UNIVERSITÄT DORTMUND

Fachbereich Informatik
Lehrstuhl für Software-Technologie

M E M O Nr. 127

Project Group Chairware Final Report

Markus Alvermann Martin Ernst Tamara Flatt
Urs Helmig Thorsten Langer Ingo Röpling,
Clemens Schäfer Nikolai Schreier Olga Shtern

Ursula Wellen Dirk Peters Volker Gruhn

August 2002

Internes Memorandum des
Lehrstuhls für Software-Technologie
Prof. Dr. Ernst-Erich Doberkat
Fachbereich Informatik
Universität Dortmund
Baroper Straße 301

D-44227 Dortmund

ISSN 0933-7725

Project Group Chairware

Final report

University of Dortmund

Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer, Ingo Röpling, Clemens Schäfer, Nikolai Schreier, Olga Shtern

Ursula Wellen, Dirk Peters, Volker Gruhn

Plus the Melbourne Chairware team

Stuart Birnie, Renee Bonnett, James Corr, Winston Fletcher, Jonty Goldin, Lorraine Johnston, Jamie Khoshaba, Adriano Migliore, Peter Reynolds, Jean-Guy Schneider, Grant Sheppard, David Wengier

Table of Contents

Table of Figures

Preface

The faculty of computer science of the University of Dortmund integrated a special kind of lecture into its courses of study, the so-called "Projektgruppen" (project groups).

The trained computer scientist is in his profession often confronted with questions

1. which (caused by the rapid development of the realm of computer science and the fast expansion and propagation of the range of use of this realm) are new to him in a little while after leaving the university in the sense that he cannot revert to the methods and techniques learned in his study. The computer scientist is often asked to comprehend the development of methods relevant to the problems by himself and to adapt and enhance the offered solutions respectively develop his own solutions on the base of his basic knowledge.
2. which processing (caused by the expected volume of work) demand coordinated work of many workers so that the work has to be organized as a team. Therefore the computer scientist has to divide up the concrete problem into parts, has to care for the solution of these problem parts and has to integrate the partial solutions to one complete solution.

Project groups target on teaching and exercising the skills described by 1. and 2..

They provide a setting in which

- Knowledge and techniques from (sometimes different) realms of computer science should be used on a concrete problem, so that the acquirement, adaptation, expansion and development of methods relevant to the given problem are practised,
- A problem of greater complexity should be handled, so that the process of becoming acquainted with the problem and the work in groups (organisation. leadership, coordination, cooperation, representation and procurement of the own ideas etc.) can be practised,
- The factual and organisational problems of the analysis, decomposition and treatment of considerable problems become clear and how they are managed.

A project group consists of one or more supervisors and 8-12 students. The supervisor (at least one of the supervisors) is professor or assistant to a professor of the faculty of computer science. The students are students in the second part of their study, i.e. the PG (project group) is part of their postgraduate programs of study. The project group works on a concrete and clearly described problem. In the face of motivating the members of the project group the problem should be realistic. Interdisciplinary problems are explicitly allowed but an external product or finishing deadline must not exist. A project group spans over two consecutive semesters. It consists of 14-16 hours per week per semester (2 semester a 6-8 hours per week). The factual workload of each member of the project group should be measured so that they can visit other courses parallel to the project group.

1 Introduction

This document is the final report of the project group 386, "Chairware".

This chapter introduces into the topic of conferences, their usual procedure and the resulting scope of work plus the constraints given for PG Chairware. An overview about the project team members and their responsibilities can be found in chapter 2.

As preliminary preparation, the group had a seminar phase that served as mean to get to know each participant as well as learning details about the topic of and the technology to be used for Chairware. Chapter 3 is a summary of this seminar including the abstracts of the lectures.

After having discussed the technical and political constraints of the project, the following chapters 4 to 11 tell about the approach to the final product. Thereby the proceeding in the analysation, design, implementation- and testig phase are described. Chapter 8 deals with the special reflections concerning the security aspects of the final product done by the security team. The authors tried to emphasize the trains of thought during development that lead to the decisions current now.

The description of the used technology is provided in chapter 12. It covers the configuration of the hardware as well as all operating systems and applications installed.

As this project is developed in cooperation with our Australian partners from Swinburne University of Technology, and since most of our documents are written in this language, too, we decided to write the report in English. Parts of this report were taken from documents provided by our counterpart in Melbourne. We would like to thank them for letting us use their work to get this report done.

After some consideration we decided to name our project Chairware. As the person responsible for the programme of a conference is called 'programme committee chair' (PCC) and because we want to produce a piece of software to support his work, the name Chairware was born.

1.1 Why an electronic submission tool?

The process of many of today's academic conferences is run following a certain pattern, which can be summarized as depicted by figure 1.

After installing the conference's data concerning location, date, personal and topic, a so-called 'Call for papers' takes place. That means authors from around the world are asked to write and submit a paper concerning the conference topics. These submitted papers are reviewed by an independent and impartial programme committee (PC) consisting of predefined researchers of that special field. There are always more than one reviews of one paper in order to guarantee a fair treatment for every submission. Having reviewed and given ratings to all submissions, the members of the programme committee (PC) meet and compare their review results. It is during this

meeting when the actual papers to be introduced during the conference are chosen according to their combined rating.

To increase efficiency and motivation during the 'submission of papers' and 'review' phases, a preliminary phase, the 'submission of abstracts', is used to fasten things a bit. After the call for papers, the authors that plan to participate submit an abstract of the paper they want to write. The paper does not have to exist at this point of time, it is needed at a later date. Using this abstract, the prospective reviewers (REVs) are allowed to give their preferences of what paper they would like to review. This could be a paper from topics a reviewer is especially interested as well as just competent in. The programme committee chair (PCC) is then allowed to already work on the different assignments, i.e. the reviewer-paper relations, while not every paper has been submitted yet.

After the actual acceptance of an author's paper, s/he is to submit a brushed-up version of the paper (camera ready copy, CRC) that will finally appear in the proceedings of the conference.

Nowadays it is common practise to use electronic submissions for conferences, workshops and journals. Usually, people are allowed to send their submission by email. Although, compared to sending hardcopies by surface mail, this is a big step forward, it may still be quite cumbersome to maintain the administration. Moreover, collecting, maintaining and comparing reviews by the programme committee chair may also become quite tedious, if not troublesome, due to the large amount of material, even when submitted by email.[20]

An Electronic Submission Tool (EST) should support this process.

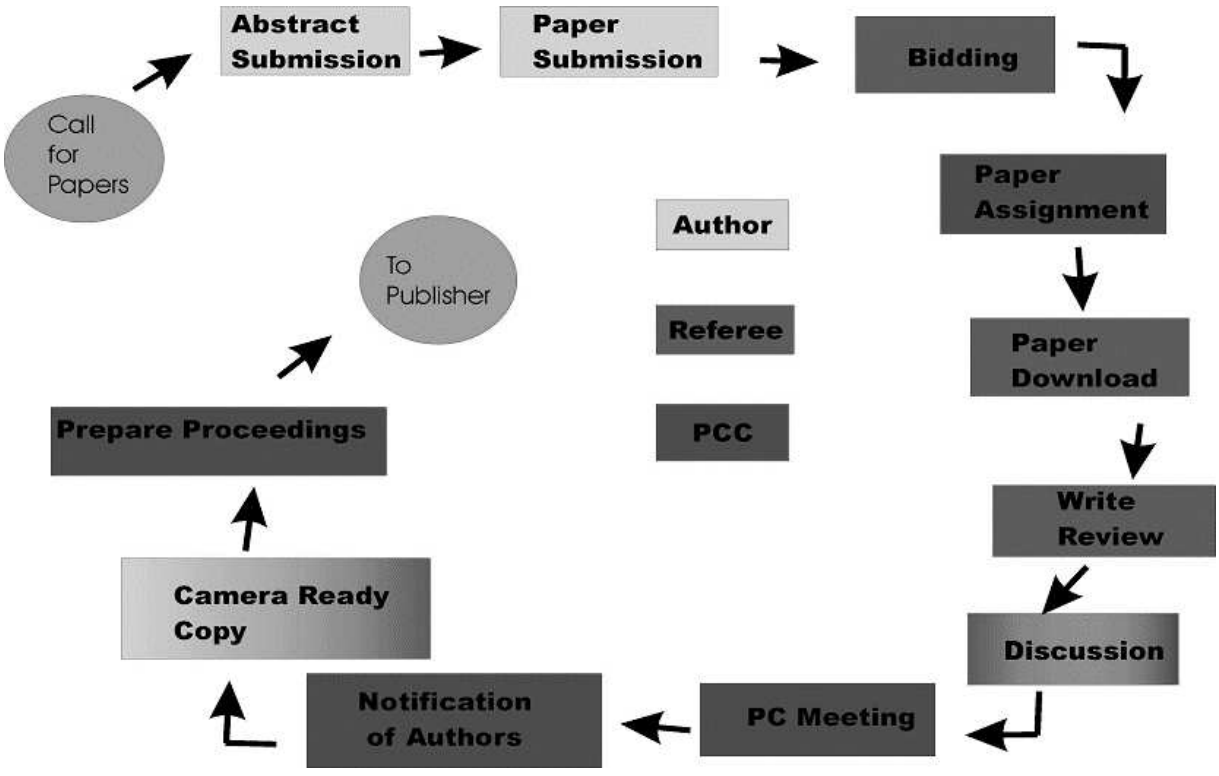


Figure 1 - Conference Phases of Chairware

1.2 The Scope of the Product

In most academic conferences the process from submitting the papers to the final review is managed electronically. So our electronic submission tool (EST) will have an interface to the programme committee (PC), to the

authors and to the reviewers. Therefore, the resulting application will have to be distributed, to run under different operating systems and, because of the essential privacy (for example an author should not be able to look up which reviewer will rate his contribution etc.) of the conference, security aspects will have to be considered in a special way.

A vague object model could look like Figure 2. The links between the objects represent simple associations:

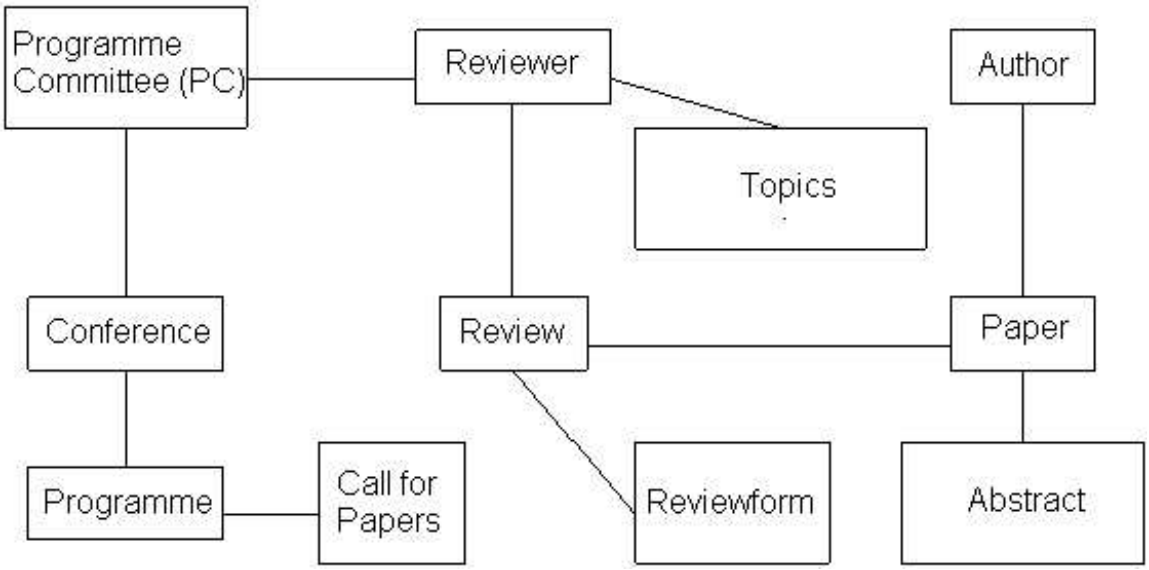


Figure 2 - Vague Object Model

The following requirements will be essential for the resulting tool:

- The objects shown in the diagram above will have to be registered, managed and requested.
- The user interface must be easily configurable by the PCC.
- There must be a possibility to set deadlines for submitting and reviewing the papers.
- The reviewers should be able to choose the papers they want to review from a list of abstracts.
- The PCC should be supported in assigning the papers to the reviewers. Personal interests and conflicts should be considered.
- The PCC must have the option to configure the given templates (review, e-mail etc.).
- The reviewers should be able to browse and print the abstracts.
- The reviewers should be able to write the reviews while being online.
- The status of the reviews has to be accessible by the PCC at any time.
- The reviewers will receive a note if the deadline for submitting the reviews is reached.
- If there is a major deviation between the ratings for one paper the PCC must be able to make a certain review accessible to other reviewers.
- Once a review has been submitted the reviewer should not be allowed to change it without permission of the PCC.
- There must be the option to expand the assemblage of the accessible evaluations easily.

1.3 Development constraints

In this section the constraints and special requirements of the development are described.

1.3.1 Component Based Development

The resulting application will have to be distributed, to run under different operating systems. The software should be cross-platform executable and installable and adaptable with a minimum of effort. As a consequence, the realization should be done with the component models Java Beans and Enterprise Java Beans.

1.3.2 Distributed Development

The realization of the submission tool is done in cooperation with Swinburne University of Technology, Melbourne. There are 10 students on each side at about the same level of experience who will have to cooperate in order to develop the application distributed and simultaneously.

This constraint again suggests the use of component-based development.

1.3.3 The Security Approach

When building e-business applications, assuring security constraints is essential for successful operation. Failure in this respect (i.e. vulnerability of sensitive data) would lead to customer dissatisfaction – in the best case. In our project group, we assumed to have the same situation, although, in reality, security is not that mission-critical.

The difficulty with enforcing security in large-scale IT systems is mainly the distribution of responsibility for the different components that make up those systems. Traditionally, security requirements are considered as fulfilled if the testing phase of the whole system is completed successfully. Unfortunately, testing is not a very comprehensive measure for assuring security so our intention was to take the responsibility from the developers and implement a more declarative security system, which performs the checks for security violations automatically in the background.

The security aspect should be realised using the concept of object-based protection that means the developer of the security component should be able to assign additional security attributes to certain classes that will force the application to obey to the given privacy policies during the runtime. To provide the object-based protection the Virtual Machine of Java should be modified.

1.4 Minimal goals of the project

The minimal goals of this project group given by the supervisors are

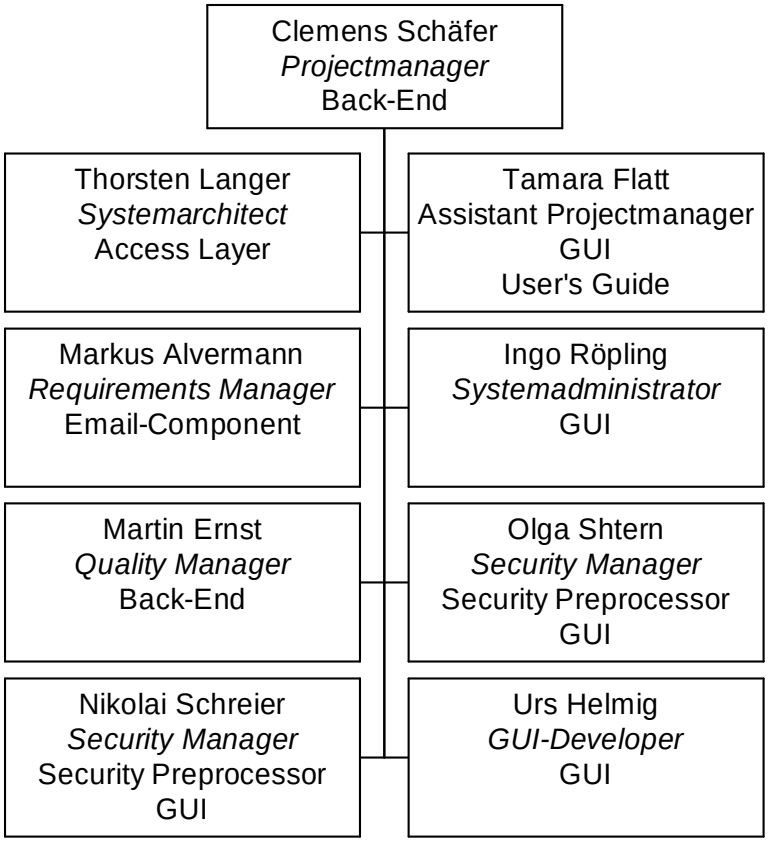
- Description of the development process,
- Comparison between existing submission-tools,
- Creating a requirements analysis for the submission tool,
- Realisation of a concept for the submission tool,
- Realisation of a prototype for the submission tool,

- Realisation of a prototype using the modified virtual machine to provide the object-based security aspects,
- Realisation of a security configuration for the submission tool.

2 Team Chairware

In the following the organisational structure of the project group is presented. It should provide a quick overview to the reader about the responsibilities taken by the different persons in their different roles. There also is an Australian team which isn't mentioned here. As you can see we chose a role based approach, where every team member gets a special field of duty where he has the control.

Structure of the project group Chairware



3 The Seminar in Nordhelle

During the seminar, each project team member held a lecture on topics that seemed to be important for the further progress of the project. This chapter provides a short summary of the different lectures.

3.1 Enterprise Java Beans

The Enterprise Java Beans specification defines an architecture for the development and deployment of transactional, distributed object applications-based, server-side software components. Organizations can build their own components or purchase components from third-party vendors. These server-side components, called enterprise beans, are distributed objects that are hosted in Enterprise Java Bean containers and provide remote services for clients distributed throughout the network. The specification mandates a programming model; that is, conventions or protocols and a set of classes and interfaces, which make up the EJB API. The EJB programming model provides bean developers and EJB server vendors with a set of contracts that defines a common platform for development. The goal of these contracts is to ensure portability across vendors while supporting a rich set of functionality. [8]

3.2 Application Server

An application server is, as the name points it out, a program with server functionality. It is part of the three-tier-architecture and forms the second tier or in other words, the business logic of the program. The intention of this lecture was to compare different application server to ease the decision about what product we should use for the project. One important aspect was that the application server had to be freeware, could deal with EJB technology and could run under Windows NT, which were some constraints of the project.

The candidates were: Orion Application Server, JBoss Application Server, Enhydra Enterprise, JonAS Application Server.

All of the products have their advantages and disadvantages.

The detailed tests (please refer to the elaboration of the lecture) showed that there were only two servers that would be useful for the project: Orion and JBoss.

Orion made it just because it supports the EJB 2.0 standard which provides message driven beans that could be quite useful.[2]

3.3 GUI Design Using Forte

The design of the graphical user interface (GUI) is a major aspect within the developing cycle of a software product. Since the GUI is the only part of the software the user gets in contact with, his impression about the quality of the software product is highly dependent on the quality of the GUI design. Beside aesthetic reasons, ergo-

nomic aspects require some thoughts about appropriate GUI design as well.

Within the lecture [3] some basic principles of the dialog design are shown. After a short overview about different style guides and their underlying requirements for ergonomic GUI design, one section shows the purpose of the AWT and Swings packages which provide GUI elements for Java applications. Next, the so-called layouts – the platform independent Java concept for placing dialog elements in different ways – are discussed.

With Forte an integrated development environment (IDE) for Java is discussed. It combines editor, compiler, debugger and GUI-builder in a single user interface. The lecture describes the key concepts of the Forte IDE to provide a decision support for the question which development tools should be used. A tiny demonstration - building a small Java application using Forte – clarifies the components of Forte and their use.

As German students have to work on an English GUI and text is an important design element which appears throughout the application in various components, one section of the lecture deals with the standards for the capitalization of English text. The two major capitalization patterns, Headline Capitalization and Sentence Capitalization, are illustrated within the lecture.

The development of the Electronic Submission Tool is done on two different locations at the same time. This means, that some additional aspects have to be taken into consideration to ensure one unique GUI approach. In its summary the lecture provides some hints and proposals for rules to be set up by the two sub teams to come to a common and unique GUI design.

3.4 UML with Together

This lecture [4] is an introduction of the unified modelling language (UML) and Together to design UML- diagrams.

In UML, eight different diagram types exist but not every type is used in our project. For example sequence diagrams and collaboration diagrams have nearly the same functionality. The only difference between them is the way of presenting the information. So it is not necessary to use both of them.

The most important diagram is the class diagram. Classes, methods including parameters, attributes with types and values, packages, and documentation of all these parts are included in it. Together creates the source code out of this diagram.

Sequence diagrams show the interaction between objects which is necessary to fulfil a task. Objects can be created in it and methods which interact between these objects. The method calls are ordered by time so that the exact course of a task can be seen. Together can insert the methods from a sequence diagram automatically into a class diagram.

Use cases show the interaction between a user and the system. This diagram type describes the functionality of the system. There is an actor in the diagram who communicates with the system. Visualisation of the interfaces between the user and the system can be seen here.

Out of the use case diagram activity diagrams can be created. Links between use cases and activity diagrams can be set. Activity diagrams show the control stream and the object stream between activities. This diagram type is used to describe a use case in detail.

Together is a UML modelling tool, which is written in Java. Every UML diagram type can be designed with it but not every fineness. It is a two-way tool; this means that source code can be created out of the class diagram and a class diagram out of existing code. Together supports parallel working. Different people can work on the same

project at the same time.

3.5 Access- & Information Flow Control

Since security is a special topic in our project, one person has been assigned the task to present some information about classic security models. The intent was (and still is) to incorporate some concepts into the application to remove the necessity to consider security in every component. Therefore the focus of the lecture is, after a basic introduction into what security is all about, set to Mandatory Access Control models and a model dealing with automatic enforcement of security constraints. This one was created to make security safer than what is possible with current technologies while having the same flexibilities than traditional Java security or manual programmed security mechanisms. Additionally, information flow should be considered to avoid the copy-information-and-pass-further security policy violations. Changes in the Java Virtual Machine seem to be required for enforcing the security checks and information flow consequences.[5]

3.6 Configurable Security in Java

The lecture "Configurable security in Java" [6] presents the security-related features of the JDK™ 1.0, 1.1 and 1.2 architecture. It starts with the description of possible security attacks and security requirements to the program. Following, it describes the evolution of the security model, beginning with the simple sandbox model in JDK 1.0, which provides just a very restricted environment to the applets and allows everything to the local applications. The extension of this model is provided in JDK 1.1 through the introduction of signed applets, which may become extended access to the resources. The JDK 1.2 introduces the new fine-grained, permission-based access control architecture via an easily configurable security policy and protection domains. The advantages and disadvantages of each model are discussed. Additionally, the lecture presents the security service JAAS (Java Authentication and Authorization Service). Finally, the problems of the Java security model are discussed and the examples of security attacks are presented.

3.7 BSCW/CVS

The seminar paper [7] deals with BSCW and CVS. It compares both tools and ends with a suggestion which tool should be used by the Chairware project group.

BSCW is an abbreviation for „Basic Support for Cooperative Work". It is software for supporting cooperative work of a distributed group of co-workers developed by the GMD (Gesellschaft für Mathematik und Datenverarbeitung). BSCW offers common workspaces where the team members can store, edit and exchange their common documents.

Why BSCW?

- Workgroups have their own workspaces accessible only by their team members.
- Work can be done on common documents.
- It is independent of the client's operating system. The clients only need a web browser
- Basic support for versioning of the common documents.

- Discussions like in newsgroups are possible.
- The Workspace can be divided and ordered by directories.
- Every action on documents (reading, editing etc.) is logged by the server.
- Different access rights for different persons are supported.

CVS is an abbreviation for „Concurrent Versioning System". It is a repository-based approach to version control. You can control every type of documents with CVS, i.e. not only source code which is the typical application for CVS. Every version put into the CVS-repository can be restored. It is possible to work with a nearly unlimited number of developers spread over the whole world. CVS applies a merge-process, which allows that all developers have mutual access to the common documents. Therefore parallel working on the same documents is possible.

Why CVS?

- Only changes are stored (saves hard disk space)
- Every developer creates his or her own workspace in which he can deliberately edit, delete and add files. The changes affect only the local copies until they are committed to the repository
- Different changes by different developers to the same files are merged automatically.
- Conflicts merging different versions are reported by the system.
- By checking out or updating his or her files the developer has access to the newest versions of the documents stored in the repository at any time.
- There are (graphical) interfaces to CVS on all major platform

BSCW supplies a good environment for synchronous communication and mutual time scheduling but the versioning system does not support conflict-solving-strategies.

CVS does not support communication between the developers besides the log entries but it supplies a mighty conflict-solving-strategy especially for text files and source code which often solves conflicts automatically.

The conflict-solving-process was considered more important than the synchronous communication by Tamara Flatt so she suggests to use CVS instead of BSCW.

The German project members followed the suggestion and decided to use CVS.

3.8 CyberChair

CyberChair is a completely web based groupware application and supports all phases in the whole reviewing and paper preparation process. It has successfully been used for several conferences in the past, e.g. the European Conference on Object-Orientated Programming 2000 (ECOOP).

For the authors of papers CyberChair supports the submission of their documents to the programme committee (PC). CyberChair supports reviewers by providing a common, easy to use, paper-free platform, that can be used to read or download, review and discuss documents. Moreover it supports the reviewers to get in contact with the authors. For the PCC (programme committee chair) CyberChair provides a convenient way to assign papers to

their reviewers, to plan the PC meeting and to ensure, that best papers are accepted for the conference.

CyberChair is implemented according to (most of) the patterns described by Oscar Nierstrasz' "Identify the Champion"[22]. This paper describes some patterns to optimise the review process.

Both CyberChair and „Identity the Champion" have big influence on the Chairware project. „Identify the Champion" is important because of its proven patterns and overviews to ensure a high quality product – in this case the right selection of papers. CyberChair influenced Chairware with its user interface and most of the terms of CyberChair are used in Chairware, too. [9]

3.9 Role based Software Development

This lecture [10] is an introduction to the concept of role based software development [19]. After explaining the overall concept and the arguments behind it, the 'classic' roles are presented including their responsibilities, needed qualifications, the used tools, methods, and techniques. These 'classic' roles are commonly used within today's software industry. Following this, there are proposals for new or modified roles suitable for project group 386, Chairware. With regard to our partners, Swinburne's choices of roles are presented as a comparison.

This lecture served as a baseline for the discussion of which roles should be used during project Chairware.

During a software development process, there is a close interaction between persons with different interests, goals, responsibilities. These interactions are numerous and vague. As a consequence, there are conflicts between the different jobs and participating persons.

The interaction is the decisive factor for success or failure of a software-development process. It is therefore highest priority to find a balance of interests between the participating roles and thus to enhance cooperation by understanding the role-induced views and problems.

'Classic' Role Descriptions

The following is an overview of some 'classic' roles as used in the software development industry.

- Project Manager

  Project-planning, staffing, organisation, project pursuit/examination/verification, guidance and motivation, reviews, reports to superiors

- Risk Manager

  Estimation of current risks, drawing up risk strategies, identification of counter measures

- Requirements Manager

  Initial understanding of essential requirements, forming useful groups of requirements, drawing up requirements document, promotion of discussion on document (user, developer), validation of requirements document

- Specification Manager

  Drawing up specification, introducing specification, adjustments of requirements and specifications, identification of test cases

- Designer (System Architect)

  Identification of components, breaking down components into modules, keeping track of decisions and decision-making, organization of reviews of design document

- System Administrator

  Check the integration of parts to target-platform, procurement and installation of needed hardware and software, scenario-like testing and documentation of test cases

- Developer/Programmer

  Coding of design, reuse of components, provide reusable components, test procedures and modules, measure own productivity, determination of effort- and advantage-characteristics

- Testing Manager

  Design and conduct all tests, test documentation, determination and notification of typical reasons for errors, assessment of results

- Quality Manager

  Creation of quality assurance plan, quality guidance and assurance, quality verification, coordination of internal and external quality assurance actions

- Maintenance Manager

  Stabilisation and correction, optimisation, adjustment, extension, change management

- Configuration Manager

  Creation/identification/control of configuration plan, coordination of the creation of release notes, version management, creation of a deliverable configuration

Additional Roles

- Security Manager

  Special security requirements of project, support of designers in their decisions

- GUI Developer (Useability Manager), role in Swinburne team

  Design of all user interface elements, implementation of GUI

- Technical Writer

  Creation of manuals (in a foreign language), testing of GUI

These roles and their descriptions were sent to Dortmund by the Swinburne team before our seminar in Nordhelle. They could therefore be considered when choosing our assignments. The Australian team named the following roles as important:

- Project Manager
- Requirements Manager
- System Architect
- System Administrator
- Testing Manager
- Quality Manager
- Development Manager
- Useability Manager

4 Quality issues

The quality standards of PG Chairware were assured by developing procedure-instructions on how a certain job has to be done. This includes naming a person responsible, a group performing, a template to be used, a deadline. In this way the quality was assured during the development and not solely by reviewing artefacts after their creation.

Most guidelines or instructions were developed during the semester and not before as the team learned from difficulties occurring.

4.1 The Meetings

The regular meetings of the Dortmund team took place two times a week. The meetings on Monday were held synchronously with the meeting of the Swinburne team (Dortmund at 8.00, Swinburne at 17.00).

A penalty catalogue was installed to assure an effective meeting and no unnecessary distractions to occur. Examples for these distractions are being too late, the ringing of mobile phones, and a delayed submission of the minutes.

4.1.1 Keeping the Minutes and the Action List

There has always been one person responsible for taking the minutes. This person, the scribe, has been selected in alphabetical order once per week. The scribe is responsible for taking the minutes, for updating the To-Do-list and for publishing both until Tuesday 19.00 cet or until Friday 12.00 cet depending on which session has been recorded. Additionally, the scribe is to put the minutes into CVS.

The minutes are taken in German since they only are of interest for the team Dortmund. There are templates derived from a preceding project group of the chair (IPSI) for both, minutes and To-Do-list, that proved to be useful and therefore have to be used by the scribe.

The To-Do-list includes so-called "actions" that were resolved by the team before a meeting. All actions have a number, description, date of the meeting in which the "action" was resolved, the deadline, one or more persons responsible for the "action", and a status. The numbers are increased sequentially for each "action". One session after the "action" was carried out, it is removed from the list to keep track.

This "action-tracking" is useful as no task will be forgotten and it is always clear who is responsible for what and when.

4.1.2 Moderator/Chair

Every meeting has a predefined moderator or chair. Usually this is the scribe of the preceding week, this means he/she moderates for the whole week. It his/her responsibility to generate an agenda (TOPs) and to moderate

the whole meeting as to assure that the important topics are handled. A proper preparation by prior reading of the last minutes and having a printed version of the To-Do-list at hand proved to be useful.

The greatest difficulty during moderation was to keep a reasonable level of abstraction during the discussions. As the border between main topics that have to be discussed during the meeting and technical details is crossed very quickly, the moderators job is to always watch for these pitfalls and to prevent the group from discussing topics only three persons really understood.

4.2 CVS Repository

To help dealing with version control, the concurrent versioning system (CVS) was installed. It helps the individual team members keeping any kind of artefacts up to date.

To ensure the proper work of the system and its assistance functionality, the following rules should be obeyed:

- Any kind of main document the group or any sub team created has to be put into the CVS. This ensures that the most current version is accessible. The author of the document should keep in mind to use the same file name in order to show CVS that it is the same document and to use CVS' functionality, i.e. version control. If one uses version numbers within the filename, CVS treats them as different documents.
- All minutes should be put into CVS, where they always are accessible.
- If two or more persons work on the same artefact simultaneously, they are to first check out a second time and let CVS merge the two versions. After that, the artefact can be checked in without losing information. This procedure is necessary since the CVS server itself does not merge documents but simply overwrites them.

With the start of the implementation phase there will be a slightly modified CVS system. As it is impractical to let one team check in and out parts from a CVS from overseas, the Swinburne team will use an own 'source code' CVS for its implementation work since the Dortmund team is on holiday. At a later point of time, as the Swinburne team will finish the project earlier, the source code repository will be moved to Dortmund then.

4.3 The Language

Of course every main document that is sent to Australia must be written in English (BE). The Dortmund team decided to create these documents in English directly without prior versions in German since there are be severe difficulties in keeping these two versions consistent and up to date. The only documents not written in English are the meeting minutes of the Dortmund team since they are be of no interest for Australia.

4.4 Role Assignments

In this project, we are using role based software development. [10][19]

Multiple roles were investigated and their suitability for the project was assessed. After that the roles were assigned to the team members. Because of the complexity of some of the roles and a possible rotation of the assignments, some roles already have their successors defined. As it turns out, the rotation of roles is too complicated and time intensive, so the assignments have not been changed.

Generally speaking, a role defines an area of responsibility of a person. It does not mean that this person will do the job on his/her own. The Requirements Manager, for example, is responsible for the creation of the requirements document but the whole team will gather these requirements.

4.4.1 Requirements Manager

The task of the Requirements Manager consists of talking with the ordering party to find out which functional and non-functional requirements are to be included in the product. He brings the results of this conversation to the designers. They develop a first blueprint. The requirements manager shows the requirements document to the customer. In case of dissatisfaction, another conversation is necessary to find out which lacks exists and as a result a new blueprint is made. These steps have to be repeated until the customer is satisfied. Even if the design is ok a requirements manager is needed at a later time of the project because the customer might have new ideas, which have to be included in the tool.

In our project group the requirements manager's task is to identify the needed features from an evaluation of different existing conference managing tools and discuss the design with Melbourne. He also has to look up other conference managing tools and look what useful features they have. He makes a list of the collected features and shows it to the group. He has to ensure that all features find their way into the design. In our project group the requirements manager is responsible for the requirements document. That does not mean that he has to write it on his own but he coordinates the work and is responsible for the result. [21]

4.4.2 Security Manager

The role Security Manager is very important. The main task is to define and to realize the security requirements of the project. In this way he/she should support the whole team in the main design decisions and realize the concept of the object-oriented security.

There are many security requirements in our project group, because we are going to reach a high level of confidence. For this purpose it is not sufficient to use the usual security mechanisms of Java, so we are to use the concept of the object-oriented security.

The security requirements are an addition to the functional requirements and therefore tend to stay in the background and to be added to an existing design. This adding is pretty difficult, sometimes impossible. The best solution is to integrate the security concepts into the design as soon as possible. [21]

4.4.3 Project Manager

The primary task of a Project Manager is planning, controlling and checking the project group's work. This means, it is up to the Project Manager to keep the whole project "running". Apart from organisational aspects it is the duty of a Project Manager to care for the communication within the team, recognizing and reconciling competing interests. The making out of the project plan is of importance: a software process model has to be rendered more precisely according to the demands of the current project and formed into a project plan. The project plan has to be supervised continually and adapted on demand. The keeping of the project plan has to be supervised to be able to react on probable deviations from the plan quickly. Therefore it is necessary to prepare checkable milestones, which can be established by the group members. The co-operation with the Quality Manager is important

to ensure the success of all development efforts. Motivation of the team members is the last – but not least – aspect of the Project Manager role: especially in times when the project "hangs" the Project Manager must help the group proceeding by a positive view on the project and personal engagement. [21]

4.4.4 Quality Manager

Generally speaking, quality is the correspondence with the requirements. The first step will be the creation of a quality assurance plan, in which all marks of quality are defined explicitly. Of course, these should be controllable.

In PG386, Chairware, security and usability should be top priority. After creating the plan, concrete criteria should be found, i.e. quality indicators and metrics, and the persons who realize the measures, are named.

The second step is to supervise the measures, e.g. quality assurance and guidance. The planned metrics are checked, and counter measures have to be taken if there are divergences.

For the process-oriented quality management, procedure-instructions have to be generated. These have a "group-character" and should not be created by the quality manager alone (kinds of description languages, who has to participate in what step?, what template is to be used for the design?, etc.). The quality manager has to make sure, that the artefacts and documents produced meet the standards laid down by the quality assurance plan and procedure-instructions. He participates in all reviews of code and documents.

Later in the project, the QM is responsible for designing and conducting all tests and for coordinating all testers. This includes designing test-procedures, drawing up a test-plan, controlling the realization, etc. [21]

4.4.5 System Architect

The System Architect's tasks are:

- Designing the majority of the architecture

  S/he has got to work out the major architecture design of the whole system, deciding the main type of all components that will be necessary to fulfil the requirements of the SRS (Software Requirements Specification) document.

- Supervising the development of the several components

  It's the system architects task to ensure the division of the components and to watch and file every component's development progress. S/he has to solve interface problems and find answers to questions of responsibility between several components. A good design reduces the amount for these discussions

- Not defining implementation details

  S/he is not contact person for implementation details. This is the part of the programmers and all necessary additional information should be in the specification documents. In the case of rising problems, these documents should be updated, possibly respecting the work of existing solutions and maybe work arounds.

- Delegating coding tasks

  As the progress in the several tasks might differ strongly, there might arise a need to re-delegate the jobs to the programmers. This must not lead to softening the borders between the several tasks respective components. Major problems with the interacting of different components should be solved, according to the existing specification documents. [21]

4.4.6 System Administrator

The system administrator is the person responsible for the test and integration of software and hardware components either into a system currently being built or a running computer system. After the tests he considers the risks and the advantages that get along with the integration. The system administrator will think twice before he changes the configuration of a properly running computer system.

Additionally, he is responsible for the maintenance of the network and each computer in it. To avoid loss of data if a server blows up he has to backup the important files of the system.

The system administrator is one of the central roles during the set-up-phase. He will suggest which software and operating system should be used, how the network should be configured and so on. During later phases his job will be to take care of the network and computers. This role will never be redundant since technical problems may occur at any time. [21]

4.5 Creating New Artefacts

For the main documents or artefacts, e.g. Requirements Document, OOA ,or OOD, standard-templates were provided by the person in charge in order to keep the whole artefact consistent after merging all parts and to ensure that all required information was given (i.e. it works as a guideline). A sub team could, for example, forget to name priorities for tasks if there was not provided any template including the specific field.

All textual documents are created with Microsoft Office. All diagrams concerning the design and analysis of the system are created using Together. This is a global constraint of the project as we tried to use the same tools in Dortmund and in Swinburne.

4.6 Reviews

Reviews are conducted at the end of each document creation. The date and time of all reviews is specified in the Project Plan [15] and/or the To-Do-list respectively. They may also be conducted when a serious problem arises. Suitable persons for the review team are chosen every time a new document has to be released. The reviewers should not have participated in the creation of the document to be reviewed – just one representative (the author) of the team that created the artefact takes part as to explain details or note any changes that have to be applied.

Reviewers are expected to read the document they are reviewing as well as the standards the document should adhere to - noting any inconsistencies detected. Such inconsistencies can include: layout problems - fonts and formatting; typographical errors – spelling mistakes and grammatical errors; content problems – missing information, repeated information, information which does not meet the documents aims or scope. These problems are to be noted and a list of them prepared before the review meeting takes place.

The time and location of the review meeting is left to the discretion of the particular review team.

During the review meeting, reviewers should bring to the author's attention, any problems they have discovered with the document. It is then up to the author to agree to make any changes to the document that are required or provide suitable justification for the facts mentioned.

Changes that have been accepted are to be made by the author's representative and any associated authors. [16]

Problems during the reviews of our teams were the points mentioned above:

- The review team was not prepared before the actual meeting and therefore lost a lot of time for this
- The time of the review meeting was up to the team itself which resulted in one-day-before-deadline meetings and in combination with the problems above to an incomplete task
- The creating groups left it to the review team to eliminate any inconsistencies and did not pay intention to these details which made the creation team produce unsatisfactory artefacts and reviewing a tough job
- Any changes noted by the review team were directly applied to the document (not by the authors) due to time constraints which did not make reviewing easier

Reviews of documents from overseas were even more difficult as there were no representatives who could explain details to the review team. What is more, the time from creating, reviewing, sending to the other continent, reviewing again, sending comments back, and applying these to the document were not usable.

5 Collecting Software Requirements

5.1 The Scope of the Tool

The electronic submission tool (EST) has to have an interface to the programme committee (PC), to authors, and to reviewers. Therefore, the resulting application has to be distributed in order to run under different operating systems. The PCC is supported during the different phases of the conference (finding committee, call for abstracts, call for papers, assignment of papers, discussion of conflicts). The same applies to the authors (uploading contributions) and the reviewers (bidding, review, discussion, CRC).

5.2 Dortmund Part

To find the requirements for our tool we began to look at existing conference-management tools (like CyberChair). Every person of the team had to look up a conference-management tool. Then we held a lecture and gave a summary of the results to the requirements manager. He merged all results to one document, which was divided into the phases of the conference and the different views (PCC, Author…). Then we subdivided the work. Every sub team had to work out phases of the conference. We took the best requirements of the existing tools, thought of missing requirements and added them gaining a first version of our requirements document but not a homogeneous one. The requirements manager wanted to design a template for the requirements but in the end we used an existing template of a former project group. We fitted the requirements into the document and had our first stable version of the document. After many reviews, changes, supplements and renaming the document we finished our work and had a final version of our requirements list which only consisted of the pure functional requirements.

5.3 Melbourne Part

The Melbourne team used a different approach. They interviewed persons involved in the conference topic, i.e. possible customers for our system, including Prof. Grant who already was PCC and reviewer in former conferences [17]. From these interviews they gained possible requirements for the new tool.

For the creation of the SRS document, the Melbourne team used the "Volere Requirements Specification Document", a template derived from a software consulting company, "The Atlantic Systems Guild" [18], where Volere is a collection of services devoted to the elicitation and specification of requirements.

It therefore includes more than the functional requirements, which were collected in Dortmund. It includes Use Cases, non-functional requirements, attribute types and values, test plans for these attributes and not only the scope of the product but also the scope of the work.

5.4 Merging the Two Different Parts

While the Dortmund team developed a pure requirements list [11], the Melbourne team developed a more detailed and more extensive document [12]. Besides the different approaches and levels of detail, there have been different opinions on the scope of the project, i.e. what should be implemented during the year by our project group. While the Dortmund team thought of a pure "Electronic Submission Tool" which deals with the abstract/paper submission and reviewing process, the Melbourne team went further and tried to figure out a "Conference Management System" with a lot more functionality.

The problem of the merging was not only the geographical distance between the two teams but the scope of the documents was that different. At the beginning we reviewed the Melbourne document and the Melbourne sub team did the same with the Dortmund document. Because of the different approaches there was much criticism and both teams fought for their own version. Some of the criticism was applied to the documents but a big difference remained. After the two teams found out that the scope of the documents differed we decided to have two different requirement documents with the same requirements in it. Both teams reviewed the documents again, and again some changes and supplements were made in both documents but the requirements were still not the same. Then two members of the Melbourne team visited Germany. This was the first time that members of the two teams did a review together. This sub team decided to keep both versions of the document and only review one of them. The Dortmund requirement list was reviewed, changed and supplemented. After one week of work it was finished and the final version of the requirements list is now used by both teams.

5.5 Conclusion

As described above, we had many problems to create a single document for the requirements. The main problems were the great geographical distance and the different scopes of the documents. These problems can be solved easily. The choice of the format and the content of the document must be done together or the supervisors of the two different teams have to develop a standard for the requirements document. The distance is a problem, which cannot be solved this easy. The answer time for requests and reviews via e-mail is too long. To solve this, you have to have faster communication (for example via phone or a live talk). The best solution is to have an earlier visit of one team so the requirements document can be developed together. Another solution is to synchronize the documents at an earlier stage of the project. If it is synchronised too late it is nearly impossible to merge two different standards in one document.

5.6 Results

The result of all the work is the "Requirements List" [11] (Figure 3).

Name	Requirement	Pr.	Type	Reason	funct.Req.	nonf.Req.	fulfilled
GL/P1	The AUTH may delete his conference data from conference data base at any time. See SY/P1.		MUST		X		
GL/P2	PCC may reject abstracts/papers at any time.		MUST		X		

GL/P4	The AUTH may change his data. See SY/P1.		MUST	For example change his email address	X		
GL/P5	The AUTH can withdraw his/her already submitted abstract/paper. See SY/P5.		MUST	If the AUTH has changed his mind, he should be able to withdraw a paper from the conference.	X		
GL/P7	The REV may login and change his data. See SY/P7.		MUST	see GL/P4	X		
GL/P8	There is a To-Do-List (checkboxes) provided for the PCC about what has to be done.		MUST	PCC may keep an overview about things that need to be done or have already been completed.	X		
GL/P9	PCC may add/remove/modify To-Do-List-Entries.		MUST		X		
GL/P10	All system actions are logged for traceability. See SY/P24.		MUST		X		
GL/P11	The ADM may change/add/remove papers/abstracts/authors/reviews/reviewers.		MUST	he is the administrator	X		
GL/P12	The REV may withdraw from conference at any time.		MUST		X		
GL/P13	The system supports only one conference at any time.		MUST	Multiple conference management should not be supported.	X		
GL/P13	The ADM can convert PS-files into PDF-files.		CAN	This provides the same file format for all documents.	X		
GL/P14	There can be more than one PCC.		MUST	Needed, if a conference has more than one PCC.	X		

Figure 3: Excerpt of the Requirements list [11]

The list consists of the different phases a conference has: The "Global phase" which consists of requirements needed during the whole conference and the following phase listed in order of appearance: "Set-up", "Finding committee", "Call for Papers", "Submission of Abstracts", "Submission of Papers", "Bidding Phase", "Assignment of Papers", "Reviewing", "Discussion", "PC Meeting Preparation", "Notification of Authors" and "Camera Ready Copies". For each of the phases a table has been created. It contains the following information: The "name" of the requirement (abbreviation of the phase (in this case GL) plus a process number (P xx)), the description of a "Requirement", "Pr" which is the priority of a requirement., "Type" differentiates between "MUST" and "CAN", the "Reason" why the requirement is needed, the difference if a requirement is functional (funct. Req.) or non functional (nonf. Req.) and the state of the requirement i.e. if it is fulfilled or not.

6 Object Oriented Analysis

6.1 Our Procedure

The OOA development phase started by forming an OOA task force. During an initial meeting, the team tried to find entities of the system by using the list of requirements derived from the last development phase. These were drawn into a first draft of a class diagram [13]. From this point, the work was subdivided according to the conference phases. This means that the five task force members were assigned to two phases each. They had to find additional classes and their corresponding methods by designing UML sequence diagrams. Together, the tool used to design the diagrams, is able to link the diagrams so that newly found operations and classes are included into the main OOA class diagram automatically.

As it turned out, not every member of the design team knew about sequence diagrams and their purpose. What is more, the special features of Together have to be actively used in order to gain any advantages from the tool, which is not easy when working with it for the first time.

Additionally, the subdivision by phases proved to be not suitable. Many similar tasks are repeated in different phases of the conference and were therefore designed in many different ways (if any) by the different participants according to their part.

This resulted in several confusing ways of doing one single task, method names that were not at any decent location, names that nobody could refer to, god classes, or just missing requirements. It was left to the review team to form a complete work out of the different parts and therefore it was not possible to do any reasonable review of the results from that second stage at all.

These facts becoming obvious, using some results from the review, the OOA task force tried to rework the diagrams by linking them correctly and eliminating redundant systems but did not manage to eliminate all superfluous method names due to time constraints (in spite of a two-week delay) and motivation problems.

The final step of the OOA task force was commenting every diagram in order to make it possible for somebody not actively participating to understand it.

6.2 Results

In Figure 4, the class diagram is subdivided into parts marked by bounding boxes. The following is a description of the structure and functionality of these parts.

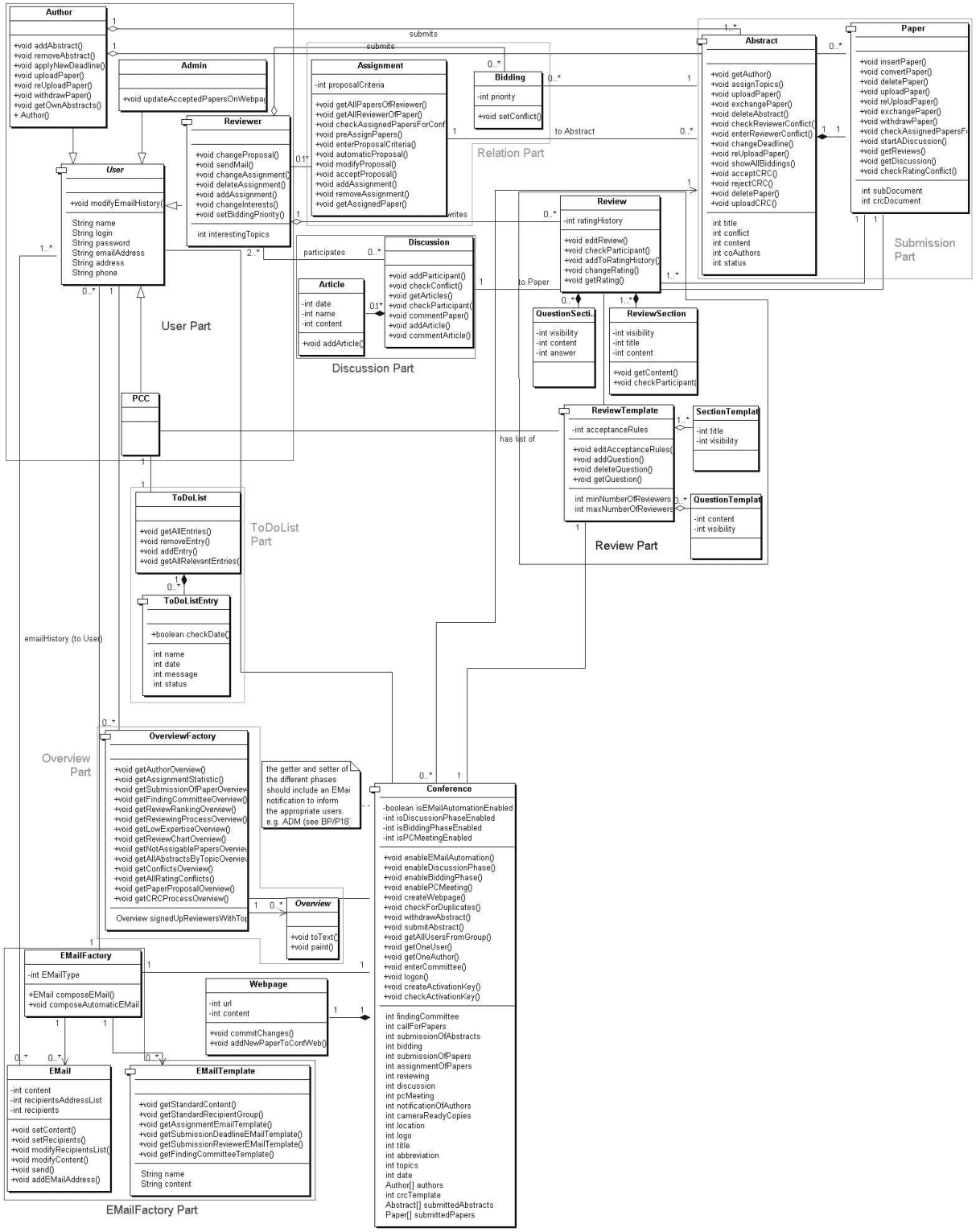


Figure 4: OOA Class Diagram

6.2.1 User Part

There is one user super class, which includes all attributes common for the different user types. A sub class, namely PCC, Admin, Author, and Reviewer, which includes type specific information, represents every type.

6.2.2 Relation Part

Assignment and bidding-relations are administered in this part. There is one Bidding-object per Reviewer-Abstract relation, which defines its priority (e.g. 'want to', 'don't want to', 'ACM conflict'). The Assignment class incorporates all assignment functionality including the Reviewer-Abstract/Paper relations and the assignment proposal generation for better usability.

6.2.3 Discussion Part

The discussions were meant to help coming to common ground during the review/discussion phase when the reviewing results are seriously conflicting. Therefore these inter-review discussions are held between the reviewers of a specific paper and discussions are opened for all papers individually. As a consequence, every Review has its own Discussion-object holding zero to many Article-objects.

6.2.4 Submission Part

All artefacts submitted by the authors are designed in the Submission Part. Artefacts are abstracts and papers, where every paper has exactly one abstract and every abstract has zero to one paper, depending on the phase the conference is in.

6.2.5 Review Part

The design of the review system is found in the Review Part. In order to keep the reviews flexible (any conference may have different needs), the actual Review object is subdivided into zero to many QuestionSection and zero to many ReviewSection objects. During runtime, the composition of reviews is looked up using the ReviewTemplate, SectionTemplate, and QuestionTemplate. This enables the PCC to define the review appearance at the start of the conference (set-up phase)

6.2.6 To-Do List Part

This is a rather simple system. The PCC is to have a To-Do-List, which should remind him of important deadlines etc. There is a one to one relation between PCC and the To-Do-List, which holds zero to many entries.

6.2.7 Overview Part

As there are many different kinds of overviews and statistics, we decided to let all of them implement two methods: toText() and paint(). To realize this, there is an abstract super class Overview that all different kinds of overviews have to extend. The different types are not included in this class diagram.

The OverviewFactory is to provide any client with an Overview.

6.2.8 EMailFactory Part

Similar to the OverviewFactory, the EMailFactory provides clients with requested EMail objects. These are built according to the EMailTemplate objects, which define content, recipient, etc. The actual EMail object is then re-

turned to the client who may modify its content and send it by its own send() method.

6.3 Conclusion

In general, the first step to be made is to make sure that every member of the team knows what is to be done and how it is to be done. In this case, we should have made sure that everybody knew the purpose of the OOA, the purpose and results of sequence diagrams, and the correct way to work with Together in order to use its advantages. Before this step is not done, any subdivisions of the task to the individuals are not suitable. This would mean conducting a small workshop prior to the OOA work.

To avoid any double work, the mode of the subdivision should have been altered. We should not have split the sequence diagrams according to the phases because of the problems, which are described, in the preceding chapter. Each participant has to be actively involved in the process and 'hot spots' have to be designed within the whole team since there are some sequence diagrams who use the same classes and methods. The check for consistency, logic, and completeness cannot be left to the review team but must be done by the authors themselves.

A good documentation for each class and each diagram is very important and should be written by the time creating new objects as without a documentation nobody is able to understand the work another task force member did which would lead again to double work.

As a final conclusion of this phase, one can say that after a two-week delay, the results were still poor due to the facts mentioned above. This led to a heavy workload in the subsequent phase, namely the two weeks of design meetings with the Australian sub team and the OOD phase afterwards.

7 Object Oriented Design

We intended the object orientated design of the Chairware software system to be done during the visit of the Swinburne sub team. As mentioned above, the time was almost completely used for the OOA because of two main reasons: First, the discussions and arguments from the Dortmund OOA phase had to be redone, because the OOA of the Australian team did not deliver a complete class diagram and therefore the German OOA diagram was used as basis. The other reason was the general opinion that there would not be a significant point of time for the switch from the analysis phase to the design phase.

7.1 Our Procedure

While a fluent migration from OOA to OOD is quite common, the circumstances of the Chairware project (especially the distributed development) made it necessary to make a cut: The project was required to be component based. This was not respected in the OOA at all. After the visit of the Australian sub team there was a big class diagram, but not a single component – while there were plans for the components to be split among both teams to develop them independently. It turned out that almost no part of the program was independent from another. [14]

During the two weeks after the visit, the design of the program was switched to components. Because of the holidays in Australia, the redesign was completely done in Germany using Together. It included a lot of movements within the classes of the diagram and the addition of many interfaces. This was done according to some very simple rules:

After the single generic components had been identified, the relations between them had to be resolved, as no generic component was allowed to rely on another generic component or the Chairware business logic. That makes pretty sense, as the relations and connections between the generic components are a major part of the Chairware specific business logic and their data structure.

All data (containing classes) that is supposed to be stored by the application was put into an abstract database by using (container managed persistent) entity beans or – in the case of configuration files – saved in files on the server's local file system. Every component should compile, deploy, run and be testable separately. We found out that there are almost no generic data classes suitable for the Chairware system, because every data object was to be processable or usable by two or more components, e.g. the reviewer has many roles on the system: he is a writer of reviews in the eyes of the ReviewManagement component, a person taking part in discussions for the discussion component and a recipient and sender of emails in the eyes of the email subsystem. As these three components must not interact on each other and therefore cannot be used to create an appropriate object for other components, the creation of these multipurpose entities was up to the Chairware business logic and the reviewer classes ended up in the Chairware business logic.

7.2 The intermediate results

The preliminary design of the whole application consists of four groups of classes or columns. In the class diagram they are easy to discriminate by their colours, which gave the diagram another dimension. The diagram itself was hardly printable, as it reached up to 130 DIN A4 pages to print on or a 20" hi-resolution screen and a fast computer to view. The colours of the classes can be seen by using Together.

The purple classes and interfaces define the Chairware application and its specific business logic. These classes do not define a real component, but the main part of the application. They are highly dependent on all other components, except of the web application mentioned later. We named this part the access layer, because it controls the user's access to the database.

The yellow classes and interfaces are generic components with generic functionality and a few data classes interfaces.

The red classes and interfaces contain data classes (usually entity beans), which are Chairware specific, but will have to be processed by one or more generic component.

The green layer defines the complete web application. The web application strongly relies on the Chairware application as it exchanges much data with the application: the web pages content.

7.2.1 The 'Purple' Packages – Access Layer

Session beans of the access layer can be understood as a collection of XML servlets. These stateless beans are created by the user interface and process a text string that conforms to a special XML stream. This string contains all data needed for a specific 'event' in the system. The session bean processes this stream and replies with another XML string (or stream) with all relevant data to the user interface and displays the system response of this request. So the view of the response and the content are completely separated from each other. The design of the access layer, the separation of the functionality into several session beans and the fine design of this layer including the communication to other packages is a task of the Swinburne sub team. A toolkit or a framework was finished until the visit of the Dortmund sub team in Australia.

7.2.2 The 'Yellow' Packages – Generic Components

We found about ten generic service providing components that are needed for the application, but are not conference management specific. Due to the lack of need for any other functionality than the one needed for the Chairware system and the quite special purpose of the Chairware system, the functionality of these components is sometimes very limited. Every component is intended to be usable as a standalone software that might be used by another application. Some components that have reasonable generic parts are the EmailFactory, the User Management and the Contribution Management.

There are components that are split into a generic and a specific part, while others cannot be split, e.g. on the one hand the Relation Management component doesn't know a general kind of relation, because the relation management only makes sense in the context of conference management. On the other hand the email factory is completely generic, because there are no special conference-related requirements for this component.

7.2.3 The 'Red' Packages – Specific Components

The red classes are completely entity beans. Every entity bean provides some properties. Most of these properties are mapped directly to a field in the database, while others are evaluated every time they are used. The Remote Interfaces of these entity beans extend quite a few interfaces (besides EJBObject, as every entity beans Remote interface does), some of the generic ('yellow') components are implemented against. This ensures that the generic components can operate with the specific documents, but also makes the red layer depend on the generic components.

7.2.4 The 'Green' Packages – User Interface

The green package has to create a user interface. Its implementation is intended to be completely independent from the other components of the Chairware system. The communication between the business layer and the web application is realised via a XML basis and possibly via streams. This enables the systems front end to be exchanged very easily. Possible ways to provide a user interface are web applications (as servlets or JSPs), client side applets or stand alone applications. Even ASPs or other techniques are possible. These package's main aspects are the presentation of the user interface, the session tracking if this is needed by the user interface, the navigation, the internationalisation and localisation. The whole functionality of the Chairware conference system will be reachable and presentable by the user interface.

7.3 The final design

During the weeks of independent development, every of the named packages was started to be implemented, and some refinements of the initial design were done.

The Access Layer appeared to be the central container for the Chairware business logic that contained lots of tightly linked functionalities. The first design of this layer was a major collection of Enterprise Java (Session) Beans. This appeared to be a necessarity of an EJB project, but resulted in many EJB specific problems like reentrance and other transactions. Additionally, splitting of the business logic among several application servers was not intended, nor the exchange of parts of the business logic at runtime. So the EJB approach was dropped and common Java classes were used for the business logic. The separation of the different business logic parts was another issue. The very first separation of all the methods made the cut between methods necessary for web page display and functionality that made changes to the data base. This resulted in (the very last) two Session Beans of this layer: 'ContentManager' and 'Processor'. These Session Beans were intended to be something like XML-Servlets that got an XML structure as parameter and returned another XML structure as result. This was dropped for several reasons:

(a) Performance. This XML parsing can end up as bottleneck,

(b) the String/XML serialisability is not necessary, because we decided to implement the user interface to be written in Java, so the Java intern mechanism was enough for our purpose and

(c) the parsing of XML was very type unsafe anyway. We designed generic 'tree-able' value object classes on our own that had better runtime characteristics and a slightly better type safety than the original XML idea and called them CommBeans. These CommBeans are extremely flexible and failure tolerant.

The functionality of the generic components ('multi purpose components') were more and more degenerated

during the fine design of the system. An example: At first the User Management appeared to be a generic component that might have been very independent. During the fine design and implementation (which was done in parallel due to some other problems) we found out that the User Management doesn't have too many generic functions and almost all we expected the component to do was very Conference Management specific and not generic. All the generic User Management functionality we found was the detection of two users with the same loginID – not too much. The approach to implement a system with generic components was misleading. The components functionality was too generic. We should have tried to implement a generic component for User Management in Conference Management Systems to end up with a better separation of the functionality. Anyway – some components remained independent and are really reusable components, e.g. the discussion component is almost free of any chairware specific code. The Access Layer's work in the context of the discussion forums is to moderate and translate the method invocations from the User Interface.

The Specific Components were almost resolved. Most of the EntityBeans ended up in the Access Layer, as they contained Chairware specific logic. The largest specific component we designed in the project is the Relation Management. Every reviewer (or referee) has a certain relation to every paper. The referee can bid for the paper, can have a (confirmed or unconfrmed) conflict, can be assigned or might have confirmed that a camera ready copy of the paper is ready to go to the press. Some of these 'types' depend on others, while some must not be combined ('conflicts'). This logic is very Chairware specific, but easily separateable from the rest of the business logic.

The design of the architecture of the user interface is not too complex, although there are many classes. It is based on the Struts Web Application Framework (Jakarta / Apache). The decision for this framework was made fairly late, but was very lucky. Struts delivers it's very own Model-View-Controller framework. The models in struts are value objects that can conveniently be displayed on JSPs pages. These value objects (namely ActionForms or Form Beans) can be thought to be mapped to entity beans in an EJB container. The view in the struts framework are JSPs that contain pure display information and no other logic. In fact we tried to reduce the Java code in the JSPs to a minimum and use self made tag libraries for common visualisation problems. The controller of the framework is a collection of java classes that implements the first level logic behind links or buttons in the web interface. Usually these classes are responsible for retrieving content from the Access Layer and processing links form input from the web interface. These three components are binned in a big configuration file, namely struts-config.xml – the first file in the whole projects repository that reached more than one hundred revisions during the implementation phase.

8 Security Aspects and Approaches

8.1 Overall Process

Starting from our initial intention to modify the Java Virtual Machine, we experienced some major technical problems in conjunction with the EJB architecture and the orion server. The result was that we gave up to this method and built a pre-processor which had the responsibility to incorporate the security code into our business logic.

The goal of our investigations was to separate knowledge about security issues and business logic. By doing this, application developers should not have to bother about security (or at least not much). By automating security checks, testing for security leaks should have been a minor issue (although we did not plan to realise a provable system). Traditionally, when these requirements arise, people often implement "declarative security", which means that security checks are done automatically but the permissions are defined totally separate from the implementation (e.g. in a configuration file). This approach does not fit our needs, since we have a dynamic set of users and very specific security policies and modifying a configuration file every time users are added to the system is not acceptable.

We therefore tried to find a way between the error-prone and difficult to test programmative security concepts and the non-flexible declarative security concepts.

The concept should work similar to Figure 5:

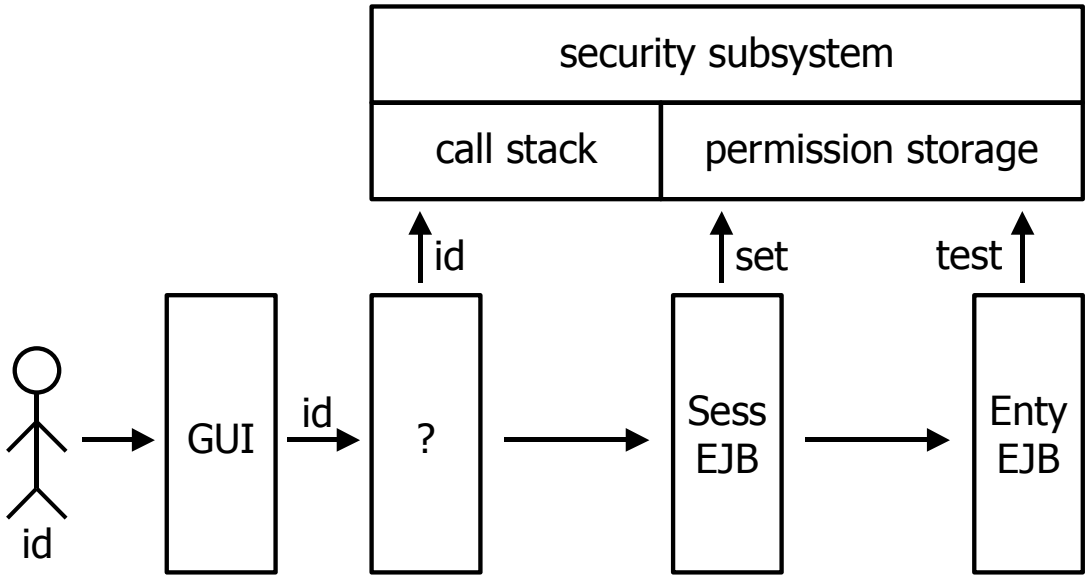


Figure 5: Schematical diagram

Each request of the user interface to the business logic should contain some identification of the user, which should be a realistic assumption for most practical client-server applications. The identification might be contained in a session identifier or something similar, but the corresponding user identification should be retrievable from that. The first business layer after this identification is stored in the call stack. Permission checks can be done later on. Permissions are divided into read and write permissions and attached to object instances. This setting up of permissions is done in Session Beans in our case. The corresponding objects, for which they are defined, are Entity Beans. The testing for the actual permissions should take place in the methods of the latter. That is the time when the user identification is obtained from the call stack and tested against configured permissions.

The question remained how to realise the vertical arrows (in Figure 4), i.e. how to insert the security code into the business logic.

We initially designed a scenario where the whole application runs under a modified Runtime Environment, where security attributes are attached to every object and the Java Virtual Machine checks access to the objects via so-called "Wrappers", classes which are generated and compiled "on the fly" at runtime. The first difficulty we encountered was the Container Managed Persistence of the EJB application server which only serializes those attributes specified in the deployment descriptor into the database. The whole concept would have required several major changes in the functionality of the application server (Container Managed Persistence, the transaction system, multithreading and other things which might have encountered later). As we wanted to develop a system fairly independently of the middleware and the alike, this effort did not seem worth the effect.

We investigated whether working with custom class loaders instead of a modified Java Virtual Machine could alleviate those problems. Unfortunately, the application server refused to work with those, presumably by intent of the manufacturer.

So, shortly after the beginning of the second semester, we decided to use a source file pre-processor for inserting security code into the application source files. Unfortunately, this lead to a more complicated deployment process.

8.2 Experience

8.2.1 Modifying the Virtual Machine

During the Design and Prototyping Phase, we build some testing scenarios to examine the Virtual Machine Approach. Doing that, we took the research results of Thomas Bühren's diploma thesis [1] into account. He examines therein a modification of the JVM for coupling arbitrary method calls with Security permissions and customisable cryptographic layers. Our approach wanted to extend this by adding additional data to the objects. This would have been enabled us to specify our object-based protection.

Based on his experience, we constructed a modified Virtual Machine prototype and incorporated some test attributes into the Java class "Object", which would add those attributes into all inherited objects. Unfortunately, the consequences were difficult to handle: Object serialisation simply did not consider the additional attributes and other major problems arose.

8.2.2 Storing the User Identification

In parallel, we tested how to store the "responsible" user of a method call all way down the call stack. Therefore, we used something called a thread-local variable, which attaches an object to the current thread. This technique can be applied quite universally, since almost every multi-user server will handle new requests by creating new threads. In general, threads can be created thereafter as children. Although this does not apply in our environment, the "InheritableThreadLocal" class offered by Java is able to handle this.

8.2.3 The Class Loader Approach

Because of our problems with the modified "Object" class in the altered Virtual Machine we decided to look for possibilities of using a modified class loader to overcome these problems. The idea was to install that modified class loader in the application server. It generates a compatible class by inspection and hands it back to the client. That compatible class could have contained all the attributes we needed. Unfortunately, we did not manage it to get a custom class loader to be used in the application server. Apparently our application server was fixed to his own class loader and refused to work with a different one.

8.2.4 The Source Code Pre-processor

The approach we finally realised is to build a Java source code pre-processor which analyses the classes and adds the needed security checks directly into the source code. The problem of this approach is that the source code must be available. Instead, it has an advantage of technology-independence. The generated code should simply be compiled just like the original Java code.

The pre-processor uses the Java compiler's own tool classes for parsing the code, such as ClassDeclaration, MemberDeclaration etc. It renames all the non-static methods of the class and inserts methods with the original names which contain a call to the renamed methods surrounded by security code performing the rights check and transmission. The security code is added automatically, based on the kind of access the method adds to the members of the class. The pre-processor analyses the methods and tries to detect if the method adds read or write access. The following rules are used by the detection:

- get-/set methods are considered to have read or write semantics, respectively
- if there is no return type (void) and only primitive parameter types, write semantics are assumed
- if there is no return type (void) and at least one non-primitive parameter, both read and write semantics are assumed
- if there is a return type, both read and write semantics are assumed.

Especially the last two rules are very restrictive, and they will be used in many cases. That's why the pre-processor gives the developer a possibility to configure the rules for each method. This can be done by using special comments in the Java source code, which are written inside the normal javadoc comment block for the method. The following keywords are used:

- `@read` - method is "read-semantics", i.e. the method returns data from its object.
- `@write` - method is "write-semantics", i.e. the method modifies its object.
- `@none` - method is neither modifying nor retrieving data from its object.

- `@wrap` – this comment should be written in the javadoc comment block of the class which should be analysed by the pre-processor.

Except, there is a configurable list of the methods, which should never be analysed.

The user identification should also be configured everywhere the user logs into the system. This should be written inside the javadoc comments using following keyword:

- `@userid` codeForRetrievingUserHandle – this comment should be used with the method that gets a user identification parameter which can be turned into a javax.ejb.Handle referring to a User object with the code provided.

The configuration of the rights should also happen with the help of similar comments, which are now written in the methods where the specified objects are created or the additional rights should be added. The comment should be written directly into the method using following syntax:

```
/**ifsec
* grantread/writeaccess
*/
```

The pre-processor removes the comment signs so that the code will be compiled, too. This code should be easy to understand by application developers, since they would probably end up creating and maintaining it. Therefore, we created a utility class (SecurityStuff), which abbreviated some usual, domain-specific permission configurations. Calls to this class were inserted into the ifsec statements.

8.3 Evaluation

In general, we expected much less security-relevant code in the business logic. But the personalisation of views forced the application developers to analyse relations between objects like papers/reviewers, papers/authors to an extend that removes the necessity for the automatic security mechanisms for convenience reasons.

It was not possible for the application developers to *rely* on the security mechanisms to filter out data that is inaccessible, because the security subsystem was specified too late in the development process.

After creating an integrated example of a very limited subsystem, we saw several practical problems which could not be solved within the scope of our project group. The major problem was performance: every call to a secured bean leaded to numerous database queries to verify the permission to actually perform this call. Given that creating an object like a Contribution requires about 10 subsequent method call (set-methods), this adds up to a relevant amount of time (over 30 seconds in our production environment). This time will linearly grow with the amount of permissions configured in a system (i.e. tupels of objects and users/groups). So, for a practicable solution, technologies like caching etc. are definitely required.

9 Implementation Phase

9.1 Introduction

In this chapter we want to describe the whole implementation process of the Chairware product. It starts with an overview about the procedure of planning and accomplishing this project hase .

Afterwards we describe our experience we made using the chosen developing tools like TogetherJ, Forte etc.. There, the problems we went through are described.

During the implementation phase we built sub-teams that were responsible for the different parts of our programme. One team designed and implemented the graphical user interface (GUI), a second team implemented the functions in the so-called access layer, another one was intended to work on the reports that could occur in Chairware. The work of these different teams and which problems occurred are described in paragrahps 9.3 to 9.6.

One major part of the whole developing process was the so called "Baustellenliste" which was the to-do list during the implementation phase. In the last paragraph of this chapter we want to describe how we used this list to determine responsibilities and deadlines.

9.2 The Tool Experience

During the whole project time, several applications and software tools were installed and used. Already in Nordhelle, some utilities were introduced to the team members and some decisions were made about the subset to be used in the first weeks.

9.2.1 Office Application

We also used Microsoft Office to create some source code. We developed our phase change table (see chapter 9.4.4 for further details) with Excel and exported it to a textfile. The textfile was integrated into the source code. See chapter 12 for further details of the Office application.

9.2.2 Modelling Application

For modelling tasks, there where only two competitors to decide between: Together Control Center and Rational Rose. The decision to use Together Control Center (TCC) was based on the fact, that there is a campus license for the product, as it is used in some other classes and (hence) some persons of the team were familiar with the product.

TogetherSoft claims TCC to support every phase of a software development process (round-trip engineering): model, build and deploy. This worked quite fine for us. In fact TCC hid a lot of the setup of a deployment of the team. The german team never ever deployed an application into the application server, that used a DD (deployment descriptor, an roughly 50k XML-file) other than the one used by TCC Deployment Expert. After the setup of the tool, the whole deployment of the application was a thing of about 8 mouse clicks.

The trade-off of TCC are its high requirements regarding memory, processor speed and screen diameter. We had only very few suitable workstations, that fulfilled the needs of half a gigabyte of installed main memory, a 500+Mhz processor and a 17+" diameter screen. Another trade-off is the high complexity of the J2EE framework and TCC trying to hide it as good as possible. The J2EE described in tutorials expected the programmer to implement almost everything by hand, while Together assisted in (often non-transparent) ways. If an error occurred during the deployment or at runtime there where many directions to investigate: the application server, the deployment descriptor built by Together or our application code.

9.2.3 Web Application Development

Forte for Java 3 (Community Edition) was used to build the web application's classes, while we had no common tool for editing .jsp-files. The application server orion supports 'exploded' jar-files, so the web application was not required to be binned in .jar and a single compilation of the source files into the right directories made them run. (This is why we did not need to use the Enterprise Edition.) Once a class was loaded, a newer version of the file was not reloaded until the next application server's restart – which was a matter of seconds, if there was no redeployment of the EJBs necessary. To edit the .jsp-files, both UltraEdit32 and TextPad were used. Both text editors supported a certain syntax highlighting, but no code completion. To ensure a high quality of the html-code, we validated the code with the online html-code validation service provided by the w3 consortium.

9.2.4 Source Code Management

To manage the source code we used CVS, a concurrent versioning system. Properly setup and used it supports concurrent work on the same files, which can be merged. Many team members were used to this system from other classes. On the client side there was both a standalone application to retrieve and submit code, namely WinCVS, or the integrated versioning modules of the IDEs Together and Forte.

We did not run into any major problems with the source code or the versioning system, so it's hard to say if CVS was appropriate or not. CVS keywords where not used in the source code. We had one repository and three modules in daily use.

9.3 GUI – The technical Approach

9.3.1 Java Servlets

The Servlet API was developed to leverage the advantages of the Java platform to solve the issues of CGI and proprietary APIs. It's a simple API supported by virtually all web servers and even load-balancing, fault-tolerant application servers. It solves the performance problem by executing all requests as threads in one process, or in a load-balanced system, in one process per server in a cluster. The servlet engine loads the servlet class the first time the servlet is requested, or optionally already when the servlet engine is started. The servlet then stays loaded to handle multiple requests until it is explicitly unloaded or the servlet engine is shut down.

Security is improved in many ways. First of all, the user rarely need to let a shell execute commands with user-supplied data since the Java APIs provide access to all commonly used functions. The user can use JavaMail to read and send email, Java Database Connect (JDBC) to access databases, the File class and related classes to access the file system, RMI, CORBA and Enterprise Java Beans (EJB) to access legacy systems. The Java security model makes it possible to implement fine-grained access controls, for instance only allowing access to a well-defined part of the file system. Java's exception handling also makes a servlet more reliable than proprietary C/C++ APIs - a divide by zero is reported as an error instead of crashing the Web server.

9.3.2 Server-Side Scripting (JSP)

JavaServer Pages™ (JSP) is a web-scripting technology similar to Microsoft Active Server Pages (ASP). However, it's more easily extensible than ASP, and it isn't proprietary to any vendor or any particular web server. Although the JSP specification has been managed by Sun Microsystems, any vendors can implement JSP in their own systems.

JSP is a presentation layer technology that is placed on top of a Java servlets model and eases working with HTML. It allows the user to mix static HTML content with server-side scripting to produce dynamic output. By default, JSP uses Java as scripting language; however, the specification allows other languages to be used, just as ASP can use other languages (such as JavaScript and VBScript).

JSP offers a robust web application platform and a simple, easy-to-use language and tool set by providing a number of server-side tags that allow developers to perform most dynamic content operations without ever writing a single line of Java code. So developers who are only familiar with scripting, or even those who are simply HTML designers, can use JSP tags for generating simple output without having to learn Java. Advanced scripters or Java developers can also use the tags, or they can use the full Java language if they want to perform advanced operations in JSP pages.

9.3.3 Struts

Struts is a Model-View-Controller (MVC) framework compatible with Sun's J2EE platform and primarily based on servlet and JSP technology. Struts is part of the Jakarta project (which is part of ASF, the Apache Software Foundation). The framework has gained considerable attention over the course of the last year because of its ease of use and ability to fit today's developers' needs in building applications and building them fast. Struts combines Java servlets, Java Server Pages (JSP), custom tags, and message resources into a unified framework and spares

the developer the time of coding an entire MVC model, a considerable task indeed.

The Struts package, even though it is revision 1.0, has complete documentation. This includes a user's guide as well as developer's guides. One thing important to note is that even though this is a 1.0 product, it is extremely stable.

The framework itself can be broken down into four primary areas, three of which correspond nicely to the MVC pattern:

- The model, which is an Action class, provides the business logic written by the developer. The dispatch from the controller to the Action class is based on a configuration that is provided by a `struts-config.xml` file.
- The view, which is a set of JSP custom tag libraries, works in concert with the controller servlet, and that allows fast creation of forms for an application.
- The controller, which is a servlet, dispatches incoming requests to an appropriate Action class.
- A number of utility classes support XML parsing, automatic population of JavaBean properties, and internationalisation of prompts and messages.

The **controller** is really the traffic cop of the framework and determines when things happen. The controller servlet is responsible for packaging and routing HTTP requests to the appropriate object in the framework. This is either a JSP page or an Action class. The controller is in the `web.xml` file as an instance of `org.apache.struts.action.ActionServlet`. When the control is initialised, it reads a configuration file (`struts-config.xml`) that specifies action mappings for the application, as well as a slew of other things. The controller then uses these action mappings to determine where to send the HTTP request. The application action then takes over and performs whatever business logic is necessary. An important aspect to note is that Action objects have access to the servlet's methods. This is a powerful feature, because when the Action object forwards control to another object, one or more shared objects can be forwarded by adding them to one of the standard collections shared by Java servlets.

The **model** is an object that takes the request from the user and stores the results for the duration of the process. Usually a JavaBean passes along the information in the process. Model objects are typically application specific since they are basically the business logic of the application. Struts provide the ActionForm and Action classes, which extends to model objects creation. It is in these model objects where form validation, or any type of preprocessing of request data that is necessary, usually takes place. It's possible to have a one-to-one correlation between model objects and request pages, but it's not necessarily required. It's also quite possible to reuse a model object for multiple page requests. If there is no action necessary for a model object, the controller can send the request directly to the view object, as specified in the `struts-config.xml`.

The **view** object is often a JSP page. The Struts framework doesn't actually provide the JSP, but it does provide a number of tag libraries that easily allow JSP integration into the Struts framework. The interaction of Struts with JSP allows for storing data of an input form in a form bean; that is the ActionForm we mentioned earlier. The ActionForm is used by the Action class for doing field validation. If errors are found during validation, there is a shared mechanism available in the framework for raising and displaying error messages.

The Struts framework includes a number of custom tag libraries, which are used in a variety of ways. While the use of these libraries is not required to use the framework, it is not so bad to them anyway. These libraries include:

- `struts-html` tag library - used for creating dynamic HTML user interfaces and forms.

- `struts-bean` tag library - provides substantial enhancements to the basic capability provided by `<jsp:useBean>`.
- `struts-logic` tag library - can manage conditional generation of output text, looping over object collections for repetitive generation of output text, and application flow management.
- `struts-template` tag library - contains tags that are useful in creating dynamic JSP templates for pages which share a common format.

Additionally, the custom tags in the Struts framework use the internationalization features built into the Java platform by using resource files. To provide messages for another language, simply add another resource file.

The **`struts-config.xml`** file is really the glue of the entire framework. While `web.xml` defines where a request should go upon arriving, `struts-config.xml` determines exactly what will happen to it. This is where the mappings of actions are defined. As with the growing number of XML configuration files, the advantage of using one is in keeping a system modular and easy to maintain. It prevents the hard coding of objects to be called within a component. Changes can be made on the fly by changing the configuration files without having to redeploy applications. The controller uses the `struts-config.xml` file to determine which object to call for a given action request. The file is read on startup and the relationships are stored in memory for optimal performance.

The entire control of the flow is done by the `struts-config.xml` file, depending on return values from the various action classes. This is all centrally located and easy to adjust. Neither the action's servlet nor the JSP need to know (and in proper n-tier development, shouldn't have to care) where the result set that is being displayed comes from.

The only drawback that we've found with the framework is this: since it is a large-scale project, there are many screens and many request types. It seems that there are many Action classes, since we basically need only one per request type. We think that this could be avoided by doing some careful planning upfront, but for new product development it isn't always possible to identify the common requests.

9.4 GUI – The Surface Blueprint

In this paragraph we describe the process of planning, designing and implementing the graphical user interface (GUI) for Chairware.

9.4.1 Planning Phase

The planning phase for the GUI design started during the visit of the Melbourne team in Dortmund. We had to decide between two alternatives: a client application or a web-based user interface.

We chose the web interface due to the following advantages:

- no installation on participants' computers required
- independency from a certain machine, only web browser needed
- no special operating system required

We agreed that the user interface should only be the interface between the user and the database. It should only get data from the database or send data to the database. No logic should be directly implemented into the Java Server Pages, but into a different layer, the access layer.

The whole user interface should consist of three different sections, one for each user profile.

To get an overview of all functions the user interface should have, we went through the requirements document, requirement for requirement, and started to draw a sitemap with these informations.

After the visit we split up responsibilities for the implementation of chairware. Dortmund team wanted to implement the backend functions, Melbourne team would do the layout and implementation of the user interface.

9.4.2 Implementation Phase

Melbourne team started to design the GUI. After their part of the project was finished for them in October 2001 they sent us the results of their work.

What we got was just a prototype of the actual web interface. Only a few functions in the PCC section had been implemented. The only thing that was quite useful was the neat layout of the web interface.

A lot of work was to do to get the user interface up and running. So we tried to understand what and how the functions had been realized in the UI. In this phase we encountered three major problems:

- The first was that there was no documentation about the whole user interface. To understand the used techniques we had to look directly into the source code.
- The second was that we could not contact the Melbourne team anymore. The project was the final exame for their studies, and so they had left university.
- The third was that all business logic had been implemented directly into the JSPs.

We decided to implement a completely new user interface. The only thing that was left untouched was the layout of the web pages.

9.4.3 Reinventing the user interface

We started to build a prototype for the web interface. The first step was to write dummy JSPs for each page of the different sections just to make the web application "browseable". A new directory structure was created. It consisted of one directory for each user profile, a "common" directory which housed JSPs that implemented functions not related to a special user profile and a "shared" directory in which we wanted to put the JSPs that would be used by more than one user profiles.

A GUI subteam was formed. The team members should learn the techniques to use JSPs with the Struts Framework.

At first the web pages had to be visualised to get the layout and all data on this page. For each web page a sketch was drawn by hand, the needed data was noted on the sketches. These sketches were reviewed to integrate functions that might have been left out by mistake. The GUI team specified which get-/set-methods the access layer should provide information on the pages and all data into the database. It was planned to finish implementation on the author section till Christmas 2001.

Due to lack of communication inside the GUI team and between the GUI team and the access layer team, nothing happened. The access layer team waited for the method that should be done by the GUI team. The GUI team was convinced that the access layer team could better specify some useful methods by itself and the GUI team would just use them. Over Christmas 2001 we started to implement a prototype for the author pages to be able to test the yet implemented access layer methods. After the Christmas break finally the implementation phase

began.

9.4.4 Re-Implementation Phase

The process of implementation was guided by the phases a conference would have, because in each phase some functions had to be disabled or enabled. For example an author should not be able to upload a new paper during the review phase etc. To determine which functions are allowed in a certain conference phase the requirements team created a matrix that showed the relations between phases and functions. In the columns the phases were noted, in the lines the functions. A red square at postion (x,y) indicated that it was forbidden to use function y in phase x, a green square showed us when to enable the function. This matrix was reviewed as well and some errors were eliminated.

To enable phase-related enabling/disabling of functions a tag library was written containing the so-called "phase-case-tag". Menu entries to be used in certain phases were embedded into this tag and were now only be enabled during the related phases.

In March 2002 the GUI was in a really useful state. Only some errors had to be corrected. The user interface was presented and we agreed that not all functions could be implemented anymore because of lack of time.

So we did some debugging on the functions and improved the layout which was quite problematic because of the huge amount of data which had to be displayed in some areas (e.g. assignment of papers).

9.4.5 Ergonomical Issues

One issue during the GUI design / implementation process was to create a user interface which easy and intuitively to use. The user is able to find the functions he wants to use directly. For that reason a description of each function is given on the index page of a section. The headlines are links that directly point to that particular function to provide easy access to that function. Options that are not available at the moment are greyed out. While the user works with Chairware he should not get "stuck" in the program that means from any section in the program he should be able to reach another section without using the "Back"-Button of the web browser. After an action is finished the user will be redirected to the toplevel in the particular section or will get an error message that something has gone wrong. If the user for example forgets to provide data needed to fulfil his request the user is redirected to the form and is told what data is missing without losing the data he already provided in the form. Form elements that are required to contain data to successfully complete the action are marked with an asterisk so the user can easily see which fields have to be filled.

9.5 Access Layer

The Access Layer finally consisted out of two access points implemented as Enterprise Session Beans, about a dozen of logic classes and many Entity Beans. There was one Session Bean that was intended to be used only for the retrieval of content to be displayed on the web interface and one Session Bean especially for the processing of the input from the web forms and links to the business logic and finally the data base. This separation was chosen freely and almost randomly. Other approaches were to have only one very complex Session Bean as single entry point to the business logic from the user interface or to have several Session Beans for every user type. For several reasons (described in chapter seven), the separation of logic into internal Session Beans was dropped, but normal Java classes are used for this. As they are named by the 'stuff' they care for, their names were 'XYZS-

tuff' and all of them were subclasses of AbstractStuff, provides some methods to access other 'XYZStuff'-classes. These classes had to be tightly connected, because the functionality offered to the user interface usually infect many components and hence XYZStuff classes. The classes are ConferenceStuff, DiscussionStuff, ExtraStuff (File Management and others), MailStuff, PhaseStuff, RelationStuff, RepositoryStuff, RnRStuff (Review and Rating), TodoStuff (for the Tasklist) and UserStuff.

Two more major parts of the AccessLayer are mexplained more in detail.

9.7.1 Object-IDs

To identify the user who logged into the web application and Database objects (Entity Beans) on the web interface, we developed a mechanism that mapped these things (users are Entity Beans) to webable Strings, that we called object-ids. This simplifies the retrieval and management of objects within the access layer in cooperation with the web application very much. (These usually ends up in the value attributes of input elements in form elements in the web application.) One of the main tasks of the Access Layer is to resolve these object-ids and to delegate the request (with the real objects) to the components (if any) or process it directly within the Stuff classes.

9.5.2 CommBeans

CommBeans are Chairware only value objects for the data transmission between the Access Layer an the web application. As the implementation of the application is not based on a specification and the web presentation was not finished, the values to be tranferred between the two parts of the project were changing very often. So having a single value object for every method invocation (and another for the returned value), would have resulted in more that 150 simple bean classes, that would become inconsistent within days. CommBeans are mainly a convenient wrapper around a hash table, that provides some additional functions. For example you can drop a String object into the CommBean under a certain 'attribute-name' and retrieve the value of the very attribute as a Boolean value if you want. The saved String is parsed for it's Boolean value. This works for all primitive data types. There is a mechanism to ensure that there are no NullPointerException when using missing values from a CommBean. Another valuable function is the ability to write out the whole content of a single CommBean to the console, which we used for testing the Access Layer while the UI was not yet ready to display the output or to identify communication problems between the two applications. Needless to say, that these CommBeans have the ability to be nested into each other, which was found useful to transfer complete discussion threads. We wrote a complete Toolkit for the CommBeans with the functionality of filtering and sorting a collection of CommBeans.

9.6 The Components Review and Report

In the following section the two components Review and Report will be discussed in detail. These components are discussed together as they form a logical unit and are dependent on each other. Initially, the motivation for these components is described. Then, the focus is moved onto implementation details.

N.B.: The terms “Report” and “Overview” represent the same entity and can thus be replaced with each other.

9.6.1 Motivation

The review phase is an important phase within the conference cycle. During this phase the referees performs reviews for all papers that have been assigned to them. They are enabled to enter their review results into the system. This is done by review forms. Typically, these review forms contain fields for textual data like comments and fields for representations of numeric values as e.g. a rating score on a scale from –10 to +10 etc. Based on these single ratings an overall ranking of the papers can be calculated to simplify the papers acceptance or rejection process.

As there are different ranking systems used in real-life conferences, the structure of the review forms can vary from conference to conference. Thus, it is necessary to provide flexible review forms. They should be easily adaptable to different needs of different conferences. This implies, however, that the part of the Chairware system dealing with the ranking system has to be adaptable as well to ensure flexible support for different ranking systems.

Another important feature of a conference tool is to provide some statistical overviews about different processes during the conference for the PCC. So e.g. the progress of the bidding process can be visualised to provide the PCC a basis for reminding emails to the reviewers. Moreover, the review results entered into review forms and the calculated rankings need some kind of visualisation. For all these functionalities so called reports or overviews are used. As some of these reports are based on the adaptable ranking system it is obvious that the reports also have to be generated in a flexible manner.

9.6.2 Realisation

The design of the review and report components was driven by the need for flexibility. As the rating subsystem should be plug-able as well as the report subsystem, the actual design provides some narrow interfaces to ensure the required flexibility.

In the core of the Chairware system there are the interfaces `IRatingDescriptor`, `IOverviewFactory` and `IOverview`. These have to be implemented either for a rating system or for an overview component. Every ranking system has to implement the `IRatingDescriptor` interface and thus has to provide a rating descriptor. In other words, a rating descriptor provides all information needed to store a generic representation of the review data in the Chairware database and to enable the system to check a user given review form for its suitability for the rating system. Components providing overviews have to implement the `IOverviewFactory` interface. Each factory is used to create the different overviews it provides. All overviews are classes implementing the `IOverview` interface. This interface mainly provides functions for describing the overviews (needed for listing them in menus) and a method to generate the actual overview content. An UML class diagram of the interfaces and their relations described here is shown in figure 6 . (In this class diagram a part of the ITC-rating system and ITC overview package is also depicted.)

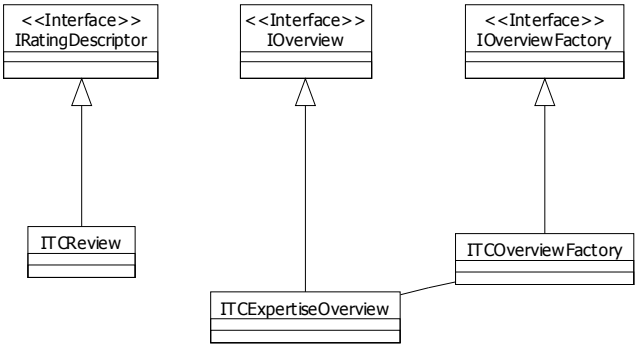


Figure 6: UML Class Diagram of Review Component

For the creation of overviews we decided to create two different packages containing overviews. Each package has its own overview factory. One overview package, called "basic overview package", provides all of the overviews showing information independent from the rating system. E.g., one overview of this type is a bidding report showing all reviewers and their bidding data. The second package, the so-called ITC package, comes with a rating descriptor and implements the rating system "Identify the Champion" [22]. This package also contains an overview factory providing additional, rating system dependent overviews.

To specify the actual layout of a review form we decided to use an XML representation. This representation contains several keywords that refer to elements in the review form as edit fields for comments, checkboxes for numerical values, graphical elements or describing text elements. Each element for input data has an unique identifier. These identifiers are used to assure that the review form template fits to the installed rating system. The default review template is depicted in Figure 7.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<review-form>
  <text-simple>
    Rate this paper by filling in the following fields. Please take care
    of the different types of review comments as they are sent to the
    indicated persons automatically.
  </text-simple>
  <radio value-id="ITC_QUALITY">
    <question>
      Paper Quality
    </question>
    <description>
      Please rate the overall quality of the paper.
    </description>
  </radio>
  <line />
  <text-area value-id="COMMENT_PC">
    <question>
      Review Comment for the Programme Committee
    </question>
    <description>
      This comment will be sent to all members of the programme committee
      assigned to this paper and the PCC.
    </description>
    <content>
      Please paste your comment here.
    </content>
  </text-area>
  <line />
  <text-area value-id="COMMENT_AUTHOR">
    <publish-author />
    <question>
      Review Comment for the Paper's Author
    </question>
    <description>
      This comment will be sent to the author of the paper when the
      decision of acceptance or rejection of this paper has been made.
    </description>
    <content>
      Please paste your comment here.
    </content>
  </text-area>
  <line />
  <radio value-id="ITC_EXPERTISE">
    <question>
      Your Personal Expertise
    </question>
    <description>
      Please indicate your personal expertise for this paper.
    </description>
  </radio>
  <line />
  <text-simple>
    Thank you very much for your cooperation. And please, do not forget
    to press the "Ok"-Button. ;-)
  </text-simple>
</review-form>
```

Figure 7: The default review template

At loading time – when the review form description is converted into a review form - this description is checked with the data from the review descriptor. Information for mandatory fields within the form is provided by the review descriptor. These details are retrieved from the descriptor and inserted into the XML description. An example: in Figure 8 there is a radio button field with the identifier "ITC_QUALITY". In the description only the information where this question will appear is provided in the review form (besides the question text). The information that "ITC_QUALITY" is a radio button field with four possibilities from "A" to "D" is retrieved from the rating descriptor.

If no review data for a review is stored in the database (i.e. the reviewer has not yet entered any review information) when a review form is to be displayed, preload values for the review form fields can also be read from the

rating descriptor. In the other case, the values entered earlier are added to the XML representation of the review form. So a complete XML document is created containing all information for the review form. Then, a XSLT transforms the XML data into HTML code to provide the actual form for the user. Figure 8 shows the XML review form rendered as HTML.

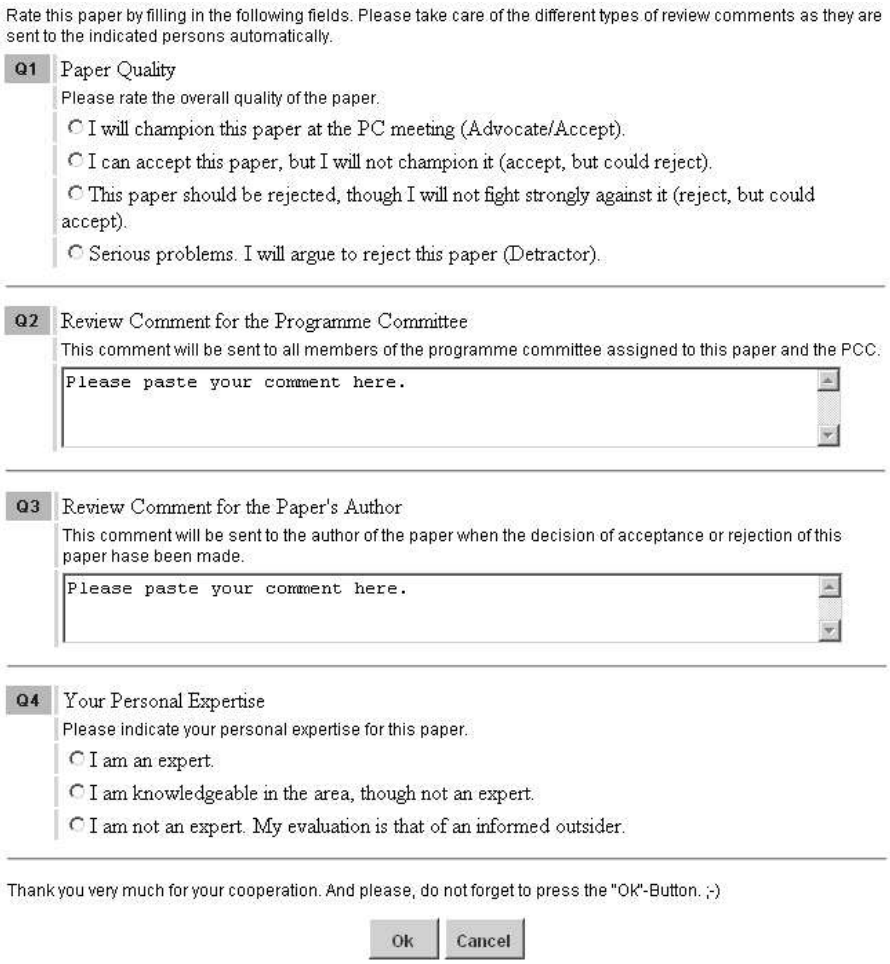
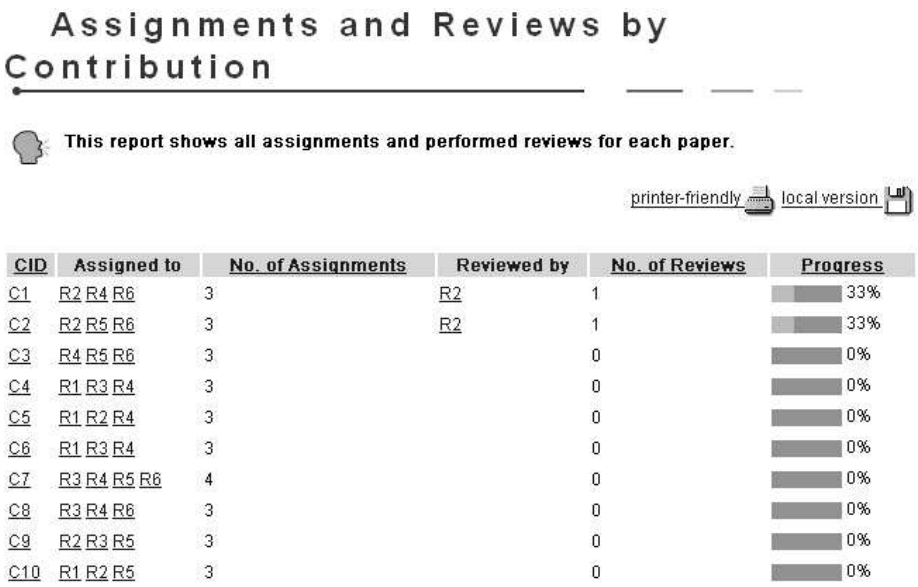


Figure 8: Sample Review Form Description Rendered by System

Generating XML and transforming it into HTML is also used when the reports are generated. Every overview has to implement the `IOverview` interface and thus provides a method that returns a complete XML document representing the report. Using an XSLT, these reports are transformed into HTML code and displayed for the user as shown in figure 9.



Assignments and Reviews by Contribution

This report shows all assignments and performed reviews for each paper.

printer-friendly local version

CID	Assigned to	No. of Assignments	Reviewed by	No. of Reviews	Progress
C1	R2 R4 R6	3	R2	1	33%
C2	R2 R5 R6	3	R2	1	33%
C3	R4 R5 R6	3		0	0%
C4	R1 R3 R4	3		0	0%
C5	R1 R2 R4	3		0	0%
C6	R1 R3 R4	3		0	0%
C7	R3 R4 R5 R6	4		0	0%
C8	R3 R4 R6	3		0	0%
C9	R2 R3 R5	3		0	0%
C10	R1 R2 R5	3		0	0%

Figure 9: A Sample Overview

The advantage of this approach is, that it is rather easy to provide two versions of each overview: one version is shown in the Chairware GUI and makes use of hyperlinks and buttons etc. Another version is more printer-friendly with less formatting and colour information. This printer-friendly version is generated by applying a second XSLT on the overview instead of the first one.

10 Testing Phase

10.1 Testing Procedure

Testing was a further phase during our project work. It was delayed because of the never-ending implementation phase. The testing manager and the quality manager started to create a test document containing the different tests: component tests [24], integration tests and product tests [25].

10.1.1 Component Test

The back-end was tested with component tests. The people who coded the component had to test it, too. Every non trivial method of every single class was tested. Because the implementation of the access layer wasn't finished by then, drivers had to be written. An excerpt of the component test is shown in Figure 10.

Component: activationkey		Responsible: Martin Ernst		
Class/Bean: org.chairware.server.activationkey.session.ActivationKeyFactory		CVS Version: 1.3	Date: 20.12.01	
Methods:		Passed	Failed	Not Tested
1	getNewActivationKey():String	X		
2	checkActivationKey(String):boolean	X		
3	removeActivationKey(String):void	X		
Class/Bean: org.chairware.server.activationkey.entity.ActivationKey		CVS Version:1.2	Date:20.12.01	
Methods:		Passed	Failed	Not Tested
1	getKey():String	X		
2	create(String):ActivationKey	X		

Figure 10: Excerpt of the component test template

10.1.2 Integration Test

The integration of the backend, the access layer and the web interface was done during the implementation phase. Every time a new component was added to the project the whole software was compiled again to see if the integrated component worked or not. There should be a integration test document but because of the lack of

time during the project there isn't.

10.1.3 The product test

The scope of the product test was to test all functional requirements of the software. The requirements list was checked with our "Baustellenliste", a list of what still had to be done to fulfil all requirements of our requirements list. The tester only had to test if the functional requirements work. Every single interaction with the web interface is included in this document with extra information of what should happen and what really happens. The requirements were only fulfilled, if these two criteria fit for every interaction. An excerpt of the product test document is shown in Figure 11.

Author:

Participate at the conference:

1. An Author wants to participate at the conference. Therefore he has to click on the link: submit paper and create new account.
   What shall happen?
   The Author should get to the "Submit paper and create new account" page.
   What happens?
   And so he does.
2. The Author has to fullfill three steps. Fill in the form correctly.
   What shall happen?
   If the Author fills in the form correctly he should be logged in and get to the Author main section page.
   What happens?
   The Author logs in automatically anf gets to his main section.

Figure 11: Excerpt of the product test template

10.2 Results

During our project testing was not as important as it should be because the end of the implementation phase was postponed again and again. No one was interested in a proper way of testing so everyone tested his code with the trial & error principle. Another disadvantage of the never-ending implementation phase was that every time the web interface was changed, the product test had to be changed, too. That caused a lot of additional work.

The main problem during the testing phase was, that there was no testing phase. After the end of the implementation phase there was a code freeze and the official presentation. After that there was no need for testing anymore. The product test was finished after the implementation phase but it had no effect on the project because nearly no one read it and no one changed the found out errors of the software.

For future projects there is only one solution: Testing must be motivated more.

11 Project Plan

This chapter depicts the project plan of the Chairware software project in a chronological order. Initially, the project plan is described. Then, the experiences and conclusions sum up the lessons learned from the project plan and its realisation.

11.1 From the Model to the Project Plan

The project plan [15] shows the planned processes within a software development project and is generated by rendering a software process model more precisely. In the case of Chairware, the actual project plan is partially based on the waterfall model. The waterfall model is a very simple model, easy to understand and use. This model consists of set of phases that a project progresses through in a sequential order. Each phase must be finished before the project can progress to the next phase. At the end of each phase there is some form of gate-way, normally a formal review where the decision if a phase has been completed is made. We used this model for phases as collecting requirements, the object oriented analysis (OOA), the object oriented design (OOD), the implementation phase and the testing phase. Otherwise, several phases of the project were in parallel to other phases extending the part of the plan based on the waterfall model. These parallel phases not fitting in a water-fall-style project are the research activities regarding security aspects, the creation of prototypes and writing the intermediate report and the final report.

The project plan was updated regularly and reviews of the project plan were performed approximately all four weeks during the project. Several milestones have been set up to divide the project plan into handsome parts. These milestones and how they could be reached are described within the next section.

11.2 Milestones and Achievements

The first milestone of the project plan, the completion of the requirements collection, has been reached in Dort-mund with short delay. Unfortunately, the resulting requirements document could not be merged with the Mel-bourne Software Requirements Specification until the visit of the students from Melbourne in Dortmund.

The second milestone was the completion of the Object Oriented Analysis. This phase was delayed for two weeks, caused by the fact that the requirements' review process with the Australian sub-team did not make fast pro-gress. Finally, this phase could be finished by June 22, 2001.

As a visit from Melbourne team members was scheduled from June 23 to July 7, 2001, the project plan was adapted to fit this into the plan. During this visit, the main tasks were reviewing the OOA, performing the OOD and splitting of the work onto the two sub-teams.

The third milestone – finishing the Object Oriented Design – was initially planned to be reached by July 10, 2001. Due to the delay in the OOA and due to the fact that the OOD could not be completed during the visit of the Melbourne sub-team members, this deadline had to be extended. A consenting version of the OOD we and the

Melbourne sub-team members could agree upon was finished on August 7, 2001. On this day we made a code freeze of the OOD to provide a basis for implementation planning in Melbourne.

During the semester break in summer 2001 we continued finalizing the design of the components assigned to the Dortmund team. Also, we implemented first components that should be integrated during the visit of some of our team members in Melbourne.

In September 2001 three students and one supervisor of our team visited Melbourne. During their visit they performed integration tasks and tried to enhance the communication between the two sub teams. One goal of their visit – to get a detailed assessment of the implementation status of the Melbourne sub-team – could not fully be reached.

As our team members in Melbourne could not clear how far the implementation work in Melbourne has gone we had to perform a detailed review of the Australian code in order to create a implementation plan.

After two weeks of examination we set up the project plan and scheduled the implementation phase to be finished by December 21, 2001. Four weeks of integration and testing were planned to follow the implementation phase. As the implementation in Germany did not make fast progress, the implementation phase actually ended before presentation in May 2002. In fact, there was no real testing phase as initially planned. However, the system has undergone several tests during implementation. Many integration tests were performed, but there were no planned integration tests nor a large scale product test before the final presentation.

11.3 Experiences and Conclusions

11.3.1 Synchronizing with the Australian Project Plan

Obviously, in a distributed development project there should be one unique project plan for the whole team. As the to project management sub-teams never had real conversation about how to conduct the project, two different project plans co-existed. In the Melbourne plan our milestones and deadlines had been entered, but not been updated regularly. The Melbourne dates had not been entered into our project plan but we tried to integrate their plan in ours.

Of course it would be helpful to have one unique project plan, but in fact it was no real problem that two plans co-existed. The major aspect is, that the two single project plans are synchronized properly. This means, that if the project manager is aware of the deadlines of the other sub-team there is no real need for a unique project plan. Furthermore, there might be additional difficulties in setting up this common plan Chairware did not run in as there have been two separate plans.

11.3.2 Work in Our Team

A problem that occurred rather often was the difficulty to settle an appointment for a meeting of a sub team if more than three or four people were involved. If for an action there had been a time slot of one week for completion, the team members often only were able to arrange a meeting just one day before the completion date. Sometimes it was almost impossible to get actions by larger sub-teams done in shorter periods of time than one week.

Obviously, the problem with meetings of sub-teams cannot be solved for student projects. In real world situations

this problem would not occur, as people would have to be present every day.

Unfortunately, the project plan was no big help during the second semester of the project group work. It became clear with the beginning of the implementation that the project plan did not guide through the project. On the contrary, the project plan had to be adjusted for every review and the implementation and testing deadlines had to be postponed regularly. So it was no surprise that the project group could not finish their work by the initially planned deadline of mid-February 2002.

During the implementation phase we were faced with the additional problem that the project plan as well as a coarse grained implementation plan were not sufficient to ensure that the implementation work made the necessary progress. Thus a fine grained "ToDo-List" was set up and updated weekly from February 2002 onwards. In this list all implementation activities were described in detail and assigned to one or multiple persons. The fulfilment of these tasks was interrogated during the weekly project group meetings. Using this approach we could ensure that a status of the implementation progress was at least available once every week and get the implementation work done.

11.4 The Project Plan

Figure 12 shows the project plan in its version of May 6, 2002. For each phase the activities are listed together with their starting dates and their durations. The chart displays the temporal order of the activities combined with their dependencies by indicating the predecessors and the successors of any activity. Next to the bars, role abbreviations indicate which team members are involved in the activities.

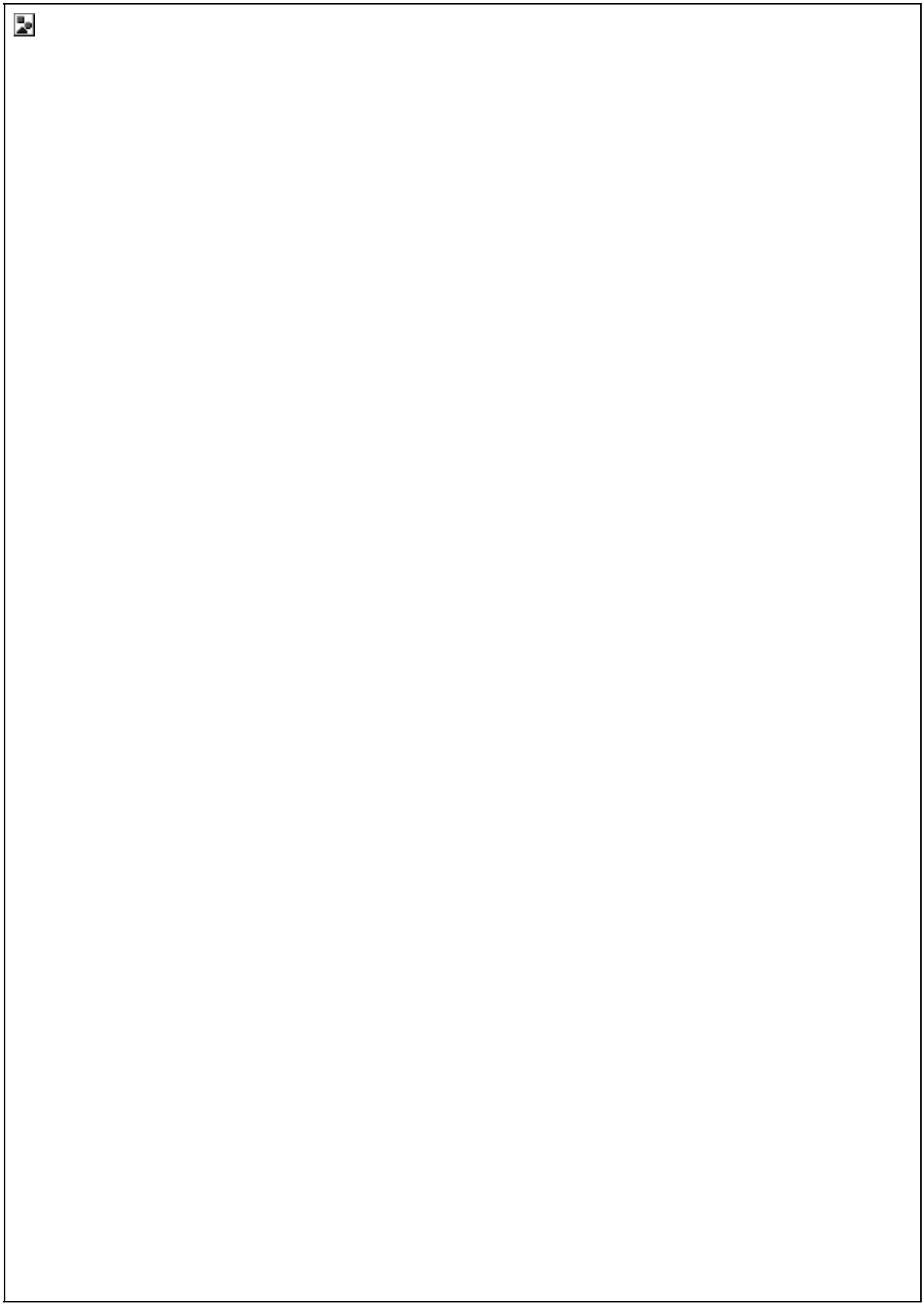


Figure 12: Project Plan as of May 6, 2002

12 Hardware and Software

12.1 Introduction

Developing software is a complex task. For that reason it is very important to design a system architecture that fulfils all needs of the certain software project. The following chapter describes the work environment used by the project group (PG) Chairware.

The chapter is divided into two major parts:

- **Hardware configuration**
- **Software environment (Server/Workstation)**

12.2 Hardware

The hardware used by Chairware was provided by the Chair of Software Technology, Department of Computer Science, of the University of Dortmund (LS 10). All computers are in a separate room at the University. That room is only accessible for members of Chairware, which is a great advantage because the working environment could be configured to meet the requirements of the group without considering the needs of other students who do not belong to the project.

All computer systems used for Chairware are integrated into a Local Area Network (LAN). The network topology is based on the client/server architecture and therefore is divided into servers and workstations. Some PCs are equipped with 100 MBit network interface cards (NIC) while others still use slower 10 MBit connections. The names and IP-Addresses of the computers match the defaults for the network at LS10, so the use of existing resources like printers and the SMTP server is possible without further configuration. The LAN can be divided into two major parts. The first part is the domain Chairware, the second part is a stand-alone Linux server that operates as web- and CVS server. Sure all functionality of both servers could be merged into one Linux server, but one requirement for the cooperation with Australia was that the application server should run under Windows NT. Due to the fact a CVS server is best set up on a Linux machine we decided to use an independent server for that task.

In the entire domain one PC (ls10pc29) is dedicated to be the Primary Domain Controller (PDC). On this server the user profiles are stored so each user gets his own customisable workspace no matter on which workstation he is logged on. The other server (ls10pc12) is not a member of the domain Chairware. It provides services for the web server and the CVS system. The following listing gives an overview about the deployed hardware components.

Name:ls10pc29

Usage: Server (Primary Domain Controller)

Processor: Dual Pentium II 450 MHz

RAM: 256 MB

Hard disk: 19 GB

Operating System: Windows NT 4.0 Server

Name:ls10pc12

Usage: Server (CVS, Web server)

Processor: Dual Pentium Pro 200 MHz

RAM: 128 MB

Hard disk: 8GB IDE

Operating System: Red Hat Linux 7.1

Name:ls10pc43

Usage: Workstation

Processor: Pentium II 450 MHz

RAM: 256 MB

Hard disk: 12 GB IDE

Operating System: Windows NT 4.0 Workstation

Name: ls10pc28

Usage: Workstation

Processor: Duron 700 MHz

RAM: 256 MB

Hard disk: 2 GB SCSI

Operating System: Windows NT 4.0 Workstation

Name: ls10pc44

Usage: Workstation

Processor: Pentium III 600 MHz

RAM: 256 MB

Hard disk: 19 GB IDE

Operating System: Windows NT 4.0 Workstation

Name: ls10pc45

Usage: Workstation

Processor: Pentium III 600 MHz

RAM: 256 MB

Hard disk: 10 GB IDE

Operating System: Windows NT 4.0 Workstation

Name: ls10pc28

Usage: Workstation

Processor: AMD Duron 750

RAM: 512 MB

Hard disk: 10 GB

Operating System: Windows NT 4.0 Workstation

Name: ls10pc53

Usage: Workstation

Processor: AMD Duron 750

RAM: 512 MB

Hard disk: 19 GB

Operating System: Windows NT 4.0 Workstation

Summarizing the equipment provided by the University of Dortmund fulfils the need of the project.

12.3 Server: Operating Systems and Services

We have two servers, a Windows server and a Linux server. Although one server runs Red Hat Linux 7.1 as operating system the domain Chairware can be declared as a homogeneous environment, because the Linux server is not entirely a member of that domain.

The stand-alone server provides different services. The most important one is the CVS server. If different people will work on the same parts of the submission tool, it is necessary to ensure that older programs will not overwrite new versions. This functionality is one of the most important tasks performed by the CVS server. This service is accessible via Internet so the Melbourne team is able to work in the Dortmund CVS repository as well.

The PG Chairware runs its own website at the top-level domain http://www.chair-ware.de. The web pages available at this location are made accessible by the HTTP service, which is included in the apache web server package. The web server gives the PG Chairware the option to present the project to people not attending to the University of Dortmund.

Sometimes it is necessary to access data stored on the Linux server via internet access. For this reason the server

provides an SSH-service for secure remote access.

The Primary Domain Controller (ls10pc29) is the central part of the domain Chairware. This server allows to save the user profiles and project data at a centralized location. Each user is able to work in his/her own customisable workspace after logging into the system, no matter which workstation he chooses (roaming profiles). The server allocates a home directory to each user. This option of centralized storage of data enables total mobility to the user and makes it easier to backup sensitive files to prevent loss of data due to a server crash.

For the task of storing user data it would be advantageous to have a database running that handles the requests of a Java program. We installed a database server on the PDC ls10pc29 that uses the package mySQL which performance is good for low to medium ranged amount of data. Unfortunately mySQL had problems with doing transactions involving the Orion application server. The Melbourne team used the database Hypersonic, which is a testified standard for the Orion. In Melbourne no problems occurred while running this database so we switched to this one as well.

For the project we need a program to provide the business logic for our application. This is the Orion Application server, installed on the PC ls10pc29 as well. The application server also manages the database access of the Java program.

12.4 Clients: Applications

There are two major groups of applications installed and configured on the workstations. The first and most important group cintains the tools that are needed to develop the application. The other tools are not directly involved in the developing process. They are mainly used for documentation, research and communication. The following sub-sections are a short-listing of the tools used for development.

12.4.1 Sun JDK1.3 SDK & Enterprise Edition and Forte 2.0 (3.0)

For the project Chairware we chose Java for implementing the application. Java is a modern, object-oriented-programming language that makes developing easier due to its extensive libraries. Java is very suitable especially for programming distributed applications, and applications based on internet technology.

JDK contains libraries and the basic development tools. The Enterprise tools are an add-on for JDK and contain libraries needed to implement applications using Enterprise Java Beans (EJB).

A more comfortable way of building a Java application is provided by Forte. Forte is a graphical IDE that includes a Graphical User Interface (GUI) builder for Java. Unfortunately Forte's requirements regarding RAM and CPU power are very extensive.

All these applications (JDK including EJB support, Forte) are freely available from Sun.

12.4.2 Together/J 4.2 (5.5)

In addition to the programming language a well-fitted modelling language to support the design phase is essential. We chose UML which is the standard language for that requirement. As a tool for UML we use Together/J 4.2. Together is highly customisable and covers all diagram types of UML. Furthermore Together provides "round-trip engineering". It is possible to generate Java source code from UML diagrams and vice versa.

Together/J is licensed to the Department of Software Technology of the University of Dortmund.

12.4.3 WinCVS

This program was installed to grant comfortable access to the CVS service provided by the server. It allows working in the CVS repository via a graphical user interface like the windows file manager. WinCVS is freely available.

12.4.4 Microsoft Visual Studio 6.0

One of the requirements of our project is to modify the Java Virtual Machine (JVM) to provide security policies that are specially fitted to our submission tool. The JVM is written in C, so we needed a C-compiler for that work. Visual Studio is only installed on a single workstation (ls10pc44).

Microsoft Visual Studio is licensed to the Department of Software Technology of the University of Dortmund.

Besides the tools used for development, further applications have been installed:

12.4.5 Word 2000, PowerPoint 2000, Outlook 2000

During the project different documents, like the requirements document, have to be created. Furthermore foils for some presentations have to be prepared as well. For that tasks many products are available. Due to the cooperation with Melbourne, that had the requirement to use Microsoft Office, we decided to use it as well, although MS Word has some major problems in handling large documents. This could be a handicap while writing the summary report for our project.

The most used type of communication used between Dortmund and Melbourne is e-mail. For that reason we have e-mail accounts on the mail server of LS10. As a tool for managing e-mail, we chose Outlook 2000. It offers many options for delivering and organizing mail.

All applications mentioned above are licensed to the Department of Software Technology of the University of Dortmund.

12.4.6 Babylon Translator

All documents we created were in English. Furthermore literature provided in the internet are often in English as well. Babylon features a comfortable way of translating German words into English and vice versa with just one mouse click.

Babylon is freely distributed.

12.4.7 Acrobat Reader 5.0

Some documents are only available in PDF-format. To view these documents the Acrobat Reader 5.0 had been installed.

Acrobat Reader is freely distributed by Adobe.

12.4.8 Text Pad 4

Especially for the Orion Application server it is necessary to edit XML files by hand. For that work an editor more powerful than the Windows notepad is needed. Text Pad 4 features advanced editing functions and syntax high-

lighting for numerous document types. Text Pad is shareware but can be used for free.

12.4.9 Easy Zip

Easy Zip provides a graphical user interface for the well known zip command. This is quite useful, for all documents that are received should be zipped before transmission.

Easy Zip is freeware.

12.4.10 Inoculate IT Personal Edition

In computer networks a virus is distributed very fast which may result in loss of data etc. For that reason it is recommended to have some background task running to identify and eliminate viruses.

Inoculate is available for free after an online registration.

12.4.11 Windows Commander

Windows Commander is a file manager that offers lots of features such as a comfortable ftp-client, a built in zip tool etc. Primarily the ftp-client was interesting for our project.

Windows Commander is shareware but can be used without registering. The user has to close some information about registration at each start up.

12.4.12 Internet Explorer 5.5

The Internet is one of the most important resources for gathering information. Especially about Java there are lots and lots of vaults providing information. Additionally, IE 5.5 provides a mail client that can be used as alternative for Outlook 2000.

Internet Explorer 5.5 is freeware.

12.5 Installation

The set-up phase was executed in two tasks. At first the two servers (ls10pc12, ls10pc29) and one of the workstations were manually configured. In this phase the operating systems had been installed and configured first. Afterwards the needed applications were installed and tested one by one. A backup of the running system was made to duplicate it on the other workstations.

In the second phase the ready to use configuration was copied to the remaining workstations. Norton Ghost was used to write the configuration of the installed workstation into an image file. This file was copied to the Windows server on which the Norton Ghost Multicast Server had been temporarily installed. This server program allows installing workstations via a LAN from an image file. Afterwards the workstations had to be booted from a specially created boot disk providing network functionality. The configuration was copied to the workstation which only took us about half an hour per computer instead of several hours. At last the configuration had to be modified to fit the requirements of the built-in hardware. The most important step was changing the security ID of the workstation. That ID is necessary for integrating the computer into a Windows NT domain. During the integration

process the server checks the entire domain for the security ID of the applying client. If another PC has the same ID the new client will be rejected. For the individual configuration process more time was used than for the task of cloning the workstation. This resulted from several necessary reboots of the clients. If all PCs had the same hardware components a lot of time could have been saved. Summarising one could say that there was a huge time saving in contradiction of installing all workstations manually. One PC however had to be installed the old-fashioned way, because its SCSI hard disk produces several problems.

12.6 Software Updates

During the period of a year it is normal that new versions of the programs and tools used will be released by the vendors. One should think very carefully about updating your "old" software because it does not mean that a new version of a program is always better than the old one. During our project new Versions of Together/J (Version 5.5) and Forte 4 Java (Version 3.0) were released. The vendors promised that some serious bugs had been eliminated in these versions so we gave them a try and tested them.

Both applications were first installed just on one machine so we could test the new version but still had the "old" well known versions available.

12.6.1 Forte 4 J:

Forte was very easy to install. We just had to uninstall the old version and install the new one. After a week of testing we found the new version worth installing on all machines.

12.6.2 Together/J:

With the new version of Together/J a new license model had been unleashed as well. The old system of just copying a license file into the program directory had been replaced by a quite complex task of setting up a "license server" to authenticate that we were allowed to use the program. A licence server for the whole department of software engineering (LS 10) of the University of Dortmund was set up by their system administrator which we were allowed to use. After the installation of Together we had to edit a configuration file manually to enable the access to the licence server, which worked fine. After some period of testing Together/J 5.5 was installed on all machines.

13 Summary

This chapter shall provide a short retrospective summary about the project group, its work and a prospect to the further development. In contrast to the other chapters, this chapter tells about a rather subjective point of view.

The work of the PG can be summarised as a success because

- A great part of the requirements set at the beginning of the two semesters work have been implemented into a running final product.
- The proceeding of the work was in all in all systematic (although not strictly formal and not strictly defined in advance).
- There was a good mix of the goals of a lecture and the constraints you can find in real software development projects. So all participants of the PG have made new experiences and acquired new skills in their roles ranging from project planning, preparing and moderating meetings over holding seminars to technical knowledge which will certainly valuable even after the end of the PG.

In spite of the overall positive appraisal of the course of a project group the imponderableness of the work should not be kept secret which are inseparable connected with such long-term projects. This is done last but not least to provide helpful hints to other project groups for their work and preparations.

The cooperation with the Swinburne University of Technology was not as expected because of several reasons:

- The two groups have had too different ways of working.
- Caused by the different time zones the communication was asynchronous which caused delayed decisions.
- There was no central co-ordinator co-ordinating the distributed teams and keeping care that the teams do not develop incompatible conceivabilities.

Anyhow it was a valuable experience for both teams.

Looking into the future it remains to hope that the work and its result will provide further positive impulses. Be it by the allocation of diploma thesis or by publishing articles about interesting aspects of the work done by this PG.

It also remains to hope that the friendly relationships developed in the course of the PG will persist over the end of the project.

The complete Chairware-Team wishes all PGs to come a project group work as funny and successful as ours.

14 Glossary and Definitions

Abstract

The abstract is a short description of a paper that is submitted by an author who wishes to contribute a paper to a conference (before the paper is actually submitted for review.) The programme committee uses the abstracts to determine what authors will be chosen to submit papers to the conference for review. Henceforth, when an author submits his or her paper they do not have to enter an abstract again.

ACM

Association for Computing Machinery. Founded in 1947, ACM is the world's first educational and scientific computing society.

AD

Architectural Design, design of the architecture used in the system, e.g. a 3-tier architecture

Application Server

A server application that provides a playground for EJBs to exist in. It also provides common functionality required by server-side components.

CBD

Component Based Development

Conference Phases

The different phases a conference can be in. They are not exclusive, i.e. many phases can run simultaneously.

1. 'Call for Papers'
   Authors are asked to write a paper for the conference.
2. 'Submission of Abstracts'
   Authors submit the abstract(s) of the paper(s) they intend to contribute.
3. 'Bidding'
   Reviewers have the possibility to give their preferences of what paper(s) they want to review.
4. 'Submission of Papers'
   Authors submit their completed paper(s)
5. 'Assignment'
   Based on the bidding, the PCC assigns papers to the different reviewers.
6. 'Review'
   The reviewers rate the papers assigned to them.
7. 'Discussion'
   If there are serious rating conflicts, a discussion between the corresponding reviewers may clarify misunderstandings and help coming to a common agreement.

8. 'Camera Ready Copies'
   Authors submit a brushed up version of their paper to be published.

Conflict

There are different kinds of conflicts that may occur.

1. 'Rating Conflict'
   The review results of one paper show a huge gap between them. Further investigations may be suitable.
2. 'ACM Conflict'
   A conflict between a reviewer and a paper/Author as defined by the ACM [29] rules.

CVS

A CVS (Concurrent Version System) is used to maintain and control any file that is part of the software development effort. It maintains versions of documents or files and can be used to maintain source code for a project as well as any other relevant files.

DD

Detailed Design, design of the components themselves

ECMS

Electronic Conference Management System

EJB

Enterprise Java Bean. There three different kinds in the EJB 2.0 specification.

1. Entity Beans
2. Session Beans
3. Message Driven Beans

EJB-Server

see 'Application Server'

EST

Electronic Submission Tool

i18n

Internationalisation, the process of designing an application so that it can be adapted to various languages and regions without engineering changes.

IDE

Integrated Development Environment

JSP

Java Server Page

JVM

Java Virtual Machine. JVM is the environment under which all Java applications run.

MDB

Message Driven Bean, see EJB.

OOA

Object Oriented Analysis, one of the development phases of the project.

OOD

Object Oriented Design, one of the development phases of the project.

Overview/Report

A report will be generated from data stored in the system that actually relates to a conference, e.g. summary of papers submitted. It does not include any information in regards to the running of the system, e.g. low level information needed to maintain the system.

PCC

Programme Committee Chair

PDC

Primary Domain Controller, a server that is used for authenticating the users and providing home directories.

PM

Project Manager

PP

Project Plan

Programme

A programme is a schedule for a conference. This would outline the events that are to take place at a conference, where they are to take place, who would attend, etc.

QM, QAM

Quality (Assurance) Manager

Referee

same meaning as reviewer, often used term in the SRS

Reviewer

Member of the programme committee. Person to review submitted papers in order to rate them.

RL

Requirements List, document containing all functional requirements of the Chairware tool

RM

Requirements Manager

SA

System Architect

SM

Security Manager

SQAP

Software Quality Assurance Plan

SRS

Software Requirements Specification

TP

Testing Plan, used to conduct the overall test of all components

UM

Usability Manager

Waterfall Model

The 'waterfall model' [23] is a simple software development process model.

15 Literature

This is only the literature we used in this report. We also used other literature which you can find in the intermediate report of the PG Chairware[23]

[1] Bühren, Thomas, Konfiguration und Durchsetzung von Sicherheitsspezifikationen mit Hilfe einer erweiterten Java-Laufzeitumgebung, Dezember 2000

[2] Röpling, Ingo, Application Server, Dortmund, 2001,
CVS-pathname `\chairware\seminar\Application Server\`

[3] Schäfer, Clemens, GUI Design Using Forte, Dortmund, 2001,
CVS-pathname `\chairware\seminar\GUI and Forte\`

[4] Alvermann, Markus, UML mit Together, Dortmund, 2001,
CVS-pathname `\chairware\seminar\UML with Together\`

[5] Schreier, Nikolai, Security, Dortmund, 2001,
CVS-pathname `\chairware\seminar\Access- & Information Flow Control\`

[6] Shtern, Olga, Configurable Security in Java, Dortmund, 2001,
CVS-pathname `\chairware\seminar\Security\`

[7] Flatt, Tamara, BSCW und CVS, Dortmund, 2001,
CVS-pathname `\chairware\seminar\bscw\`

[8] Helmig, Urs, Enterprise Java Beans, Dortmund, 2001,
CVS-pathname `\chairware\seminar\Enterprise Java Beans\`

[9] Langer, Thorsten, CyberChair, Dortmund, 2001,
CVS-pathname `\chairware\seminar\CyberChair\`

[10] Ernst, Martin, Rollenbasierte Software-Entwicklung, Dortmund, 2001,
CVS-pathname `\chairware\seminar\rolebased software-development\`

[11] PG Chairware, Requirements List, Dortmund, 2001,
CVS-pathname `chairware\dokumentation\Dortmund Docs\RL document.doc`

[12] PG Chairware, Software Requirements Specification, Melbourne, 2001,
CVS-pathname `chairware\requirements\doc-view\SRS docs\Melbourne Docs\srs.doc`

[13] PG Chairware, OOA diagrams, Dortmund, 2001,
CVS-pathname `chairware\together\OOA Arbeitsversion.tpr`

[14] PG Chairware, OOD diagrams, Dortmund, 2001,
CVS-pathname `chairware\together OOD *\OOA-OOD.tpr`

[15] PG Chairware, Project Plan, Dortmund, 2001,
CVS-pathname `\chairware\project plan\`

[16] PG Chairware, SQAP, Melbourne, 2001,
CVS-pathname `\chairware\qa\SQAP.doc`

[17] PG Chairware, Interview Doug Grant, Melbourne, 2001,
CVS-pathname `\chairware\requirements\doc-view\Questionairres – Interviews\interview_douggrant.doc`

[18] Volere Requirements Template,
`http://www.atlsysguild.com/GuildSite/Robs/Template.html`

[19] Gruhn, Volker, Software-Technologie I, Dortmund, WS 1997/98,
`ftp://ls10-www.cs.uni-dortmund.de/pub/windows/folien_gruhn/ws98-99/vorl98-final-frs-www.ppt.gz`

[20] CyberChair, `http://www.CyberChair.org/`

[21] PG Chairware, Role Descriptions, Dortmund, 2001
CVS-pathname `\chairware\dokumentation\Role Descriptions\`

[22] Nierstrasz, Oscar: Identify the Champion, An Organisational Pattern Language for Programme Committees, from Pattern Languages of Program Design 4,
N. Harrison, B. Foote, H. Rohnert (Ed.), vol. 4, Addison Wesley, 2000, pp. 539-556

[23] PG Chairware, Intermediate report, Dortmund, 2001
CVS-pathname \chairware\dokumentation\DortmundDocs\

[24] PG Chairware, Component test, Dortmund, 2001
CVS-pathname \chairware\dokumentation\DortmundDocs\

[25] PG Chairware, Product test, Dortmund, 2001
CVS-pathname \chairware\dokumentation\DortmundDocs\

/99/ T. Bühren, M. Cakir, E. Can, A. Dombrowski, G. Geist, V. Gruhn, M. Gürgrn, S. Handschumacher, M. Heller, C. Lüer, D. Peters, G. Vollmer, U. Wellen, J. von Werne
Endbericht der Projektgruppe eCCo (PG 315)
Electronic Commerce in der Versicherungsbranche
Beispielhafte Unterstützung verteilter Geschäftsprozesse
Februar 1999

/100/ A. Fronk, J. Pleumann,
Der DoDL-Compiler
August 1999

/101/ K. Alfert, E.-E. Doberkat, C. Kopka
Towards Constructing a Flexible Multimedia Environment for Teaching the History of Art
September 1999

/102/ E.-E. Doberkat
An Note on a Categorial Semantics for ER-Models
November 1999

/103/ Christoph Begall, Matthias Dorka, Adil Kassabi, Wilhelm Leibel, Sebastian Linz, Sascha Lüdecke, Andreas Schröder, Jens Schröder, Sebastian Schütte, Thomas Sparenberg, Christian Stücke, Martin Uebing, Klaus Alfert, Alexander Fronk, Ernst-Erich Doberkat
Abschlußbericht der Projektgruppe PG-HEU (326)
Oktober 1999

/104/ Corina Kopka
Ein Vorgehensmodell für die Entwicklung multimedialer Lernsysteme
März 2000

/105/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Zwischenbericht der Projektgruppe IPSI
April 2000

/106/ Ernst-Erich Doberkat
Die Hofzwerge — Ein kurzes Tutorium zur objektorientierten Modellierung
September 2000

/107/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
November 2000

/108/ Stefan Austen, Wahid Bashirazad, Matthais Book, Traugott Dittmann, Bernhard Flechtker, Hassan Ghane, Stefan Göbel, Chris Haase, Christian Leifkes, Martin Mocker, Stefan Puls, Carsten Seidel, Volker Gruhn, Lothar Schöpe, Ursula Wellen
Endbericht der Projektgruppe IPSI
Februar 2001

/109/ Leonid Abelev, Carsten Brockmann, Pedro Calado, Michael Damatow, Michael Heinrichs, Oliver Kowalke, Daniel Link, Holger Lümkemann, Thorsten Niedzwetzki, Martin Otten, Michael Rittinghaus, Gerrit Rothmaier
Volker Gruhn, Ursula Wellen
Zwischenbericht der Projektgruppe Palermo
Februar 2001

/110/ Eugenio G. Omodeo, Ernst-Erich Doberkat
Algebraic semantics of ER-models from the standpoint of map calculus.
Part I: Static view
März 2001

/111/ Ernst-Erich Doberkat
An Architecture for a System of Mobile Agents
März 2001

/112/ Corina Kopka, Ursula Wellen
Development of a Software Production Process Model for Multimedia CAL Systems by Applying Process Landscaping
April 2001

/113/ Ernst-Erich Doberkat
The Converse of a Probabilistic Relation
June 2001

/114/ Ernst-Erich Doberkat, Eugenio G. Omodeo
Algebraic semantics of ER-models in the context of the calculus of relations.
Part II: Dynamic view
Juli 2001

/115/ Volker Gruhn, Lothar Schöpe (Eds.)
Unterstützung von verteilten Softwareentwicklungsprozessen durch integrierte Planungs-, Workflow- und Groupware-Ansätze
September 2001

/116/ Ernst-Erich Doberkat
The Demonic Product of Probabilistic Relations
September 2001

/117/ Klaus Alfert, Alexander Fronk, Frank Engelen
Experiences in 3-Dimensional Visualization of Java Class Relations
September 2001

/118/ Ernst-Erich Doberkat
The Hierarchical Refinement of Probabilistic Relations
November 2001

/119/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer, Ingo Röpling,
Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Intermediate Report
November 2001

/120/ Volker Gruhn, Ursula Wellen
Autonomies in a Software Process Landscape
Januar 2002

/121/ Ernst-Erich Doberkat, Gregor Engels (Hrsg.)
Ergebnisbericht des Jahres 2001
des Projektes “MuSofT – Multimedia in der SoftwareTechnik”
Februrar 2002

/122/ Ernst-Erich Doberkat, Gregor Engels, Jan Hendrik Hausmann, Mark Lohmann, Christof Veltmann
Anforderungen an eine eLearning-Plattform — Innovation und Integration —
April 2002

/123/ Ernst-Erich Doberkat
Pipes and Filters: Modelling a Software Architecture Through Relations
Juni 2002

/124/ Volker Gruhn, Lothar Schöpe
Integration von Legacy-Systemen mit Eletronic Commerce Anwendungen
Juni 2002

/125/ Ernst-Erich Doberkat
A Remark on A. Edalat’s Paper *Semi-Pullbacks and Bisimulations in Categories of Markov-Processes*
Juli 2002

/126/ Alexander Fronk
Towards the algebraic analysis of hyperlink structures
August 2002

/127/ Markus Alvermann, Martin Ernst, Tamara Flatt, Urs Helmig, Thorsten Langer
Ingo Röpling, Clemens Schäfer, Nikolai Schreier, Olga Shtern
Ursula Wellen, Dirk Peters, Volker Gruhn
Project Group Chairware Final Report
August 2002