that the user has provided incorrect credentials.
	6b System shows an error to the user. Continue with step 3.

Frequency of occurrence:	Nearly continues. Every time a user request a service from the system.
Technical notes:	This might be implemented by using some sort of session identifier, such that the user doesn't have to enter her credentials every time she requests some service. However the use case remains the same, the system will validate the users credentials (a session indentifier in this case) on each service request.

4.2 View an artefact

Primary actor:	Programmers, code reviewers, designers, reseachers
Stakeholders:	Users
Requirements:	Browsing 2.1.1
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to view an specific artefact or a list of artefacts. The system provides the requested artefact to the user.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific artefact.	2. System shows the requested artefact.
Alternative flows	
1a. User requested a list of artefacts.	2a. System shows a list of artefacts, allowing the user to choose one.
	2b. System finds that the requested artefact does not exist.
	3b. System returns an error to the user.
	2c. System finds that the user does not have the required credentials to view the requested artefact.
	3c. System returns an error to the user.

Frequency of occurrence:	Frequently, multiple times per session.

4.3 Edit an artefact

Primary actor:	Programmers, code reviewers, designers, reseachers
Stakeholders:	Users
Requirements:	Editing 2.1.2
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to modify a specific artefact or create a new artefact. The system saves the changes to the repository.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific artefact.	2. System shows the requested artefact.
3. User modifies the artefact.	4. System provides context sensitive assistance, e.g. name and syntax completion.
5. User indicates she wants to save the artefact.	6. System stores the artefact in the global repository.
Alternative flows	
1a. User indicates she wants to create a new artefact.	2a. System shows an empty artefact of the correct type. Continue with step 3.
	2b. System finds that the requested artefact does not exist.
	3b. System asks the user if she wants to create a new artefact using that name.
4b. User responds with Yes or No.	5b. When yes, continue with step 2a. When no, continue with step 1.
	2c. System finds that the user does not have the required credentials to edit the requested artefact.
	3c. System returns an error to the user.

Frequency of occurrence:	Frequently, multiple times per session.
Notes:	When working with the native client, the files have to be checked out and committed using a SVN client. This process is not described in this use case since the SVN client falls outside the scope of the E^4 system.

4.4 Run an existing analysis

Primary actor:	Designers, reseachers
Stakeholders:	Users
Requirements:	Analysis 2.1.3
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to run an existing analysis. The system executes the specified analysis and returns the results to the user.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific analysis.	2. System executes the analysis.
	3. System returns the results of the analysis to the user.
Alternative flows	
	2a. System finds that the requested analysis does not exist.
	3a. System returns an error to the user.
	2b. System finds that the user does not have the required credentials to run the requested analysis.
	3b. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.

4.5 Create a new analysis

Primary actor:	Designers, reseachers
Stakeholders:	Users
Requirements:	Analysis 2.1.3
Preconditions:	User is authenticated by UC 4.1
Description:	A user creates a new analysis based upon the available data in the system. The system stores the created analysis for later use.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific analysis.	2. System shows the requested analysis.
3. User modifies the analysis.	
4. User indicates she wants to save the analysis.	4. System stores the analysis for later use.
Alternative flows	
1a. User indicates she wants to create a new analysis.	2a. System shows an empty analysis. Continue with step 3.
	2b. System finds that the user does not have the required credentials to create an analysis.
	3b. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.

4.6 View an issue

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Issue tracking 2.1.5
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to view an issue or a list of issues (these might be bugs or feature requests). The system provides the requested issue.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific issue.	2. System shows the requested issue.
Alternative flows	
1a. User requested a list of issues.	2a. System shows a list of issues, allowing the user to choose one.
	2b. System finds that the requested issue does not exist.
	3b. System returns an error to the user.
	2c. System finds that the user does not have the required credentials to view the requested issue.
	3c. System returns an error to the user.

Frequency of occurrence:	Occasionally, multiple times per session.

4.7 Create an issue

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Issue tracking 2.1.5
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to create a new issue. The system saves the new issue.

Basic flow	
Actor action:	**System responsibility**
1. User indicates she wants to create a new issue.	2. System shows an empty form.
3. User supplies the type of issue (bug or feature request).	
4. User supplies any other data for the actual issue (including attachments and the like).	
5. User supplies links to relevant artefacts (documents, chunks etc.).	
5. User indicates she wants to save the issue.	6. System saves the issue.
Alternative flows	
	2a. System finds that the user does not have the required credentials to create a new issue.
	3a. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.
Notes:	The term ‘form’ used in step 2 can be implemented in different ways, it does not imply an actual graphical form.

4.8 Respond to an issue

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Issue tracking 2.1.5
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to respond to an existing issue. The systems saves the response with a reference to the original issue.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific issue.	2. System shows the requested issue.
3. User supplies data for the response (including attachments and the like).	
4. User supplies links to relevant artefacts (documents, chunks etc.).	
5. User indicates she wants to save the response.	6. System saves the response with a reference to the original issue.
Alternative flows	
	2a. System finds that the user does not have the required credentials to respond to an issue.
	3a. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.

4.9 View a meeting

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Meeting 2.1.6
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to view a meeting or a list of meetings. The system provides the requested meeting.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific meeting.	2. System shows the requested meeting.
Alternative flows	
1a. User requested a list of meeting.	2a. System shows a list of meeting, allowing the user to choose one.
	2b. System finds that the requested meeting does not exist.
	3b. System returns an error to the user.
	2c. System finds that the user does not have the required credentials to view the requested meeting.
	3c. System returns an error to the user.

Frequency of occurrence:	Occasionally, multiple times per session.

4.10 Start a meeting

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Meeting 2.1.6
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to start a meeting. The system saves the new meeting.

Basic flow	
Actor action:	**System responsibility**
1. User indicates she wants to create a new meeting.	2. System shows an empty form.
3. User supplies data for the initial post in the meeting (including attachments such as audio, video, documents).	
4. User indicates she wants to save the meeting.	5. System saves the new meeting.
Alternative flows	
	2a. System finds that the user does not have the required credentials to create a new meeting.
	3a. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.
Notes:	The term 'form' used in step 2 can be implemented in different ways, it does not imply an actual graphical form.

4.11 Respond to a meeting

Primary actor:	Programmers, code reviewers, designers
Stakeholders:	Users
Requirements:	Meeting 2.1.6
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to respond to a meeting. The system saves the response to the meeting.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific meeting.	2. System shows the requested meeting.
3. User supplies data for the response (including attachments and the like).	
4. User indicates she wants to save the response.	5. System saves the response with a reference to the original meeting.
Alternative flows	
	2a. System finds that the user does not have the required credentials to respond to a meeting.
	3a. System returns an error to the user.

Frequency of occurrence:	Occasionally, multiple times per session.

4.12 Perform an export

Primary actor:	Programmers, code reviewers, designers, reseachers
Stakeholders:	Users
Requirements:	Export 2.1.7
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to perform an export based upon an existing export plan. The system executes the export plan and returns the resulting files to the user.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific export plan.	2. System executes the export plan.
	3. System returns the resulting file to the user
Alternative flows	
	2a. System finds that the user does not have the required credentials to execute the requested export plan.
	3a. System returns an error to the user.
	2b. System finds that the export plan contains an error.
	3b. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.
Technical notes:	An 'export plan' is similar to an ANT build file, but need not be implemented as such.

4.13 Create an export plan

Primary actor:	Programmers, code reviewers, designers, reseachers
Stakeholders:	Users
Requirements:	Export 2.1.7
Preconditions:	User is authenticated by UC 4.1
Description:	A user wants to create a new export plan. The system stores the created plan for later use.

Basic flow	
Actor action:	**System responsibility**
1. User requests a specific export plan.	2. System shows the requested export plan.
3. User modifies the export plan.	
4. User indicates she wants to save the export plan.	4. System stores the export plan for later use.
Alternative flows	
1a. User indicates she wants to create a new export plan.	2a. System shows an empty export plan. Continue with step 3.
	2b. System finds that the user does not have the required credentials to create an export plan.
	3b. System returns an error to the user.

Frequency of occurrence:	Occasionally, once or a few times per session.

4.14 Manage users

Primary actor:	Managers
Stakeholders:	Managers, Users
Requirements:	User management 2.1.9
Preconditions:	User is authenticated by UC 4.1
Description:	A manager wants to manage (that is create, read, update or delete) a user account. The system stores the modifications made by the manager.

Basic flow	
Actor action:	**System responsibility**
1. Manager requests a specific user account.	2. System shows the requested user account.
3. Manager makes modifications to the user account.	
4. Manager indicates he wants to save the user account.	5. System stores the user account. Changes take immediate effect.
Alternative flows	
1a. Manager requested a list of user accounts.	2a. System shows a list of user accounts, allowing the manager to choose one.
1b. Manager indicates he wants to create a new user account.	2b. System shows an empty user account. Continue with step 3.
	2c. System finds that the requested user account does not exist.
	3c. System returns an error to the mana ger.
3d. Manager indicates he wants to delete the user account.	4d. System deletes the user account. Changes take immediate effect.

Frequency of occurrence:	Occasionally, once or a few times per session.

4.15 Update permissions

Primary actor:	Managers
Stakeholders:	Managers, Users
Requirements:	User management 2.1.9
Preconditions:	User is authenticated by UC 4.1
Description:	A manager wants to modify the permissions of a specific user of a group of users. The system stores the modifications to user permissions.

Basic flow	
Actor action:	**System responsibility**
1. Manager requests the permissions a specific user account.	2. System shows the permissions of the requested user account.
3. Manager makes modifications to the set of permissions.	
4. Manager indicates he wants to save the set of permissions.	5. System stores the set of permissions. Changes take immediate effect.
Alternative flows	
1a. Manager requested the permissions for a group of users.	2a. System shows the permissions for the group of users, allowing the manager to choose one.
	2c. System finds that the requested user account does not exist.
	3c. System returns an error to the manager.

Frequency of occurrence:	Occasionally, once or a few times per session.

Chapter 5

Logical View

The logical view represents the functional requirements. It addresses the following stakeholders:

- Managers
- End users
- Developers

It addresses the following concerns:

- Does the design address the intended purpose of the system, i.e. does it conform to the functional requirements?
- How should the code be separated into classes during development of the system?
- Where should new functionality be added, and what parts of the system will be affected?

First, the concepts and terms used in this view are described. Then, a model of these concepts, shown as classes, with their attributes and relationships are presented.

5.1 Concepts

This section describes the concepts used in the logical view. Most of these concepts are directly taken from the domain of the distributed collaboration environment as described in the requirements document. They are described here to make sure their definitions are clear.

5.1.1 Artefact

An artefact is anything that is created by the users of the system and is part of the software project that is being developed using the system. As such, it includes code (either chunks or files) and documentation, but also other files stored as part of the project. Artefacts have a type, and a URI (Unique Resource Identifier) that can be used to uniquely refer to this artefact. They always belong to a group.

5.1.2 Files

Everything in the project is stored in files. Files are artefacts. They may contain chunks, but can also contain only unchunked content.

5.1.3 Chunks

A file may consist of several chunks. A chunk is an artefact, and has an optional name, as well as a variant number.

5.1.4 Export rules

An export rule describes how an export of a source distribution or documentation should be performed. It is linked to the files it needs. It does not generalize to file, since it is not chunked. This is because the selecting of chunks is part of creating a source distribution.

5.1.5 Types

Artefacts have a type. A type determines how an artefact is rendered, as well as the syntax for determining links to other artefacts.

5.1.6 Artefact groups

Artefacts belong to groups of artefacts, which are used to manage permissions. An artefact group can also be described by documentation.

5.1.7 Document

A document is a piece of documentation describing an artefact or group of artefacts. It is itself an artefact.

5.1.8 Property

An artefact has many properties (e.g. number of lines) which may be used to generate reports. The types of properties available cannot be changed by the users, but the reports generated based on them can.

5.1.9 Users

The system is used by users. These users have permissions for using artefact groups and tools. All additions, modification and deletions they perform on an artefact are also stored as historic data.

5.1.10 User groups

Users can belong to a group. These groups can be used to manage permissions more easily. A group also has permissions on artefact groups and tools.

5.1.11 Tools

A user or user group can be granted permission to use a tool. The generic tool is further specialized into the actual tools for browsing, editing, analysis, exporting, bug or feature tracking, holding a meeting, managing users or extending the system with a new type.

5.1.12 Issue

An issue can be either a bug or a feature. It contains one or more comments by users of the system, and is related to artefacts.

5.1.13 Comment

An issue contains one or more comments. Comments are linked with the user who wrote the comment.

5.2 Relationships

The classes described in section 5.1 are related to each other. These relations are depicted graphically in figure 5.1. Starting at an artefact, we see that an artefact has one type. As files and chunks are artefacts, these are also

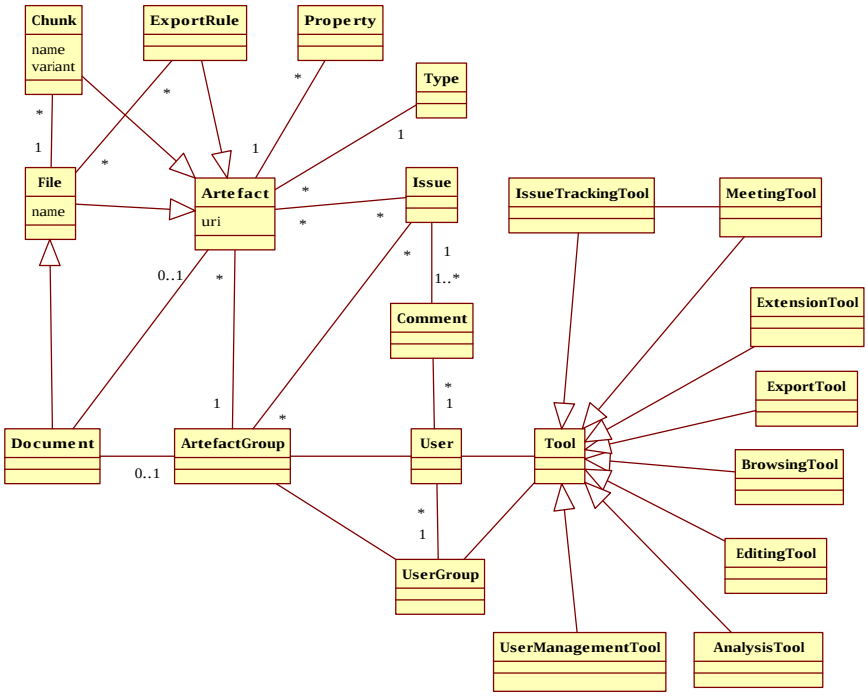

Figure 5.1: A class diagram of the entities in the E^4 system.

typed, and the type of a chunk is the same as that of its file. A chunk is part of exactly one file, and a file can contain zero or more chunks.

A document is a file, and thus also has a type. It described either one artefact, or a group of artefacts, or isn't linked and describes something more general.

Each artefact belongs to a group of artefacts. Artefacts that have not been explicitly added to a group yet could be added to a 'root' group. A direct mapping from a subversion folder to a group is also possible.

A user can belong to a group of users. This group can again belong to a group, to create a hierarchy of groups. The link between user group and users on the one hand, and artefact groups on the other, signifies a permission to use or modify the artefacts in that group. If a link from a user, a group the user belongs to, or any group higher in the hierarchy, has a permission to read a file, the user can read that artefact. Modifying works similarly.

The same permission relationship exists between users and groups on one side, and tools on the other. These links define a 'can use' relationship. If a link between a user (or any group it belongs to) and a tool exists, it means that user can use the tool.

The issue tracker is used to manage bugs and features, represented as an

issue. An issue contains one or more comments describing the issue and any progress made. A comment is written by a user. An issue is concerned with artefacts, but can also be about a new feature, for which no files exist yet.

All tools inherit from a base tool class, which handles things like permissions. Links from tools to the objects they affect are not shown, because otherwise the diagram would become unclear. These links would be rather straightforward: the issue tracker tool has access to issues, the issue tracking, browsing, editing, and export tools to artefact groups, the user management tool to users and user groups, the extension tool to types, the analysis tool to artefact properties, the export tool to export rules. As shown, the meeting tool can also use the issue tracking tool to easily share issues with meeting participants.

Chapter 6

Development View

The development viewpoint is a viewpoint for representing the static organization of the software. The development viewpoint illustrates the system from a software modules and subsystems perspective.

It addresses the following stakeholders:

- System administrators
- Developers

It addresses the following concerns:

- Can the components of the system be implemented separately before being related?
- Can the components of the system be reusable?
- Can each component of the system be easily tested? Then the whole system?
- Can the system be easily modified after being deployed?

6.1 Development Architecture

Figure 6.1 shows how the development effort will be divided into three layers. These three layers are defined as follows:

Data Storage layer Deals with basic files. Every artefacts and chunks is physically stored on one or several devices. A SVN repository application and a database allow the upper layer to access the data. SVN

(3) *User interfaces*	Web-browsing client, integrated platform.
(2) *Coding services*	Coding tools: code editing, code analysis, code browsing, code documenting, exporting. Collaborative work tools: issue tracking (including bug tracking and feature request), meeting tool
(1) *Data storage*	SVN repository, database.
Hardware	

Figure 6.1: Development architecture divided into three layers

repository allows users to check out the data, make and commit any changes. Database guarantees a faster access to the data for analysis tools. Consistency between these two must be addressed. "Modifications broadcasting" is a function permitting the information to be always up-to-date after any change.

Coding services Can bee seen as the engine of the application. Indeed, here are executed every functionalities which modify data contained in the "Data Storage layer". These functionalities are ordered by the users from the upper layer.

User interfaces The module that the users directly deal with. Interfaces allow the users to call the functionalities to execute within the "Coding services" engine. Besides, thanks to the broadcasting mechanism, the users screens are refresh every time it is necessary.

6.2 Development team and costs

A GANTT diagram helps the developing team to evaluate the amount of work required for each step of the implementation. Since the software development is divided into independent components, it is possible to work simultaneously on different phases, as shown in figure 6.2. This diagram takes into account that the developing team is composed by 10 persons.

The following table shows an overview of the development team, based on human resources needs. We consider a 10-person development team, composed by computing engineers according to the way shown in figure 6.3. Here we make the estimation that the project lasts 60 working days. Needs of each specialist in terms of working days is detailed. The "Total cost"

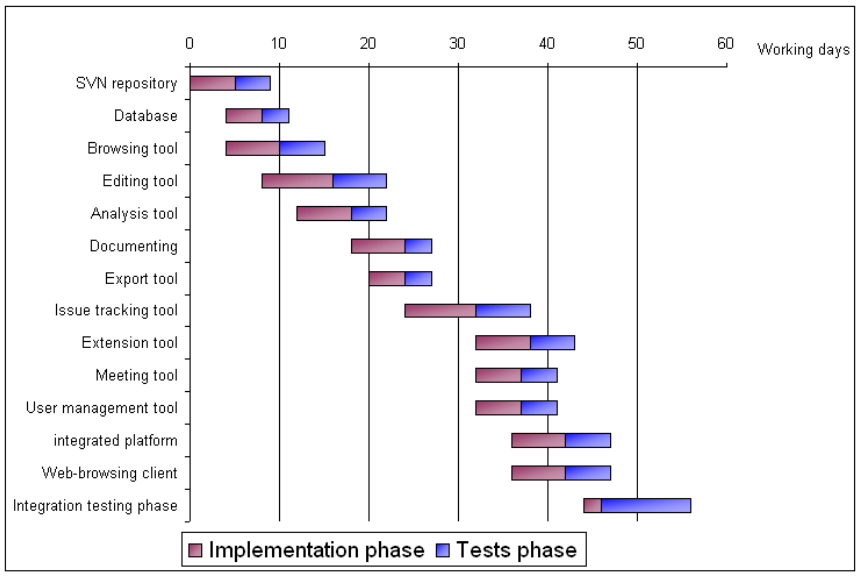

Figure 6.2: Gantt diagram of the development phase

is calculated by multiplying the salary per day (taken from Apec [1]) and the number of effective working days for each job. Regarding to the current salaries of the specialists in place within the team, we can make an estimation of the cost of the project in terms of human resources. This labor cost is evaluated to 49,000. This does not include taxes payable by the employer, since these differ according to country. However, these often amount to about 50% of the labour cost. All prices are in euros.

Job	*Salary/day*	*Quantity*	*Total days*	*Total cost*
Project leader	140	1x	60	8,400
Database specialist	110	1x	20	2,200
Networks specialist	110	2x	80	8,800
Systems specialist	110	2x	80	8,800
Programmers	100	4x	208	20,800
total				49,000.00

Figure 6.3: Project team and costs

Chapter 7

Process View

This chapter describes the dynamic behavior of the system on an architectural level. It describes the main processes of the system and how they interact with each other. Also non-functional requirements like scalability, availability and maintainability will be covered. The view will be discussed at two levels, at an overview level and a more deeper view of the different layers. At the end of the chapter there is a motivation of the design decisions we made (rationale).

7.1 Overview

The E^4 system consists of 4 conceptual layers (tiers). Each containing its own processes. A tier can only interact with the layer above or below him. The conceptual layers that are considered in this view are:

- **Data tier**: Stores and queries all data for the business logic tier.
- **Business logic**: The processes that cover the business logic of the application. It needs to provide notion of the code structure (chunks, types, files), provide access control, etc. as described in the logical view (chapter 5).
- **Web tier**: A number of processes (services) that offer different interfaces to the data. This includes a subversion service, a web service (http/ https), and a meeting service. The number of services can be extended to provide more facilities.
- **Client applications**: The web browser and all tools on the client side. Client applications are able to access and modify data through the provided services.

7.2 Tiers

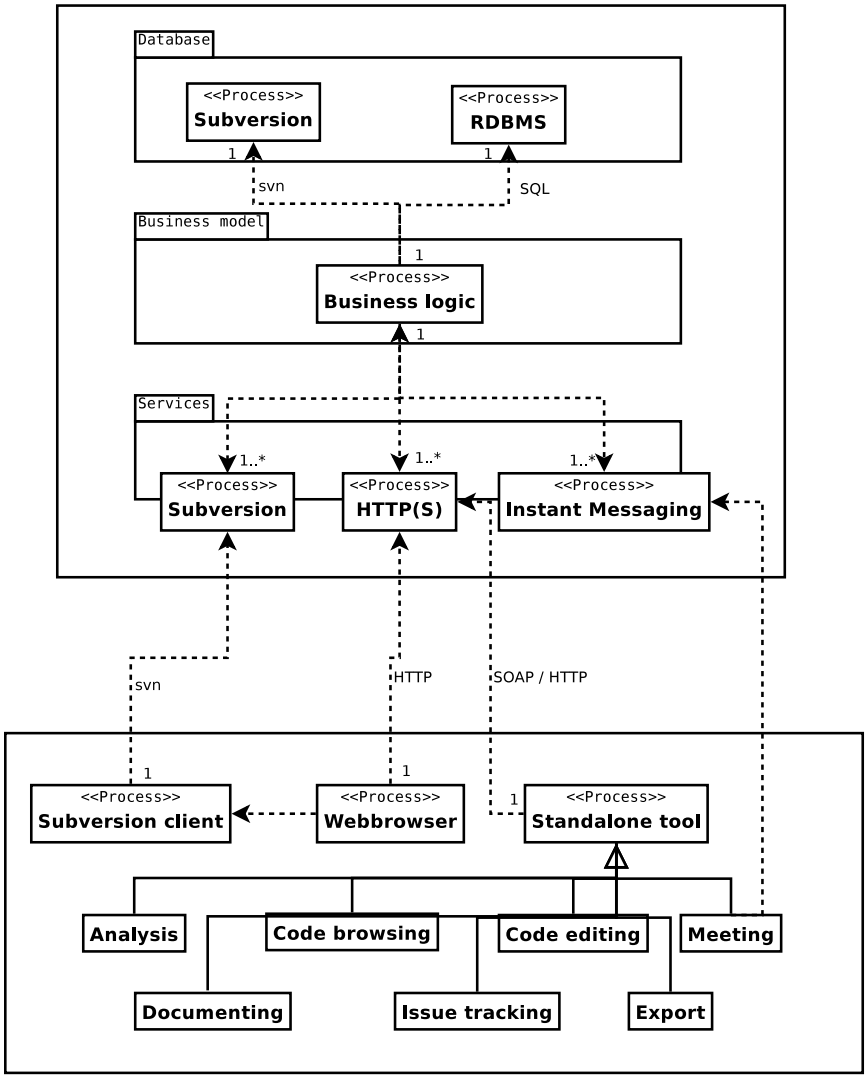

Figure 7.1: The system's processes per tier

7.2.1 Services

A service provides an interface to a client process. They are accessible through the TCP/IP protocol to make them accessible over the internet. All services have fixed TCP-port numbers and their addresses are found using fixed names (Domain Name Service).

The available services are:

- **Subversion**: This service process is a facade that redirects its calls to the subversion data back end. To improve performance, caching techniques might be implemented in this service, but at the current expected number of users this strategy is not needed. More than one subversion service is possible since they all share the same back end. In such situation caching facilities will be required.

- **HTTP/ HTTPS**: The HTTP service provides an interface to web browsers. Besides serving web pages, it also handles calls from the standalone tools on the client side through SOAP over HTTP. E.g. adding a new analysis in the analysis tool will cause a SOAP-call to be sent to the HTTP server. An advantage of this strategy is that the same functions can be invoked from the browser and from the tools. Another advantage is that it solves some issues when users are working behind a strict firewall.

  Functions on the HTTP server can be implemented using JSP in a J2EE environment or ASP.Net in a .Net environment.

- **Instant Messaging**: All instant messaging features are provided by a third service. This can be implemented using off-the-shelf solutions like Jabber or an IRC-server (Internet Relay Chat).

7.2.2 Business logic

The business logic models the logical scheme as defined in the logical view of this document. From process point of view, the business logic should provide at least the following functionality:

- Implement a logical scheme according to the logical view.
- Provides interfaces for the different services.
- Handles communication with the data storage.
- Should contain an active object that triggers generation of time-dependent actions, such as analysis-reports and builds.

For example, the business logic could be implemented as Enterprise Java Beans (EJB) in a J2EE environment.

7.2.3 Data tier

The data tier provides two storage facilities:

- The subversion system provides version control facilities. It contains all source files and is accessible through the svn protocol.
- A relational database management system (RDBMS) provides storage facilities for all other data, e.g. user data, analysis results, etc. The RDBMS provides a SQL interface.

Both facilities are singleton (only one server).

7.3 Rationale

All processes communicate through network protocols. This allows us to run all server-side processes on a single machine, or distributed over multiple machines. This strategy provides scalability and therefore can scale the performance in cases of increasing computational load. For the top-level protocols, we use common protocols as much as possible (e.g. HTTP, SQL, SVN, SOAP), increasing the accessibility of the provided interfaces.

The architecture provides services for different tools. Adding new functionality to the server will involve the extending of existing services or creating new services, and possibly modifying the business logic and storage structure.

Extending the capacity of the system can be done by duplicating services. This causes no real consistency problems. The subversion service gets its data from a central repository. Most instant messaging services already provide distribution mechanisms. The HTTP service needs synchronization, but since the data on the service is of more static nature it is sufficient to use standard file synchronization tools like *rsync*, and there is no need to model this explicitly.

Chapter 8

Physical View

The main goal of the physical architecture is to account the non-functional requirements of the system, such as availability, reliability, performance and scalability. The software described in this document executes on a infrastructure of computer systems. These individual systems are sometimes called nodes. The various elements of the architecture like processes, tasks and object needs to be mapped on the various nodes. Sometimes different configurations will be used for different goals. For example sometimes there exists a debugging environment next to the production environment for first testing and debugging new features before those features are implemented in the real system. The mapping of the software on the nodes needs to be highly flexible and have minimal impact on the source code itself to make it possible to run om different physical layouts. Besides the physical layout, a estimation of the total hardware costs will be given. This viewpoint targets different stakeholders, but mainly the system administrator who has to deploy the system, and the manager who has to have an overview of the costs.

8.1 Rationale

There are two types of clients connecting to the E^4 system. Web Clients and GUI Clients. Web Clients are clients that use the web interface to the tools, and GUI clients are clients who use the native GUI applications for the tools. These two types of clients can connect to the E^4 system from within the intranet of from the internet. The clients from the intranet are directly connected to the main switch. The clients from the internet are connected through a firewall. The source code repository is also connected to the main router. The domain logic web server presents some web services for accessing the application data. The application data includes all data needed by the coding tools, like code analysis, and user account information, etc. Access to

the data is only possible through these web services. The domain logic server is connected with the databases servers through a switch. This is needed for scaling with more servers, this is explained later in this chapter. The GUI Clients directly connect to the domain logic web server. The Web Clients use the web front end provided by the web interface. The main switch is connected with the web interface server through a load balancer. This leave us some scaling possibilities, this is explained later in this chapter. A backup server is located on another physical location. This server is accessed through the internet.

8.2 Quality

Important qualities addressed by the Physical View are scalability, performance, reliability and availability. These qualities will be described in the next sections.

8.2.1 Availability

Availability describes the amount of time the product is available to the user at the time it is needed.

Load balancer

The load balancer divides the requests to the system over several attached servers. When a server fails, the load balancer excludes this system, and just divides the requests over the remaining systems. Also a message is sent to the administrator so he can immediately try to solve the problem. The users of the system will never notice that one server has failed.

When using load balancers the load balancer itself becomes a critical part, because when the load balancer fails all the servers attached to the load balancer are unavailable. Therefore load balancers are often deployed as a clustered pair of two load balancers. If one fails, the other one just takes over the job.

Uninterruptible power supply

An Uninterruptible power supply (UPS) is a device which maintains a continuous supply of electric power to attached hardware when utility power is not available. When utility power fails it starts to provide power from batteries. But only a limited amount of power is available. When the utility power is unavailable for long time and also the UPS is running low on power, a message is sent to the server to be able to shutdown properly.

8.2.2 Reliability

Reliability is the capacity of software to maintain its level of performance under stated conditions for a stated period of time.

RAID

RAID or Redundant Arrays of Independent Disks is an umbrella term for methods to increase performance or reliability of a set of hard disks. This is done by mirroring, parallelism or some kind of parity checks. An often used RAID level is RAID-5. RAID-5 provides redundancy but with a low cost of extra disk space. Using RAID increases the reliability because the system is able to recover from a hard disk crash.

Backups

RAID protects the system against hard disk failure, but when the whole system is damaged by for example a fire, the data is still lost. Backups of critical system configuration and user data to a external server will manage this problem.

Firewall

A firewall protects the system from external attacks. This prevents the servers from being hacked. When a server is hacked it is not possible to guarantee the system can provide the service as expected.

8.2.3 Scalability

Scalability is a desirable property of a system or network, which indicates its ability to either handle growing amounts of work in a graceful manner, or to be readily enlarged.

Users

The system is designed to be extendable when the total amount of users is increasing. The system can be implemented using one single server, or using a lot of servers, depending on the amount of users. The actual implementation is explained in the Implementation chapter.

Switches

There network contains two switches. Another switch can easily be attached to increase the capacity. The Sub switch is only needed for eventually adding more database servers. This is a simple switch and this should be enough to set up a simple database cluster.

8.2.4 Performance

Computer performance is measured by the amount of useful work accomplished by a computer system compared to the time and resources used.

Load balancer

The load balancer increases performance by dividing work over several servers.

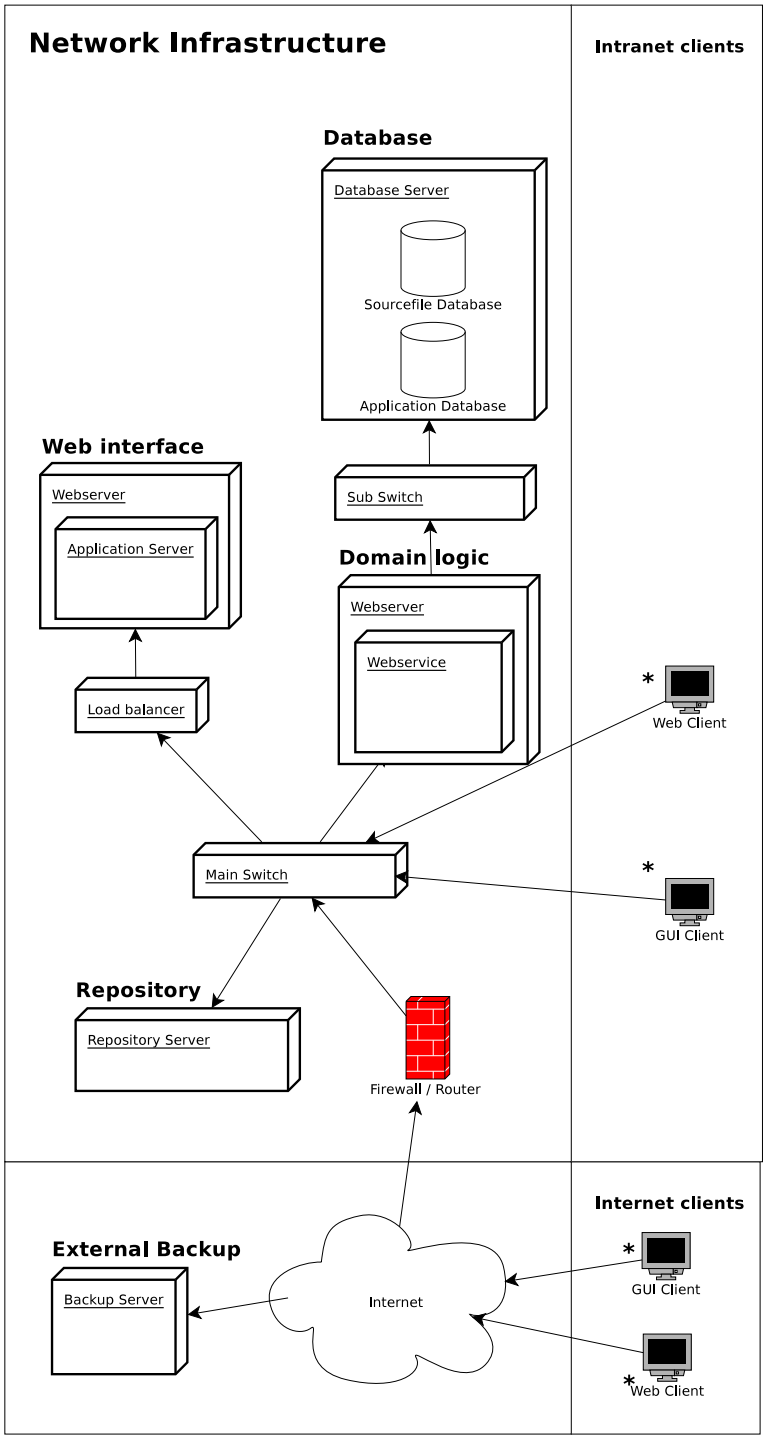

Figure 8.1: Model

Chapter 9

Implementation View

This chapter describes how the physical view is going to be implemented. This chapter is also a proof that our software architecture can be implemented. The decisions about hardware and software choices are explained.

9.1 Rationale

Looking at the previous architecture learned us that now there are only 15 subversion committers to the project. The system described in this chapter is intended to be for usage by 25 simultaneous users. A single server should be enough for that many users. These users can be all kind of stakeholders, for example, programmers, code reviewers, designers, etc. The system is designed in a way that it can easily grow up to a few hundred simultaneous users without any hardware changes. How big these numbers really are need some further benchmarking, but this is an estimate. But also further growth is possible. It is possible to add more hardware like described in the physical view to even support more users.

9.2 Hardware

This is a list of hardware needed to build the system.

9.2.1 Application Server

The application server is a powerful server. This server runs the database server, the repository server and the web servers.

Name: PowerEdge PE2900

Processor: 2 times a Quad Core Intel Xeon E5420

Memory: 8 GB memory

RAID Controller: RAID 5 controller

Hard disks: 3 x 73GB SAS hard disks 15.000 rpm hot-pluggable

Price: 3200 euro

9.2.2 Firewall

Defends incoming and outgoing traffic against worms, spyware, trojans, malware and hackers with a complete set of Universal Thread Management abilities.

Name: Juniper Networks Secure Services Gateway 20

Price: 800 euro

9.2.3 Main switch

Name: Dell PowerConnectTM 6248

Network: 48-ports 10/100 Fast Ethernet-switch

Price: 2400 euro

9.2.4 Backup server

The backup server is a light server but needs a lot of reliable disk space, so this server needs a RAID 5 array with Serial Attached SCSI disks.

Name: Dell Storage Server

Processor: Dual Core Intel Xeon

Memory: 1 GB memory

RAID Controller: RAID 5 controller

Hard disks: 3 x 73GB SAS hard disks 15.000 rpm hot-pluggable

Price: 2100 euro

9.2.5 Uninterruptible Power Supply (UPS)

Name: Dell 5000i

Capacity: 6 power connectors

Power: 3750 Watt

Price: 2600 euro

9.2.6 Hardware costs

This is an overview of all hardware costs. All prices are in euros and exclusive VAT.

Name	*Quantity*	*UnitPrice*	*TotalPrice*
Application Server	1x	3,200	3,200
Firewall	1x	3,200	800
Main switch	1x	2,400	2,400
Backup Server	1x	2,100	2,100
UPS	1x	2,600	2,600
total			11,100.00

9.3 Software

Operating system

The operating system of choice for all servers is Linux. This is a free operating system and often used for servers. It is modular and you can simple only install the software you need. Linux is also extensible because it is an open source operating system. This OS has proved being very stable and secure. The system has initially been developed as a operating system for network connectivity.

Software Development Framework

For implementing the web interface and web services the J2EE framework is used. This framework is developed for making web applications. It is used in very large projects and has gained a large market share. Also a lot of programmers have experience in programming java.

Version control

The repository server software subversion is used for version control. This software was already used in the old situation. Subversion is an improved version of CSV. It has improved some problems that where involved with CVS. Subversion is also prepared for use within a network.

Database

The system uses a MySQL database for persistency. MySQL is a fast lightweight open source database server. MySQL has basic features but this should be enough for this system.

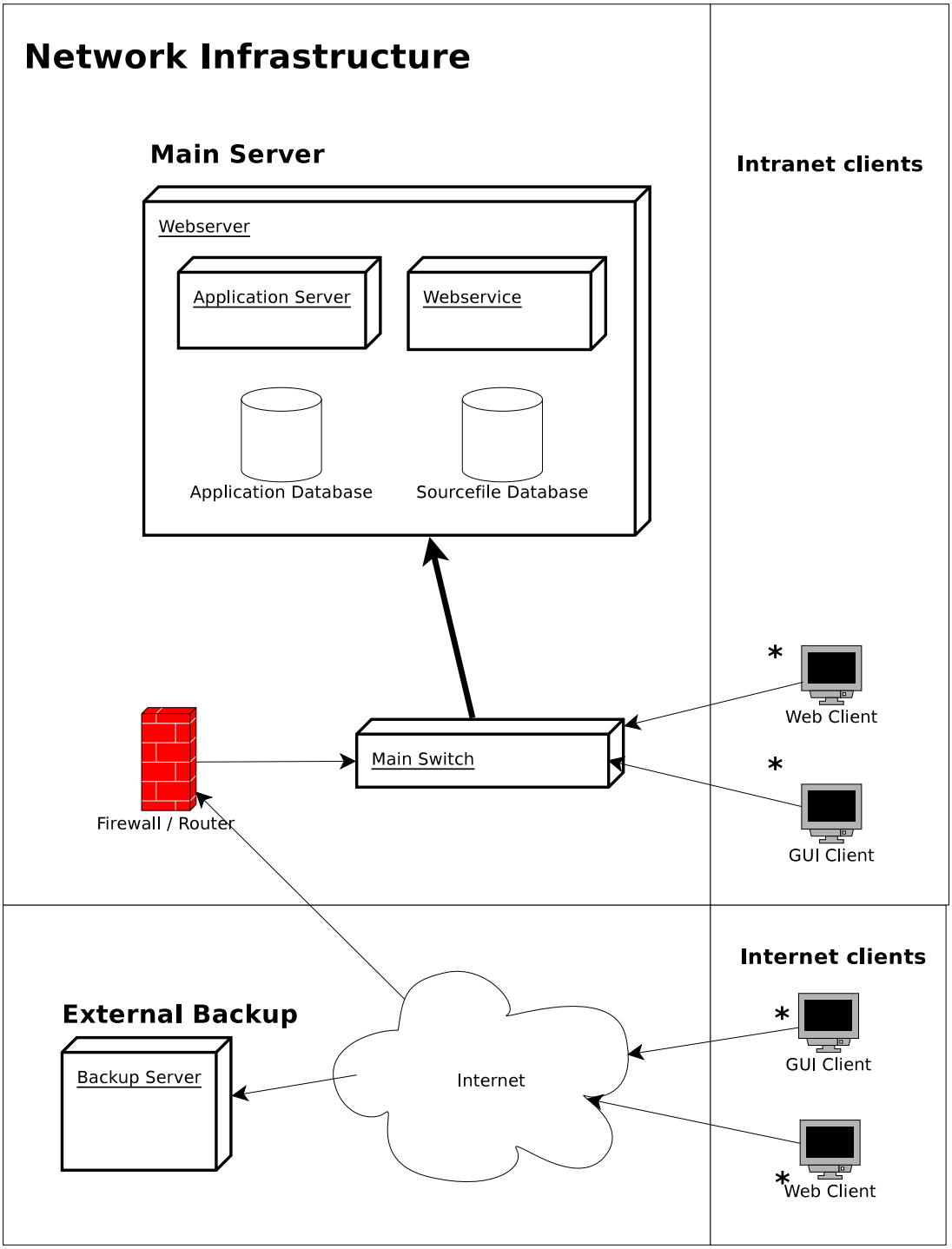

Figure 9.1: Model of Implementation

Chapter 10

Security View

The security features of E^4 needs to prevent the use of different functionality of the system by unauthorized users. Since E^4 is an open software system, the main focus is to prevent unauthorized users from modifying data. Viewing data is of less concern, but nevertheless taken into account.

10.1 Authentication and authorization

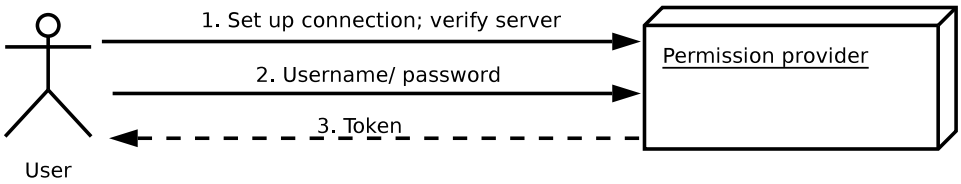

Figure 10.1: Authentication procedure

The user can login using a standalone tool, or by using a web browser. Both login procedures are identical. The user permissions are provided and managed on the server. Login information is sent as a SOAP message over HTTPS (secure HTTP). The server will then return a token that identifies the user during the current session. A token is attached to a specific IP-address and will expire after a specified time of inactivity or when a logout occurs. On every call to the server, the token is included by the client and checked by the server.

The checking and providing of the credentials is done by the *permission provider* and privileges are stored in a separate *permission database*. The authorization scheme has to be modeled as described in the logical view. I.e. users are part of a user group; permissions are assigned to groups or

individual users. The permission provider component has to be implemented at business logic level.

See also use case on chapter 4.1 of this document: "login to the E^4 system".

10.2 User management

Users are managed by the managers (see "Stakeholders and concerns" chapter 3). They can add and remove users or assign permissions to users or user-groups. The manager can edit users by using the user management tool.

On every tool or artefact a number of permissions are defined, they can be allowed or disallowed by the permission provider. A user can have usage permission per tool, or read and modify permissions on artefacts. For example, a user can have usage permission for the analysis tool, but no read permission on a particular artefact. Administration permissions allow a user to assign permissions on that tool or artefact to other users. Managers have administration permissions on all tools and artefacts. A tool can have additional tool-specific permissions as well.

See also use case on chapter 4.14 ("manage users") and 4.15 ("update permissions").

10.3 Logging

Since every action is controlled by the permission provider, it also saves a log of actions. A log record contains data such as the time, user information and action performed. In case of raised suspicion, a manager can check the logs to see if there was authorized access that should not be authorized. Consequently he/she can take action by modifying permissions to avoid these situations in the future.

10.4 Hardware security

On a lower level, our infrastructure is protected by a firewall filtering out all unwanted network traffic to our system. This can increase availability of the system since it is less vulnerable to generic attacks.

The permission provider and permission database are of importance in the system. To improve the security of the system, it could be physically separated from the system. I.e. create a separate permission server containing the permission provider and the database. This way it is easier to ensure the

safety of the system since the communication from and to the permission server are less complex.

Chapter 11

Conclusion

In this document, we have described an architecture for the E^4 system, a system to collaborate on large software projects, in particular the Essential Haskell Compiler. We have listed the requirements of the system, and presented the stakeholders. Use cases were described to show the proposed interaction with the system. The concepts of the system were presented, and the relationships between them described. These things show that the proposed system will meet the functional requirements.

The processes of the system were described, as well as the hardware the system will run on. The system will support 25 users, but allows many opportunities for scalability. The use of a web-based as well as a native interface provides portability and worldwide flexible access for those who need it, and an integrated work experience for regular developers at a fixed location. Additionally, as requested, a subversion repository will be offered for those who wish to use it.

An implementation suggestion in terms of actual hardware and manpower was also given, including the cost of implementation. The hardware cost of 11,000 is achieved by running most services on a single server; We feel that this will still provide the required responsiveness at the given number of users. Further scaling of the system to larger numbers of users is possible using the given architecture, but will increase the hardware cost. The development cost of 78,400 might be cut if standard software packages are used, but some compromises on the specific functionality will have to be made. The given development architecture will allow easy future modifications.

Furthermore, a view on the security of the system was given. Detailing both hardware and software security, it shows how this requirement was met. Testing during and after development can further improve confidence in the security of the system.

Given these various views of the system, as well as connections with real

hardware, cost estimates and technology, we feel that we have shown that building this system is feasible, and will meet the expectations of the client.

Bibliography

[1] Apec - French recruitment organisation. URL http://www.apec.fr/.

[2] Essential Haskell Compiler website. URL http://www.cs.uu.nl/wiki/Ehc/WebHome.

[3] The GNU Operating system website. URL http://www.gnu.org.

[4] PHP: Hypertext Preprocessor website. URL http://www.php.net.

[5] Subversion project website. URL http://subversion.tigris.org/.

[6] Unified Modeling Language website. URL http://www.uml.org/.

[7] Atze Dijkstra. Course Assignment - Architecture Project, 2007. URL http://www.cs.uu.nl/wiki/Swa/CourseAssignments.

[8] IEEE Architecture Working Group. IEEE Std 1471-2000, Recommended practice for architectural description of software-intensive systems. Technical report, IEEE, 2000.

[9] Philippe Kruchten. Architectural Blueprints—The "4+1" View Model of Software Architecture. *IEEE Software*, 12:42–50, November 1995.