CEWebS

A Distributed Approach to Share Process
Oriented eLearning Modules

eingereicht von:

Jürgen Mangler

DIPLOMARBEIT

zur Erlangung des akademischen Grades
Magister rerum socialium oeconomicarumque
Magister der Sozial- und Wirtschaftswissenschaften
(Mag.rer.soc.oec.)

Fakultät für Informatik der Universität Wien
und der
Technischen Universität Wien

Studienrichtung: Wirtschaftsinformatik

Begutachter:
Ao. Univ.-Prof. Dipl.-Ing. Dr. Renate Motschnig

Wien, im Februar 2005

Statement of Authentication:

The work presented in this thesis is, to the best of my knowledge and belief, original except as acknowledged in the text. I hereby declare that I have not submitted this material, either in whole or in part, for a degree at this or any other institution.

Vienna, February 2005 Jürgen Mangler

Acknowledgements and Preface:

The creation of my masters's thesis was the final point of an adventure that began 1995, and has lasted for 10 years. My merits as a student were quite often not as aureate as my instructors wanted them to be, and I spent more time trying to adopt the latest and greatest technology I've heard of than learning for exams. I always liked the time in lab courses more than the time alone with books and scripts. Around 2000 I attended the course *Software Engineering*, at this time held by *Renate Motschnig*, and became somehow involved with the institute, first as a tutor, later as a student assistant and technician. When thinking back I suppose that it was the fact, that I can never keep my mouth shut that lead to my now very close relations to the group around *Renate Motschnig* and the other members of the Faculty of Computer Science of the University of Vienna.

So I want to thank *Renate Motschnig*, for supporting me in creating a system although it would have been maybe easier to sit back and use existing tools, and for being always empathic, helpful, and supportive of my ideas. I also want to thank *Michael Derntl* who encouraged me to write the system that my thesis is all about. He provided we with lots of feedback during design, and testing phases and really is more helpful than it sometimes appears to me :-). The third person that influenced my work is *Christine Bauer*, who was the person that did most of quality assurance work. She provided feedback about the usability of almost each part of the system, and sometimes noticed errors even faster than I liked them to be noticed.

Following a long tradition I also want to thank my parents, my mother for providing a crisis telephone line and for sharing my enthusiasm, and my father for keeping calm most of the time and for providing council regarding profane things.

Finally I'd like the thank the students that had to live with some bugs, but hopefully also with a system that helped them to concentrate on the subject of the course not on some learning technology.

I want to write something original that is not only an instrument to get a master's grade, but is also inspiring to read and provides you with insight into personal views, my work and my vision about a better world without complexity for both, users and instructors. *Renate Motschnig* fortunately was in favour of my personal style of addressing professional issues as long as the purpose of the thesis, namely to document ”A Distributed Approach to Share Process Oriented eLearning Modules” was fulfilled. She likes to hear my ”whole message” (Honestly! She said so :-). So don't hesitate to join me in our adventure and read on.

Should you like to share the technical adventure of testing CEWebS, send an email to «juergen.mangler@univie.ac.at».

Table of Contents:

1 Introduction

1.1 Motivation

When I started my work as a tutor, I took over from a colleague a small webpage, that consisted of static HTML pages that were updated during the semester. In the beginning of the semester it showed the names of the participants, then it was updated to show the teams the participants formed during the first face to face (f2f) meeting. Later, when the students wrote their contributions, they sent their work to me (or a link to their self maintained homepage) and I put the work or the link online for the other students to see their fellows work.

After this the students read all the contributions of their fellow students and sent me an essay via email. These emails where structured to cover some topics like readability, structure, relevance of content, and so on. I then put the data from these emails together to write a survey about the work each team accomplished the term, which again was put online. As you can imagine this was an awful lot of work, in particular given the fact that:

- The collection of the information about the team structure often took weeks when handled via email
- Errors in students' names and email addresses were common, due to the fact that I simply couldn't read their handwriting, when the information about the team structure was collected via pieces of paper during a f2f meeting
- I got not only one version of a document, but between five and ten
- The team members often have not been able to coordinate their efforts, so I got different versions from different team members
- Some people ignored the proposed structure for the feedback emails, etc.

This procedure also just was not scalable for courses with more than thirty students. So I started to hack together some PHP pages that handled the above use-cases through simple HTML forms. The original motivation to build a piece of learning technology was, to just ease my work. It has to be said that this ease did not happen. Not because the system could not handle the above tasks, but because the requirements changed constantly, and the habit that the students had their own webpage and sent just links, vanished. At some point in the past all documents were uploaded, and then there were not only single documents but milestones with multiple documents and more complex reviewing processes following each milestone.

Now responding to the email for courses with 200 students and looking at the special cases where the students complained that they couldn't do this, and the system did not do that, is in fact twice the work than the standard cases were in the past. But ...

- The numbers of students in the courses has increased by a factor of ten

- The possibilities provided by CEWebS (for both, the students and the facilitator) are infinite compared to the approach five years ago
- Looking at special cases is not as exhausting as assembling large standard datasets into something useful by hand

So in fact the workload increased, but I like the tasks better and can carry out them more efficiently.

The motivation to develop CEWebS was to replace the hackish solution with something more durable and manageable, that is not only a loose set of PHP pages which has to be copied from one directory to another at the beginning of a term. It is an attempt to build something that is easy to use, usable in different environments and easy to manage.

1.2 The Task

So the task that I want to solve is given by the work I'm doing. I want to document my work and the system I created, regarding the following characteristics:

- Web-services are a useful technology, when used appropriately
- To design a modular distributed system pays off not only regarding scalability but also maintainability
- Learning is a process for each student or team, not a series of duties and exams for a group

The first two characteristics focus on the software engineering part of the problem. I've a strong feeling that web-services are often misused these days. They are part of a distributed hype and are used to connect parts of software that could be otherwise connected much more efficiently. What is the benefit of connection between two java systems through web-services, when it is foreseeable that they will never be connected with anything else? RMI would have been more appropriate in this case. The same goes for .NET solutions. I think I have found an application where web-services are very appropriate and have a benefit of their own: bringing together different technologies to form an optimal solution.

On the other side, the web-services allow us to embed the result of our work into existing platforms and thus allow us to concentrate on supporting students, not on building surrounding infrastructure by implementing just another platform. The system also helps to build a network and a community, because we can offer services to others and equally allow them to consume our services in their platform, as well as consume the work of others.

The last point regards the way how platforms often work. The BLESS [5] model of *Michael Derntl* and *Renate Motschnig* describes the coherences between didactical theories and an actual implementation in a learning environment through a layered model. It has five layers, with to the topmost layer describing the learning theory behind a course, in our case Blended Learning [6]. In the second layer course scenarios are identified and modelled through activity diagrams. In the third layer course scenarios are decomposed into learning patterns. The last two layers are all about realisation. Web templates are identified that help developers to implement the functionality. I think that todays learning platforms provide a loose collection of material and tools without trying to establish an overall concept for a course. The task of explaining the purpose of elements of the platform is left to a tutor or the course designer. The platforms work like a digital cupboard where all things fit into well defined compartments. But if one wants to find something in the cupboard he / she has to know the right compartment (given that everyone has put everything into the right place) or to search through all compartments. This comes from the fact that these systems are often derived from Content Management Systems or were designed just with single tasks in mind and not the whole picture of a course scenario.

This does not seem to be the right way. I tried to make the system as a whole as flexible as possible. Every part can be interlinked with every other part, to support the flow of a course and to support the findings resulting from an application of the BLESS model. In the second part of this thesis I will show actual examples of services. I try to detach the user from the task of finding the right compartment. The system also has additional interfaces that allow an aggregated view on the course. The course should be no longer a cupboard but a multidimensional data cube, a data warehouse that lets us extract and interconnect information about the students.

1.3 The Structure

This thesis has three main parts (see Fig. 1). The structure is derived from how the system was designed and used during the last two years (its roots are more than three years old).

The first part concentrates on the software engineering aspects. In fact the system is not only usable for Blended Learning purposes but could be applied to every area where flexibility and scalability are necessary. This part focuses on the technical aspects of my work. The technical terms that are used are explained the first time we encounter them, to not bore the reader with a chapter that is solely about technical perquisites.

The second part tries to show how the system is used and performs in the Blended Learning context. It contains use-cases for most of our services, shows how they are used in real life, and highlights also fields where they could be improved.

The third part will be not as voluminous as the first two and focuses on the consequences the Action Research approach had on the development.

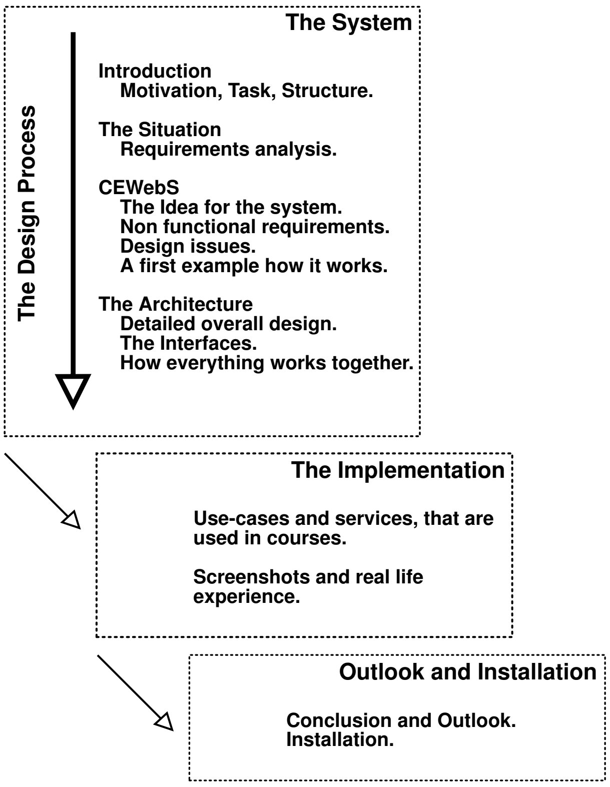


Figure 1: The Structure of This Thesis

2 The Situation

To illustrate the situation I try to give some information about recent courses and how we tried to support them by technology.

2.1 TeleWIFI Communities

In the summer term 2002 we tried to support our advanced course on *Planung und Realisierung von Informatikprojekten* (PRI II). The course setting was as follows:

- The students formed teams of three to four persons
- The goal was to create a portal for Business Informatics Students called WinLearn
- Most work-packages where done by the teams individually (e.g. Requirements Engineering) and then discussed in the group to form a final proposal for the portal
- For each work-package one team was assigned to the task of merging the results
- There were also several work-packages that were elaborated by the individual teams (in the form of prototypes)

- the *TeleWIFI Communities* (TWC) platform was available for the second half of the course (from mid April to the end of the course)

The TWC had two categories available, one of which was called *Media*, the other was called *Forum*. I use the past tense here, to signal that this was a prototype that definitely has evolved and is maybe no longer existing in its previous form. Within *Forum* one could write messages (and attach any file to it). One problem was, that if the person who wanted to write a message forgot the subject, that message silently just not appeared in the forum. This was a bug though, not a missing feature. *Media* was a filesystem abstraction, where one could deposit arbitrary files.

We were very thankful to Mühllehner and Tavolato GmbH who made it possible for us to gain hands on experience with a simple community platform. In exchange we provided feedback to allow for improvement. Indeed TWC caused some fundamental inconveniences:

- There was no access restriction at all: every user could delete / modify / add files everywhere inside the media category. We had to establish *social rules*, a guideline that users should not modify or delete anything that other users created. This worked, but some *mistakes* were made, that lead to missing files.
- The forum functionality was not very intuitive.
- The system required ActiveX (see [19], [33]) (a technology tightly integrated into Microsoft browsers) to be installed and working and therefore required a recent Internet Explorer.
- The included functionality was just too simple to realise e.g. Peer Reviews in an intuitive way (solved through uploading files)

All these points relate to the fact that the TWC platform was designed with a different setting in mind, focusing on courses held in an adult education environment with a small number of participants, where its is presumably working very well. For a university lab course with high numbers of students a more guarded approach is needed i think.

As a side note it has to be said that the most notable event of the course was a discussion if an optimal solution has to be data-centred (like TWC) or function-centred (strong focus on tasks and workflow). A function-oriented approach arose as clear winner.

2.2 Dayta

After using TWC for two semesters we were curious to use Dayta and we also got a dedicated server from our departments resources for it to help us intensify usage and research of learning technology. Dayta is based on the Zero Object Programming Environment (ZOPE) which is written in Python (see [28]), an object oriented interpreted language. ZOPE is described as follows (see [41]):

> ”Zope is an open source application server for building content management systems, intranets, portals, and custom applications.”

Later Plone (see [27]) with some additional components was reused to create the eLearning environment. Again we sincerely thank the providers (TomCom GmbH) to have given us the opportunity to use their object-oriented platform and gain hands-on experience with a very elaborated tool.

We used it for various courses, including the above mentioned PRI II. For the following example I will concentrate on a course called *Web Engineering*, held during the summer term 2003:

- The students formed teams of three to four people.
- The lecture notes were available on the platform from the beginning.
- Every team selected and elaborated its own project, including planning and realisation.
- Every team had its own working directory where they had full access-rights (add / change / delete everything as well as giving access permission for certain content to other people)
- Every team had to design its home directory so that other students could see the achieved progress during the advance of the course
- Every team had to contribute documents defined by milestones
- There was also a forum where discussion took place concerning issues that were raised during the lecture or lab hours
- Peer reviews took place after each milestone

Plone is a ”Content Management System” (CMS) that provides a traditional filesystem hierarchy with the ability to add content to directories. Special objects like chats exist and appear also as files in the directory. The forum functionality can be attached to every file that is created inside the platform. Content can be created by utilising an integrated HTML editor. Images can be uploaded to directories and used inside the HTML editor.

For being a CMS at heart the system is naturally data-oriented. For a course the course designer had to take the following steps:

- Create a directory to hold the course.
- Create a directory for each group (the course consisted of multiple groups of about 30 people each).
- Import the users (from a handcrafted text-file).
- Create group and team objects (collections of users) and add the users to these objects.
- Create the team folders inside each group folder.

- Assign the appropriate team/group objects to each team/group-folder.
- Change the permissions of each folder so that the owning team only could change the content.
- Create the content for the course, and each group.
- Create an administrative forum for the course and for each group (a document with a forum attached).

We decided that the teams should keep all data in their directory. As said before each team had to design an index (page) for their own directory (with the integrated HTML editor). In their team-folders they collaborated to create the documents for each milestone. After the document was ready they added a link to to the index (page) of their folder.

The following problems appeared:

- The creation of the directory structures and the assigning of the appropriate permissions for 300 students was exhausting and took about three days. Many errors slipped in because my concentration faded away.
- The permission system proofed to have many flaws and was quite inflexible. Although there were some predefined roles for users, these roles had to be tied to operations for every single object. The inheritance of permissions often did not work as expected for sub-directories and the objects inside them.
- Also the students had to learn how to use the permission system to maintain their own folders.
- The students did not obey the rules of publishing milestones. The design of every page was different, determined by their own interpretation of the milestone structure. They had to put quite some energy into maintaining their work-directories.
- The facilitator had to browse through hundreds of directories only to check if the milestones documents were available and was confronted with ever changing structures inside the teams directories.

We concluded that the approach that dayta took is highly useful for small groups of people that work in a creative way, but not for mass lectures or lab courses. The static nature of the system was not appreciated by both, the students and the facilitators.

Although we had a dedicated server the system performed poorly in 2003. For more than ten concurrent users, the access time was often more then twenty seconds. Additionally the operations involved in the process of creating directories and assigning users often took several seconds. In general we found the system to be quite unstable, with regular crashes. When we asked the company that provided us with the product, they told us that the speed problems are created due to a misconfiguration (debug style logging turned on), but were not able to fix it. We admit though that this could be individual problems, as Zope in general is known to be stable and scalable.

Despite this difficulties the adoption of Dayta for free allowed us to gain insight that we could not have acquired without. We also hope that our feedback proved helpful for improving Dayta for future use and are open towards potential cooperation with the EduPlone initiative.

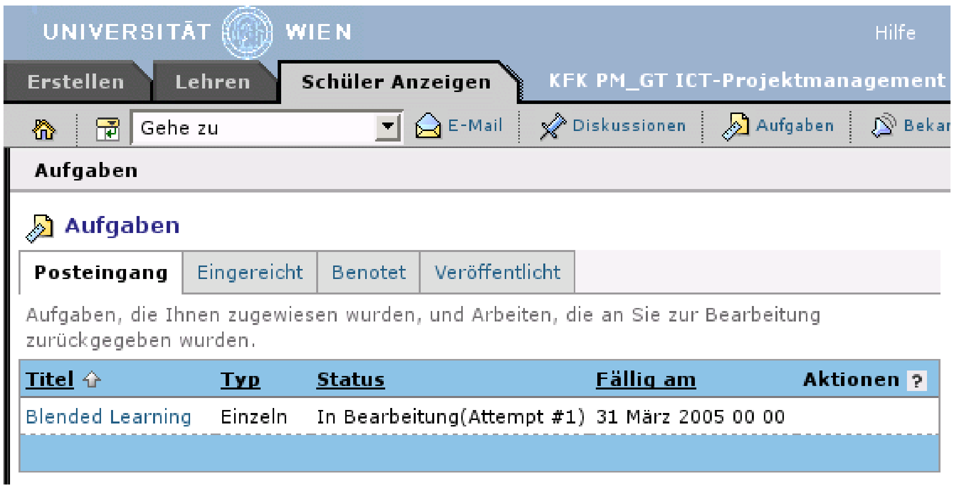
2.3 WebCT

In the winter term 2004 we had the chance to work with WebCT Vista, a service which is provided by the Vienna University Computer Center. WebCT Vista is a professional platform by WebCT Incorporated (see [32]) and is used by over 1000 institutions in over 80 countries. It is from the bottom up designed to be a learning platform and therefore promised to have many advantages. We tried it out for a course called *Projektmanagement: Grundlagen und Techniken" with about 25 students. There was also the advantage of getting help from professional eLearning tutors (thanks* Petra Dryml"). We had about the same course setting as before:

- The students formed teams of three to four people, who were enlisted in the platform
- We collected the students expectations about the course
- The students had to plan a project and submit multiple documents grouped into milestones
- The students did peer reviews of a partner team
- There was no need for a working prototype
- The students did several multiple choice tests, that were not graded though
- The users did self evaluations

WebCT proved to be very capable and everything we wanted from it could be solved. Although there were some bugs, which can be attributed to the fact that the platform was installed for the first term, everything went nice. However we noticed some (I think) severe design issues in the system:

- WebCT has big usability problems. Multiple ways of navigating inside a course were confusing for the students, the facilitator and the course designer.
- WebCT tries to resemble workflows from real life. When looking at the procedure for submission from a student we discover the following:
  - The student can enter text and can append multiple files.
  - He / she then has to post off the submission.
  - When he / she wants to make some changes after this, he has to move it back from *Eingereicht* to *Posteingang* (see Fig. 2).
- The page the students see when they enter the course can't be modified to show arbitrary objects. It is only possible to put HTML header and footer text on this page, accompanied by symbols that lead to the functionality. The usage of symbols that are not always easily recognisable, which adds to the usability problems.



Aufgaben, die Ihnen zugewiesen wurden, und Arbeiten, die an Sie zur Bearbeitung zurückgegeben wurden.

Titel	Typ	Status	Fällig am	Aktionen ?
Blended Learning	Einzeln	In Bearbeitung(Attempt #1)	31 März 2005 00 00	

Figure 2: Accomplishing a Task From a Students' Point of View

- Although it is possible to link to external pages it is not possible to take advantage of WebCT's authentication mechanisms in external applications.
- The WebCT whiteboard has a focus on painting lines and primitives (modifying pixel graphics), in computers science there is a need for more elaborated drawing and modelling facilities (we will concentrate on this issue later).
- Peer Evaluation can be realised as submission but is not obviously related to the peer-reviewed object.
- Although there is the ability to create reports that show for all students status of their submissions, it is not possible to notify them according to certain criteria (e.g. write a message to all that have only provided one file, ...)
- When working in teams, the team has to coordinate internally who submits. Only the person who first changes and submits a task can alter it.
- For a given task it is not possible to per default make all submissions visible to all students. For this behaviour one has to reuse the discussion forum which results in a not so clearly arranged structure and is not optimal regarding reports and grading.

We think that WebCT for trying to be everything to everyone has a tendency of being over complicated. Many simple workflows are quite obfuscated because the platform is so powerful, and the power is not hidden from the user. Modelling things after sequences from real life is a nice idea, but (I think) not necessary. The platform should hide things from the user he / she does not need. Another issue for us is that, for WebCT being a proprietary system, we are not able to modify it. E.g. the system has only the ability for the facilitator to grade students or write comments. Providing an add-on module that can handle the use-case of students grading other students is not possible (at least not the easy way). There are at least three reasons for the fact that we possibly can't modify WebCT:

- We have no access to the system because the University of Vienna can not afford to set up multiple test systems

- The changes should also go back into the main version after extensive testing of the whole system, which is not possible during operative usage (presumed we had the resources for doing the testing). So our changes maybe can only be merged from release to release (every 2 years?).
- The internal API's supposedly are not designed for 3rd party additions, and suspected to be quite complex (and undocumented?).

Another problem we see is the focus on technology. WebCT heavily relies on Java applets which has the effect that the back button of the browser is unusable. *"Breaking or Slowing Down the Back Button"* is the number one mistake of web design according to Jakob Nielsen (see [25]). I broke WebCT more often than I can count because out of a reflex I clicked on the back button. The focus on Java inside the WebCT server also imposes the problem that one possibly can't reuse already existing code. Everything has to be rewritten in Java.

2.4 Summary

After using different systems we can conclude several things. **None** of the used systems supported our course settings in an optimal way, we always had to make some sacrifices due to limitations of the used platform. We had to adopt our setting to the platform. I want to recapitulate (what I think are) the most serious shortcomings:

- To *keep it simple* (KIS), in other words provide the students with only minimalistic and self explaining tools, under the precondition of optimal support of a learning scenario wasn't possible.
- There were missing parts of functionality that would have eased the handling of a course (support for creating teams only instead of collecting data through emails or paper)
- The systems focused on tasks, not on the workflow inside a course (peer-review is possible but not integral part of the submission).
- The systems were either proprietary (TWC, WebCT) or derived from highly complex general systems (Dayta) and impossible or very difficult to extend.
- The systems were not flexible and locked to a certain technology like Java or Python. There is no easy way to extend one of the existing platforms.
- There was a lack of / unsatisfactory support of authentication and / or user models. All platforms were weak when dealing with changing teams/multiple teams.
- There was a specialisation on content management/presentation, rather than user interaction.
- There was no easy way to keep data about instructor-student interaction (e.g. searching through archives) and generate reports that e.g. show trends over multiple terms.
- There was no easy or standardised way to separate the data that specifies the structure of the course from the data that is generated and accumulated throughout the course.

- There was no easy or standardised way to support/generate courses with several concurrent groups, but identical structure.
- There was a general lack of usability.

I found the platforms to be static, inflexible and monolithic. I think this is a design issue with all of the platforms. They deal with data structures in traditional databases, and these data structures are also manifested in the user-visible structure.

3 CEWebS? What is it?

3.1 The Idea

In order to overcome the difficulties we encountered with other platforms we began to create a framework that would allow us to embed components into existing platforms at the University of Vienna (e.g. Plone [27] , ILIAS open source [14] , BSCW [1], ...) to bring in the flexibility we needed. The seven corners of our framework are:

- Keep it simple - Implement a minimal interface to get the desired functionality.
- Freedom of choice - Given the correct implementation of the interface, one should be able to use whatever programming language / framework one prefers in order to add functionality.
- Scalability, flexibility - Easy distribution of functionality across servers on the Internet and various server platforms.
- Open source - Everyone should be free to use our repository of existing components, everyone should be free to implement the interfaces. That allows for integrating new services into the product.
- Reusable patterns - In our concrete case, this means support for blended learning scenarios modelled conceptually by using UML activity diagrams [5] that can be saved and reused. Structural data and data that accumulates during the course are separated.
- Embedable - The components can be embedded in and interact with existing systems
- Interaction - The components can share data and interact with each other transparently without knowing of each other.

Building on research on Person-Centered e-learning [22] and blended learning patterns [6] we decided to create a framework using recent technologies like Web Services [4], SOAP [38], XML [37], and XSLT [39]. The latter should be able to provide us with the flexibility we need to combine the components we need, and react to the needs of blended learning in very short time periods.

The name for our system is CEWebS - Cooperative Environment Web-Services. This indicates that the system is not eLearning specific, although this document deals mostly with a set of services that belongs to this field. The system is more of an enabler for Enterprise Application Integration (EAI) in the sense that it can package all sorts of applications and tools and allow them to interact with each other, and behave for the user like a highly personalised environment.

3.1.1 The Art of Choosing a Solution

The first decision one has to make is whether to **use an existing solution or** to **create a new one**.

Existing solutions have a given architecture and feature set, and it is sometimes difficult for them to live up to the expectations (one has to sacrifice usability, cannot achieve the exact target or has to cut back one's expectations). The following real-life examples illustrate this quite well:

> In summer term 2001, we used a platform called TeleWIFI communities supplied by Mühllehner and Tavolato GmbH, used for courses in the WIFI environment. It provided us with the functionality of forums for discussion and workspaces to share documents and other files. Forums and workspaces were strictly separated. There was no permission system whatsoever, so a logged in user could change everything and delete every message in the course. This proved not to be a problem in the given course because the 15 participants were a manageable group. We adapted to the system and established social guidelines for the usage, where a fine grained access control would have been a solution.

> For the next one and a half years, we used a Dayta / Zope / Plone / Eduplone derivative supplied by TomCom GmbH. It was a real leap forward in functionality and overall productivity. It is based on Zope, which is in fact a content management system, with the Dayta and later Plone and Eduplone trying to add easy to use components. For being a content management system at heart, it focuses on files and folders. Every file a user uploads needs its place in the filesystem hierarchy, having its own permissions and visibility. Custom functionality like forums can be added to each object, chats can be added to folders, and so on. The main course were we used it, had then about 300 participants, with team based projects but also individual contribution, it was necessary to create a place where teams could upload their files, and to ensure that everyone had only permission to access only appropriate folders. Creating the whole infrastructure was quite exhausting (about 400 directories, each individually supplied with a plethora of access restrictions and user information). We accustomed to the setting, but the recreation of the structures for new students each term, and the constant browsing through countless folders to just check if a contribution has been made created some resentments for both, students and instructors.

> In the summer term 2004 the University of Vienna switched over to WebCT, which is in fact a whole new experience as being from bottom up designed to be a Learning Environment (LE). It has features like ”Multiple Choice Tests”, ”Chat + Whiteboards” and the ability for generic duties, be they team based or individual. The contributions are listed in concise manner, one can check easily which student has made a specific contribution. What remains, is the lack of aggregation: one cannot print out a list of all contributions made for a single duty, or the results of all ”Multiple Choice Tests” for all users (for ”Multiple Choice Tests” there is also the problem that the instructor cannot check easily which students did which test). Regarding content the platform is inferior to the solution we used before. Managing course structures is again not possible for the instructor (the technical staff offered to copy the structure of whole courses). Again we accustomed ourselves to the way courses can be structured in a WebCT environment.

The above examples illustrate what I consider a serious problem: We adapt to a way of using technology and get accustomed to it. Then our material and courses are kind of locked to a single piece of technology, that holds our minds captive. Behind each approach stands a dogma, that can not be changed and affects the real life, i.e. the courses' structure. In my mind, technology should support real life, not shaping it.

Homegrown solutions on the other hand can be designed to do exactly what you want. They support exactly your style of teaching, but have a whole world of problems of their own.

- Not everyone can write one's own solution. The resources and the knowledge involved in writing something unique and useful are quite expensive.
- Homegrown solutions tend to be specific and inflexible, because they are very tightly designed systems, architectural changes aren't easy to accomplish.
- They may die when persons leave or loose interest
- It may occur that they are not useful for anyone else

For large organisations like the University of Vienna it is (oder zumindest it's) clearly easier to buy a commercial product with support and a planned upgrade path, for being much more transparent cost-wise, and also for providing the basic support everyone needs. The specialised needs that arise from the different focuses of the faculties cannot be taken care of anyway (in my opinion).

The second question that arises nowadays is if to use **Closed-Source** (commercial) **or** an **Open-Source** product. Assuming that there exist two equally suitable products there are as well reasons pro and contra.

Regarding traditional Closed-Source productions, there is the advantage of support and persistence. If one chooses a product from a well known company one wants also to rely on the support the company provides for the product. There is also the advantage of planability.

Furthermore there often exists some uncertainty regarding Open-Source projects:

- Do they take the right direction?
- Do they continue to exist?
- Can I build up enough known-how for an own support team?

I think these fears are weak. Open-Source projects are surrounded by communities, and like commercial projects, one can take part by stuffing money in it. In Open-Source projects the money is used to pay human-resources for actively participating in the community. This also solves the other two problems, as long as the project has human-resources it will at least continue to exist, and the payed human-resources can also double as a very efficient support team. The difficult part is, to not too much affect the project, so that other parties will not start their own effort, because the project gets too specific. But i think there is a plethora of projects that show that they can handle this problem and be even more successful than every existing Closed-Source application (e.g. the Apache web-server).

3.1.2 Homegrown and Open-Source

When writing an own system that is not too specific (or does not include information that is vital for an organisation) it is definitely wise to start an Open-Source project. The possibility to create a community that adds additional human-resources, experience, expertise and ideas should be priceless.

I think the creation of a loose framework rather than another complete platform combines the benefits of all four models and can be a valuable addition to todays existing infrastructure. CEWebS was created because of the following reasons:

- Individual combinable services can help to keep the environment simple, for both the user and the facilitator.
- In the field of computer science (and in fact for all natural sciences) there is a need for special tools; CEWebS allows for the encapsulation of existing tools to provide a web-interface for students.
- There is a strong tendency in the European Union (see [8]) to support Open Source. Developing an Open Source framework for the exchange of standardised Blended Learning components can help to establish new cooperations with other universities.
- We want others to join our efforts. Open Source is a way to transparently share work with others.
- The learning platform provided by the University of Vienna offers tools that are also valuable for cooperations and projects with other universities or companies. We are not allowed to use it for other purposes. With embedable components one can bring students in touch with persons from outside the university without breaking the rules. Furthermore one can use synergy effects and reuse tools for projects.
- To explore the effects of the Person Centered Approach we need a flexible framework to experiment with. For our research to be effective we need a framework that can be changed and customised in the short-term.
- Investigation into new technologies and applications that also provide easements for our daily work seemed a good idea.

When creating an Open-Source project, one has to choose a license under which it is published. In our case I decided for a mixture between LGPL [13] and GPL [12], to allow also for embedding into commercial Closed-Source projects.

The GNU General Public License (GPL) basically allows everyone to freely use, change and distribute as long as the copyright statement and the license text is included. Changes to the source are permitted, as long as they are also made available via GPL. In other words: any derivative work has to be also Open Source and has also to be released under the GPL. Quote from the preamble of the GPL:

> *To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.*
>
> *For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.*
>
> *We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.*

The GNU Lesser General Public License (LGPL) on the other hand is less restrictive. It allows for the inclusion in a commercial product without the restriction of releasing this product as open source. But there are no changes to code allowed, that is released under LGPL. All these changes again have to be released under the LGPL. In other words: it allows for the linking to a library or in our case the implementation of an interface, without the necessity of releasing the final product as Open-Source. When an interface is implemented, it is part of the final product which (when released under GPL) would require to that software to also be released. This may not be possible because the product includes code from other copyright holders. So we chose the LGPL.

Our test-implementation and the services are released under the GPL, all interfaces are released under the LGPL. This has also the benefit that in the case of violations of the (L)GPL, one can report [11] them to the Free Software Foundation which eventually takes legal action against the violator. These actions have so far always had the consequences of convincing the violator to release the code.

3.2 Usage of The Term CEWebS

The term CEWebS should be used in its pure form after a short explanation. Under no circumstance one should use CEWebS Services oder CEWebS Web Services. Examples for correct usage of the term:

- CEWebS (plural) are made to be integrated into existing solutions. They can help to abstract problems that exist in heterogeneous environments, and create an integrated solution.

- Different departments in a company have their own specialised software, tools, and servers, but can provide a CEWebS (singular) to be integrated in a company wide intranet solution.
- All CEWebS (plural) share a common interface, data exchange, and message passing format.
- A CEWebS (singular) can share its configuration data with other CEWebS (plural) over the Internet. When its configuration changes, it can send a notification to a central broker that distributes the changed configuration to all affected CEWebS (plural)

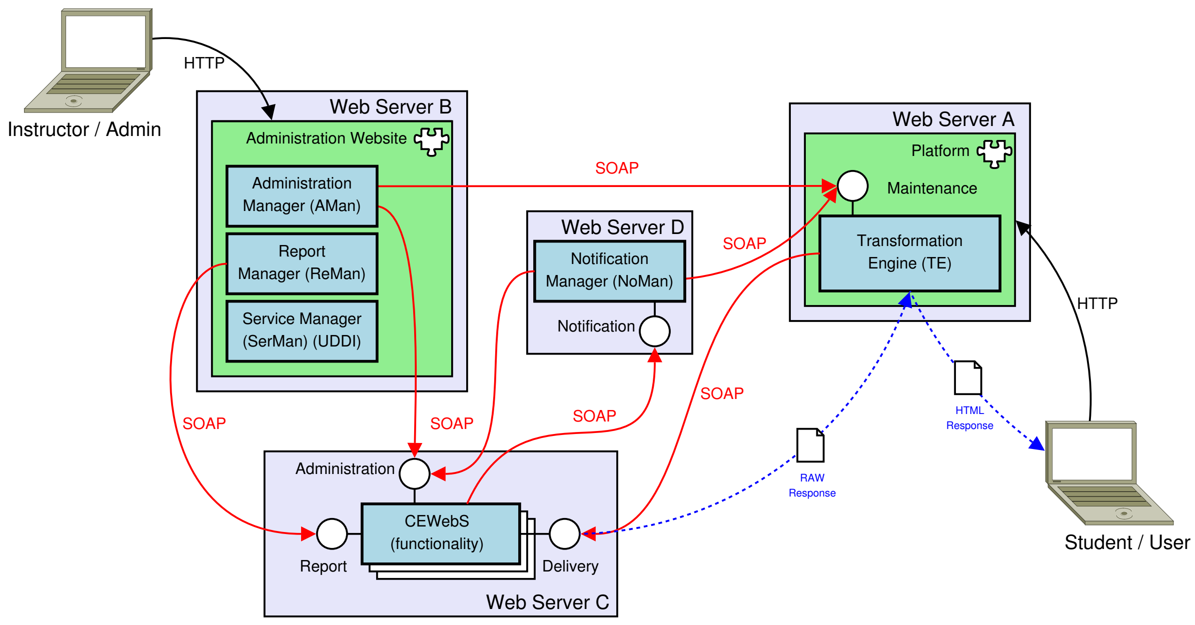


Figure 3: The CEWebS Architecture

4 The Architecture

The framework (see Fig. 3) consists of four main parts:

- The Administration Website
- The Transformation Engine (TE) (implementation of a simple standardised SOAP interface)
- CEWebS - the web-services that implement the functionality
- The Notification Manager - allows the services to exchange data

Initially there is an empty container (i.e., the course platform), which is dedicated to hold various components according to an initial course design. The administrator initialises the TE (container) with the CEWebS it should show and the users that should be allowed to use them. Then he or she initialises (configures) every service with the user information and additional data that is required by the service.

The Administration Website itself is a centralised service and user repository that additionally holds information about all available TEs. Administrators are assigned to these engines and can distribute users and services to them. The pieces inside the Administration Website are itself CEWebS and have the same interfaces. They also exchange data about available services, users, and TEs by using the Notification Manager (although the arrows describing this functionality are left out to not complicate Fig. 3.

The Transformation Engine (TE) is the link between all users and the framework. It has two main functions:

- translating user requests to CEWebS invocations
- transforming the response via XSL or CSS to meet the user-interface guidelines given by the surrounding environment (e.g. a commercial learning platform)

The second function implies also that for being easily embedable into third party products the TE can be realised using different technologies. By embedding it can enable extensible single log-on for intranets or learning platforms.

The response from a CEWebS is a raw XML document, which is defined as a subset of XHTML 1.0. It consists roughly of **<h?>**'s, **<div>**'s, **<a>**'s, **<b>**'s, **<form>**'s and **<input>**'s that can be styled via special attributes. The goal was, again, to keep it simple as well as to allow developers an easy entry to CEWebS based development.

The CEWebS are the central parts of the architecture. Every TE can communicate with an arbitrary number of CEWebS through a well-defined interface. The interface gives the TE the ability to:

- Get XML data that can be transformed into a specific user interface
- Get data that is embedded in a specific user interface (applets, images, data / documents for plugins, ...)
- Get binary data (clickable downloads)
- Decide whether a request requires authentication
- Cache data to keep the data flow between a CEWebS and TE low

Every CEWebS can hold multiple instances of user / configuration data, corresponding to each TE through which it is used.

By using the Service Manger (SerMan) the Administrator can register new services (by providing a description and the URLs of the interfaces of the new service). He / she provides this information to the Administration Manager and which acts like a UDDI for the Administration Website.

The Administration Manger (AMan) is the central configuration tool. With the AMan an administrator can add new TEs (by providing a link to the maintenance interface). These TEs can then be initialised with a set of users. Adding new (or existing) CEWebS instances is also possible. All parts (CEWebS and the TE) are provided with the user information and also access keys (only a TE that possess the correct keys can access a particular instance of a CEWebS). The concrete form of a TE holding several CEWebS in our case is a course. The structure of all courses (participated CEWebS) is provided to the Report Manger.

The Report Manager (ReMan) connects to every CEWebS Report interface. This interface simplifies the querying of reports for a CEWebS instance. It allows the administrator to collect analyses of certain incidents, e.g. users that missed the submission deadline for their contributions. Through the report interface the administrator can always receive binary data, e.g. standalone HTML pages, EXCEL Sheets, PDF's, or JPEG's containing the solicited information.

The components of the Administration Website are also CEWebS and share the information about available services and the course structure by using the Notification Manager.

The Notification Manager (NoMan) provides means to transparently exchange information for the CEWebS similar to the Blackboard approach (see [3]) that is also used in todays mobile agent systems. A CEWebS can send information that could be important for other CEWebS to the NoMan. The NoMan then selects services inside the same TE that could be interested, and forwards them the information. The same principle is also applied to user information. For the CEWebS to share information they need a common understanding about the semantics of the exchange data. For this to be transparent all services that share a particular understanding of semantics belong to a group, in our case **PCeL** which stands for **Person-Centered e-Learning** [23].

The main concept of this architecture is, that the CEWebS in a TE know nothing about the other services. Only the TE, the NoMan, and the AMan (and ReMan) know about the coherences. For the services themselves everything is transparent, which makes them easy to implement. Interlinking between the CEWebS inside a particular TE is possible though, but has to be explicitly configured by the maintainer of the course through special links (CEWebS://, which is elaborated in more detail in the next chapter)

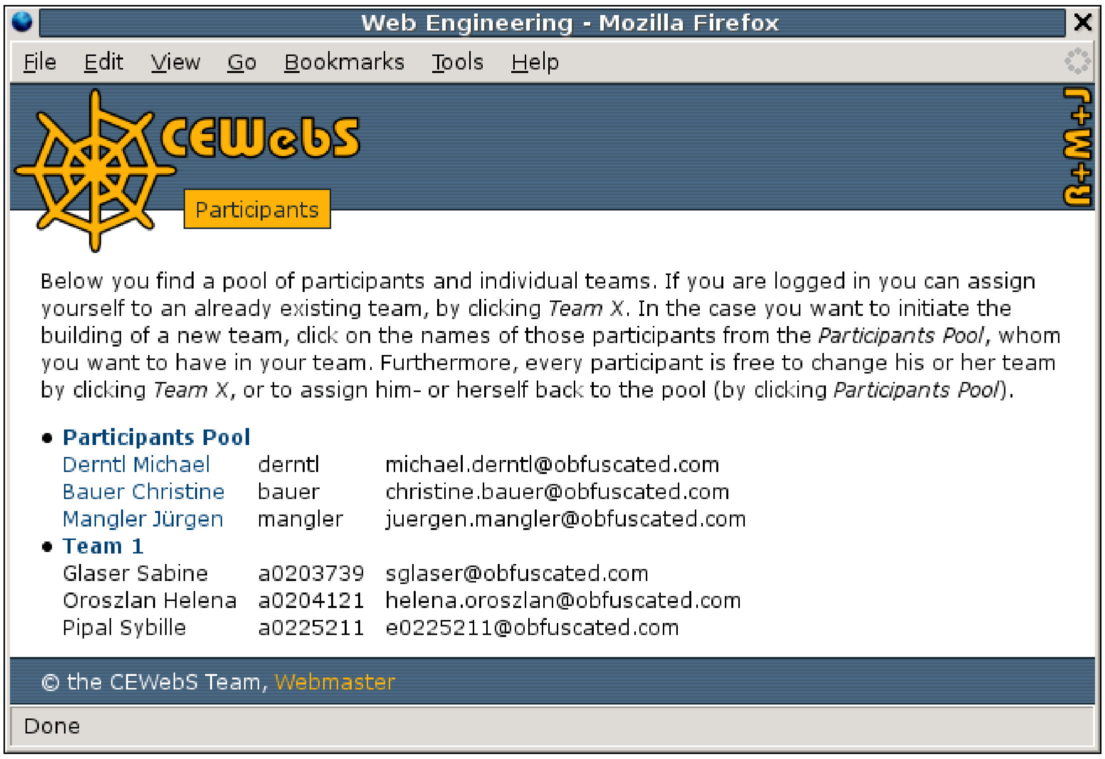
4.1 A Concrete Scenario: The Process of Building Teams

By using the concrete example of a CEWebS that enables students to build teams online, we want to show how the interaction between a User and a CEWebS works.

The basic data flow is defined as follows (see Fig. 4):

The user selects "Participants" from the menu: The container (Transformation Engine (TE)) holds a list of all CEWebS (location, name, visibility, description) locally (note that in Fig. 4 there is only one CEWebS in the container). The name of a CEWebS is a container-unique identifier that can be used to implement container-internal cross-CEWebS links. Each CEWebS also carries a visibility status, that tells whether its link should appear (in our case as a menuitem in the navigation bar). It is up to the TE whether it shows the visible menu items or whether it passes them to the surrounding environment.

The TE selects the responsible CEWebS: The TE selects the location and the initial command for the CEWebS in question. CEWebS use identifiers called *commands* to refer to certain parts of their functionality. The container also holds information about the default command that is used when a CEWebS is initially called. Together with CEWebS location and information about the CEWebS internal data instance and optional default parameters, the connection can be initialised and the data can be addressed.


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Participants
R+M+J

Below you find a pool of participants and individual teams. If you are logged in you can assign yourself to an already existing team, by clicking *Team X*. In the case you want to initiate the building of a new team, click on the names of those participants from the *Participants Pool*, whom you want to have in your team. Furthermore, every participant is free to change his or her team by clicking *Team X*, or to assign him- or herself back to the pool (by clicking *Participants Pool*).

- **Participants Pool**

Derntl Michael	derntl	michael.derntl@obfuscated.com
Bauer Christine	bauer	christine.bauer@obfuscated.com
Mangler Jürgen	mangler	juergen.mangler@obfuscated.com

- **Team 1**

Glaser Sabine	a0203739	sglaser@obfuscated.com
Oroszlan Helena	a0204121	helena.oroszlan@obfuscated.com
Pipal Sybille	a0225211	e0225211@obfuscated.com

© the CEWebS Team, Webmaster

Done

Figure 4: Building Teams

The TE collects initial information about the document: Each TE is intended to do local caching to keep the data load at a minimum. So first, the method *prepare* is invoked, which returns a cache key by which cached documents can be identified, and a boolean value that indicates whether authentication for this command is required.

The TE requests the document: The Information about the last modification of the cached document and additional parameters are sent to the CEWebS via the *doCommand* method, which returns a new document plus a new modification date, or nothing if the cache holds the current version. If a document is returned, it is saved in the cache and the modification date is set according to the provided date.

The TE delivers the document to the user: The document from the cache is then optionally XSL-transformed and sent back to the user. Due to the structural similarities between the document and ordinary XHTML, the styling job can also be done by CSS, which is less resource-intensive.

The user selects a link that leads to a group view: Basically the same process is repeated with additional parameters. Although links in the platform appear like normal URLs to the user, in fact the document holds the links in a special format, which is transformed to a browser-readable URL.
In the given case, a document showing participants of group 1 is returned (or served from cache).

The user selects a link, to start the team-building process: For the first time *prepare* requires authentication. The TE container holds information about all users that are allowed to access the CEWebS and shares it with the surrounding environment. The platform is now forced to authenticate the user, or to return the current user's identity if it is already authenticated. From now on the user information is provided with every invocation of the *prepare* and *doCommand* methods.

The user selects a person in the pool: Inside the CEWebS the two persons (the current user and the clicked-on participant) are assigned to a new team. The information about the team change is propagated to the AC part of the Administration Website, which in turn distributes the information to all affected CEWebS (the ones in the same container).

4.2 Action Research

Our blended-learning strategy is guided by *Action Research* [7]. Action research tries to compare different conditions and practices in social environments and how they change over time. In our case these social environments are supported by a technical solution, in other words blended learning courses are supported by a learning platform. One of the characteristics of social environments is, that there are constant changes, and that it acts very dynamic and flexible. So there is a pressure for the supporting technical solution to also be very flexible and to some extent it also is constantly changing.

Constant change is considered poison in traditional software development for the following reasons:

- Constant changes introduce new errors that affect the overall stability and performance of the system as well as the user satisfaction.
- Different groups of users (in our case different courses) that use the same system are also affected by the changes, although they maybe do not want them.
- Data structures (from previous terms) can no longer be read by the system.
- Old data (from previous terms) can no longer be compared to current data.

The proceeding that is forced on us to keep the system up-to-date and incorporate the constant user input, is similar to the iterative waterfall model and includes elements of exploratory programming [31, p. 8-12]. The introduced architecture is well suited to take care of constant changes. Errors in components can not affect the overall performance of the system. It will now be explained in more detail, how we are dealing with the demands:

- Every CEWebS is a truly independent component, it takes care of its own data and resources, and only is accessible to the outside world through a well-defined interface. The CEWebS can even reside on its own machine. It includes its own web-server.

- The TE aggregates all services, if one service fails an error is shown for only this service, all other services continue to work as expected. The user can at least use the rest of the system, not everything is down.
- When a improved version of a CEWebS is needed for a course, the service is forked. The new service runs on a different port, is accessible through a different path, or runs on a completely different machine. The process of changing to a different CEWebS is as easy as changing the URL for the WSDL / endpoint. The original service can happily continue to run on its own location. Archived data from previous terms can have its own CEWebS running, that allows everything to be conserved and kept in exactly the state it was in the term it was used.
- Through the NoMan parts of data can be migrated at runtime to different CEWebS, the new service can add missing pieces by using default values. Implementing a mechanism in the AMan that tells a service to migrate and then deactivates it is easy.
- The system is very scalable, the CEWebS are true objects that can be moved out of the way to a new machine if performance bottlenecks occur.

The ability to develop the CEWebS independently accounts for the relative stability and reliability from the user's point of view. Huge changes in data-structures can be intercepted by forking the CEWebS and conserving the data that was accumulated up to the forking-point. So the migration of old data can be delayed and planned, because everything is still accessible.

4.3 The Transformation Engine

The Transformation Engine (TE) is the central piece that glues the web-services together and gives them their appearance. Each TE has the following attributes:

- It holds *0..n* CEWebS
- Each CEWebS is identified by a unique name
- Each CEWebS is described by a location (WSDL) (see [36])
- The location is further described by an instance, a command, and parameters to call
- Each CEWebS has a description that is shown as text on a menuitem
- Each CEWebS has an error text, that is shown when the service is not reachable
- Each CEWebS has a visibility flag which defines if a menuitem is shown for the service
- Each CEWebS has a usability flag that defines if a click on the menuitem triggers a call to the service, or the error message is shown
- Security is implemented through a key that is unique for an instance of a CEWebS. Only the TE that is in possession of the key can talk to the instance.

The XML data that is used to store this information looks as follows (in our prototype):

```
<service name='I-KNOW' visible='true' usable='true'>
  <error>This Service is currently under maintenance.</error>
  <description>I-KNOW</description>
  <wsdl>http://localhost:2017/Delivery/wsdl</wsdl>
  <instance>17</instance>
  <command>show</command>
  <parameters></parameters>
  <key>8a76450rr14d6d07b1740666e45f18dce9</key>
</service>
```

4.3.1 How a Document is Delivered to The User

When a user clicks on a menuitem, the TE decides according to the attribute usable, if the CEWebS is called or not. In general the application flow follows Fig. 5. For now we will concentrate on the actions, the objects that are generated during the processing will be explained in more detail later as they result from calling a CEWebS (and therefore just represent the return values of the methods contained in the interface).

Our prototype is capable of providing four services:

- Authenticate a user and manage the session cookies
- Change a password and distribute the changes through the notification interface
- Translate all user action (inclusive input from HTML forms) to a form that is suitable as an argument list for a call to a CEWebS
- Transform the results of a call to a CEWebS into a form that is suitable for the client application (HTML, WML, ...)

The most complicated part, although not necessary when all participated servers share a local network, is performance optimisation. I will now analyse the steps in Fig. 5 to explain what is happening, after the webpage that embeds the TE is called:

"generate menu": The TE decides if the XML file that holds the data about the contained services is newer than the menu in the cache. If this is true then it generates a static variant of the HTML that is used to show the menu. The HTML version is used until the structure inside the TE is changed, which occurs rarely.

"select menuitem": If a user enters the webpage for the first time, the first menuitem is selected (and hence the first CEWebS asked for data). The whole process is supported by a parameter called *target* which is contained in each link that has something to do with the TE (and therefore with calls to a CEWebS). If this parameter is not present the CEWebS assumes that the user just entered the page.

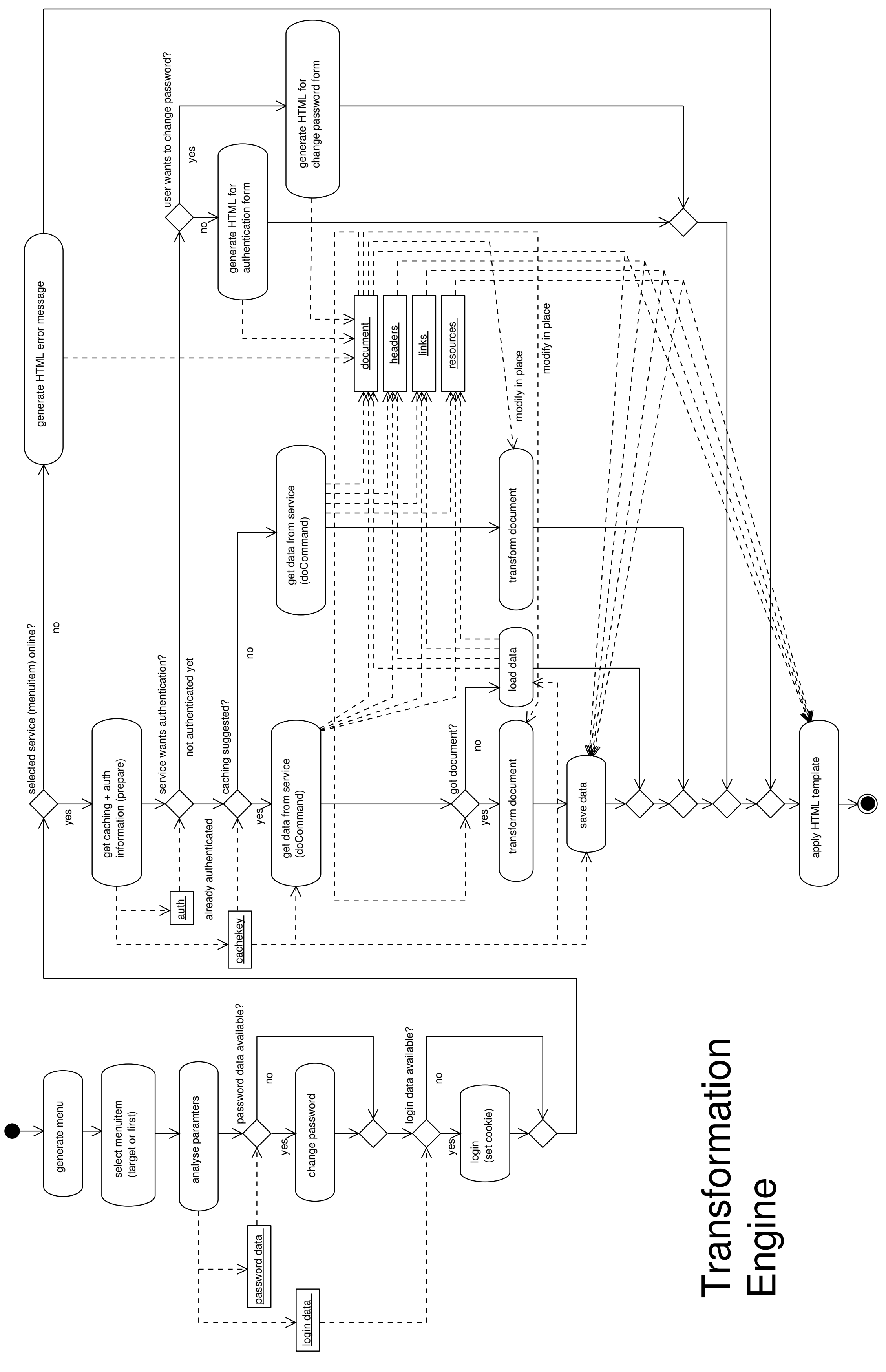


Figure 5: The General Principles for Getting a Document From a CEWebS

”analyse parameters”: All parameter with a *CEWebS_* prefix are collected. The *CEWebS_* prefix is cut off and the names and values are prepared (put into an array) to send to the CEWebS.

”password data available”, ”change password”: If the parameters user, key, keyNewA, and keyNewB are available, the TE writes the changes to its user repository and sends them to the NoMan.

”login data available”, ”login”: If the parameters user and key are present, the TE assumes that the call is the result of a login, checks the parameters against the information in the user repository and possibly sets a cookie with a random identifier (the random identifier is also saved to identify the user for subsequent calls).

These steps are done in preparation for the real interesting things that follow. At first glance Fig. 5 looks quite complicated, but it turns out to be really simple. Every data that is generated by or returned from the CEWebS is stuffed into four objects. They are involved with every activity that follows. The most important one is *document* which holds the HTML/WML that the user will see in the end. Additionally (and not contained in the diagram to keep it simple) every activity has an error handler. If an error occurs, the error message is put into *document* and the flow continues at *apply HTML template*. However this is only modelled for ”selected service (menuitem) online?”.

”selected service (menuitem) online?”, ”generate HTML error message”: To check if the CEWebS is reachable, a socket is created. If the test fails the text contained in the element *<error>* (from the above XML example) is put into *document*.

”get caching + auth information (prepare)”: The CEWebS is called to get information ...

- if the service demands authentication for the requested data
- if the requested data is cacheable (and possibly still cached by the TE)

”service wants authentication?”: A decision takes place, regarding the authentication data returned from the service.

- if the user is not already authenticated a HTML authentication form is put into *document* (step **”generate HTML for authentication form”**)
- if the user is not already authenticated and requested to change his/her password on the authentication form, the form for doing so is put into *document* (step **”generate HTML for change password form”**)

”caching suggested?”: A decision takes place, regarding the cachekey returned from the service. It the service returned an empty string it is assumed that no caching is desired, if a string is returned the TE looks up the data assigned with this key in the local cache. The cache only consists of the keys, the assigned data, and a time that gives evidence about the last change of the data. The two branches differ only in **”got document?”**, **”load data”**, and **”save data”**. In the case caching is demanded ...

- **”got document?”** checks if data was delivered by the CEWebS. If so, data is saved after a transformation, otherwise the data is loaded from the cache by using the cachekey.

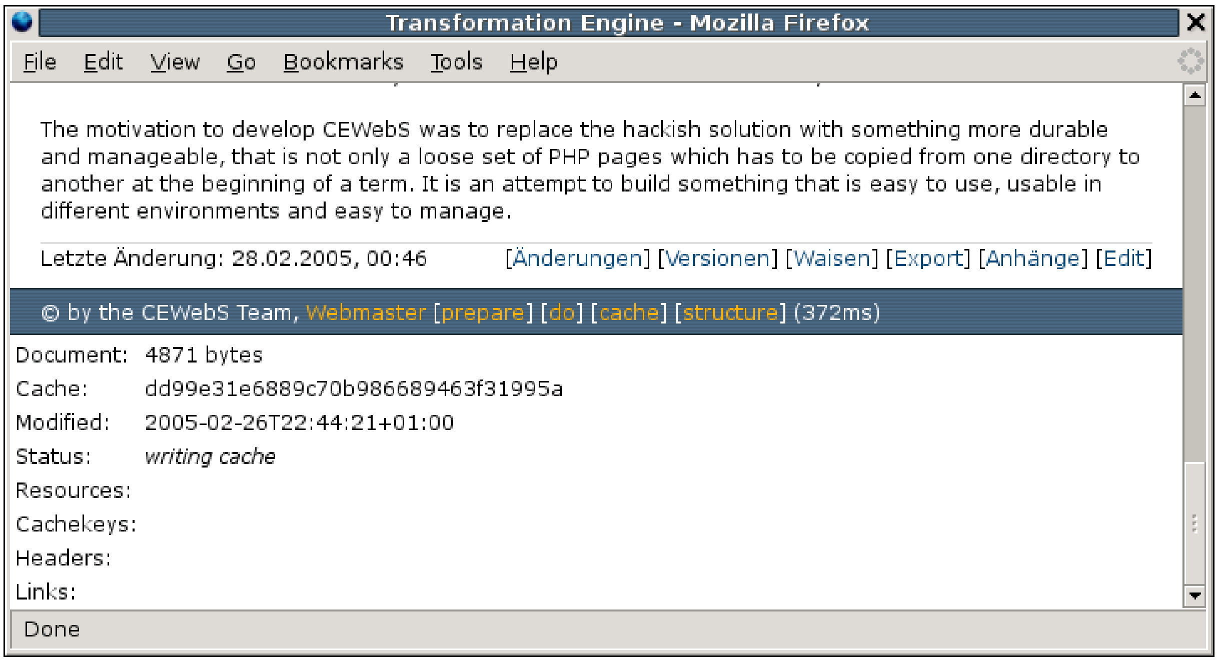
”get data from service (doCommand)”: The CEWebS is called to get data. In the case that caching was requested the time of the last data change is supplied to **doCommand**. The CEWebS can then decide if it wants to return XML data or nothing (in which case later on the document from the cache is used). The returned data has the following parts:

- **”document”** - a XML document in a XHTML derived format
- **”headers”** - HTTP headers that should be sent before delivering the final document to the user. This is useful when doing page redirects or wanting to send more additional headers (e.g. for serving P3P compact policies [40], a system that allows websites to describe how cookies are used, or what information they hold)
- **”links”** - additional information describing the served page (e.g. links to RSS [29] feeds)
- **”resources”** - resources used by the final page, including embedded images and objects (Java [30] applets, Flash [16] applications, ...)

”transform document”: The data returned from the web-service is transformed to meet the user’s requirements. Supported formats could be HTML, WML [26] or even targets like XUL [24] or XAML [21]. The transformation can be done by using XSL or some other XML transforming technology. In the actual implementation I do not modify the XML at all (as it is perfectly valid XHTML), only the CEWebS:// links inside the document, that are used to navigate inside a CEWebS or between the CEWebS inside a TE, are transformed to valid HTML links that call the actual TE (by using Regular Expressions [34]).

”apply HTML template”: This is the last step in the process of generating the final document. The content of all objects (document, headers, links, resources) is put together to form the final result that can be delivered to the users device (e.g. browser). The actual implementation supports also built in debugging of the data flow.

For the developer the actual implementation of the TE contains various debugging facilities (see Fig. 6). The user can view the SOAP Messages exchanged for both, *prepare* and *doCommand*. There is also a menupoint *Structure* that shows the returned XML document before the transformation and errors in the document (it is checked against a DTD). There is also an indicator


Transformation Engine - Mozilla Firefox

The motivation to develop CEWebS was to replace the hackish solution with something more durable and manageable, that is not only a loose set of PHP pages which has to be copied from one directory to another at the beginning of a term. It is an attempt to build something that is easy to use, usable in different environments and easy to manage.

Letzte Änderung: 28.02.2005, 00:46 [Änderungen] [Versionen] [Waisen] [Export] [Anhänge] [Edit]

© by the CEWebS Team, Webmaster [prepare] [do] [cache] [structure] (372ms)

```
Document:   4871 bytes
Cache:      dd99e31e6889c70b986689463f31995a
Modified:   2005-02-26T22:44:21+01:00
Status:     writing cache
Resources:
Cachekeys:
Headers:
Links:
```

Figure 6: Debugging a CEWebS

that tells how long the whole process took to complete (in the example it took 0.372 seconds). Of special interest is the point *Cache* that indicates how the caching was handled according to the above described principles:

- **Document** describes the size of the delivered document
- **Cache** shows the cachekey, that is returned by the CEWebS (or nothing)
- **Modified** shows when the (returned) document associated with the cachekey was modified the last time
- **Status** indicates how the document was served. Values include:
  - **From Cache** when the document from the cache was up-to-date
  - **Stream through** if the CEWebS demanded to not cache the document
  - **Writing Cache** if caching was requested and the CEWebS returned a document
- **Resources**, **Headers**, **Links** hold values according to the above described concepts
- **Cachekeys** holds keys for each file that was submitted to the CEWebS in a precedent step (this concept is described later in more detail)

4.3.2 How File Up-/Downloads Are Handled

The TE also handles file up- and downloads and supports them through caching. The whole process is handled by the same mechanisms that also handle document delivery. In fact the

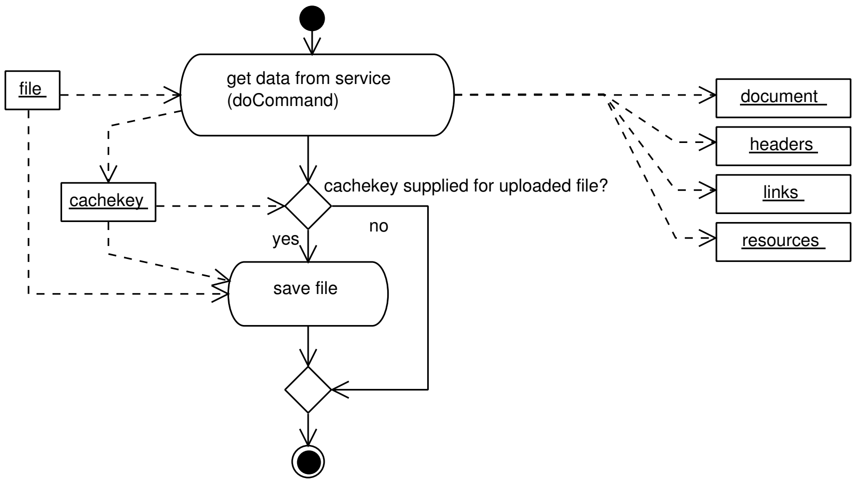

file
get data from service (doCommand)
document
headers
links
resources
cachekey
cachekey supplied for uploaded file?
no
yes
save file

Figure 7: Upload a File to a CEWebS

case described in Fig. 7 is just an addition to Fig. 5. The step **get data from service (doCommand)** additionally submits a file to the CEWebS. The CEWebS then returns a cachekey and some additional information. The file can then be saved to the cache immediately. This has the advantage that even when the file is requested for the first time it can be served from cache. Therefore the data that must be transferred between a CEWebS and the TE is cut down dramatically. For sure the CEWebS can also decide that the file should not be cached.

The opposite case is downloading a file from the CEWebS (see Fig. 8). In this case the all steps between **get caching + auth information (prepare)** and **apply HTML template** from Fig. 5 are replaced, although the control mechanisms for caching and authentication are the same. In the following I want to describe some steps that differ in more detail:

"query service (doDownload)": If caching is demanded then the CEWebS is queried by using the alternate **doDownload** method. The CEWebS return among others:

- **file data** - the actual file
- **name** - the filename associated with the file
- **mime** - the mime-type associated with the file

"apply DATA template": The download is prepared for the user by adding the additional information in the form of HTTP headers to the data

4.3.3 CEWebS Links

For navigating inside a CEWebS, it provides **CEWebS://** links, that contain a command and a set off attributes, to allow for the communication with different commands within the CEWebS.

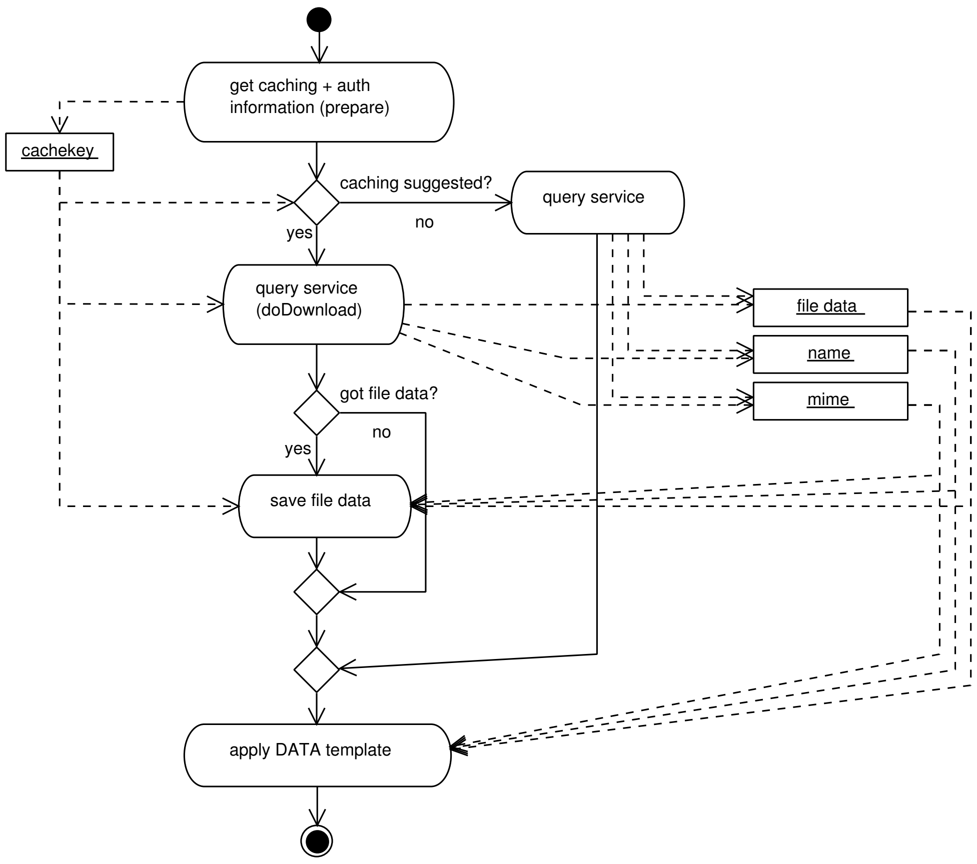


Figure 8: Download a File from a CEWebS

These CEWebS:// links are then transformed to call the actual TE. The links allow the service to function analogous to the way ordinary web-applications work. They provide the functionality of:

- Links to navigate between pages (**commands**), probably with multiple parameters attached (HTTP/GET)
- Action targets of forms (HTTP/POST or HTTP/GET)

CEWebS work in the same, but location independent way. They link relatively to the web-service that generated them, or in fact relatively to the TE they are used in. The anatomy of a CEWebS:// link is quite simple. For example, *CEWebS://self/document/details&CEWebS_id=1* is interpreted as follows:

- *self* tells the TE that the WS wants to display itself. Another possibility is that the WS wants to place a link to another WS in the container by referring to its name. A CEWebS by default does not know about the other CEWebS in the TE, the maintainer of a certain group (e.g. a course) must explicitly tell a CEWebS a name as configuration parameter. As we will later see, certain services rely on other services.
- *document* tells the TE that the result is supposed to be a document.
  - The second possibility is *download*, in which case the doDownload command is used for data retrieval.
  - The third possibility is *resource* followed by the name of a resource that has been returned by *get data from service (doCommand)*, followed by the name of the resource. This CEWebS:// link is translated to link to the resource-cache directory of the particular (TE +) CEWebS.
- *details* is the command (i.e., the requested functionality inside the addressed WS).
- *&CEWebS_id=1* is an additional parameter that is passed to the WS. Parameters have to start with *CEWebS_* as to not interfere with the standard HTTP/GET and HTTP/POST parameters used by the surrounding environment.

4.3.4 The Format of the XML Data Provided by a CEWebS

The document format of the XML data returned by a CEWebS is together with the stable CEWebS Delivery interface the key to keep the CEWebS usable by an arbitrary TE and therefore embedable into an arbitrary environment. The document format is derived from XHTML and is stripped of most elements, but every document following the structure is in fact valid XHTML 1.0. I decided to do this because:

- I wanted to provide the writers of new CEWebS with something they know very well, so that they do not need to learn a new type of markup

- I wanted to provide the writers of a new TE with something they know, and also allow for the ability to see output in the browser without a transformation.
- I stripped down XHTML because I think that most elements are not needed for producing the necessary output. The simplicity is also an advantage when creating new XSL transformations.

The following excerpt from the DTD shows the most basic concepts.

```
<!ENTITY % intext       'b|i|big|small|span|a|br'>
<!ENTITY % formelements 'input|select|textarea'>
<!ENTITY % headings     'h1|h2|h3'>
<!ENTITY % lists        'ul|ol'>
<!ELEMENT content       (%headings;|%lists;|dl|hr|div|table|
                        script|form)*>
<!ELEMENT form          (%headings;|%lists;|dl|hr|div|table|
                        script)*>
<!ELEMENT div           (#PCDATA|%intext;|div|img|%lists;|dl|
                        %formelements;)*>
<!ELEMENT td            (#PCDATA|%intext;|img|table|
                        %formelements;)*>
```

The root element of the document is **content**, which can only hold very few elements. In fact only headings (h1, h2, h3), lists (ul, ol, dl), horizontal rulers (hr), div's (as general structuring element), script sections and form's are allowed. **Form**'s can hold the same elements like content, with the exception of other forms. The elements that hold formated text are **div**'s, **table cells (td)** and **list item's (li)**. These elements can hold text mixed with basic formating elements defined in **intext**. The attributes of the elements are stripped down to a set that is defined as mandatory by XHTML. A characteristic is, that no element can have a **style** attribute, but there is a **class** attribute that can hold a wide range of predefined values to style the elements.

Adding **script** elements to the structure is sort of a dilemma. There is no guarantee that the script code in the element can be interpreted on the target system (e.g. the browser or a XAML application). So one must use this element very carefully to not break the portability of a CEWebS. There is one rule that must be obeyed when providing for example javascript with a document: Make sure that the application does not depend on the the script. Here are some examples:

- When the script provides a page redirect, make sure that the user has an alternative possibility to change to the target of the redirect (e.g. a link)
- When a script is used for the convenience of the user (e.g. to check form input), make sure the form depends not on the javascript. Use onSubmit to check the input, never use javascript links (like <a href="javascript:"/>) or buttons. Everything should always work with scripting disabled, so checking the input data has also to be implemented on the target of the form.

These rules are in fact not native to CEWebS, they apply to every well designed web application. A XML schema file, as a replacement for the DTD, to check every aspect of the document structure including the script rules, is a goal for future implementations of the TE.

4.3.5 The Structure of An Example Implementation

We are working currently with an example implementation of the TE that provides a very simple portal for our students. All courses have their own URL (directory on the server) and are grouped by course name and term they were (are) held. The TE is linked to a global location and is reused via URL rewriting by each course. URL rewriting is a technology, where upon access to a specific directory, the request is redirected to a different location. The user is not aware of this change, because the address in the browser is not changing. With this mechanism we can separate the data (cache and course information like name, web-master, services, ...) from the TE. The TE resides only once on the server, and is not duplicated for every course. This proved to be usable as lightweight replacement for a full-fledged platform that holds multiple courses. The structure of the TE is as follows:

```
./Images + ./Styles
./Output/Intl
./Output/Intl/default
./Output/dataTemplate.php
./Output/htmlChange.php
./Output/htmlTemplate.php
./Output/htmlAuthenticate.php
./Includes/admin.php
./Includes/command.php
./Includes/transform.dtd
./Includes/transform.xml
./Includes/transform.php
./Includes/transform_document.php
./Includes/transform_file.php
./index.php -> ./download.php
```

The **Images** and **Styles**, needed for the design are held separately in two directories, all links in the style sheets and the htmlTemplate are absolute to the server in order to avoid problems with the URL rewriting. The **Output** directory holds the templates for generating the final document and the translations (**INTL**) as the *login* and *change password* masks are localised to different languages. The **default** directory inside **INTL** is mandatory and holds a link to the default translation (in our case *de*, abbreviations for standard language encodings are used).

The **Includes** directory holds the logic for the TE.

- **admin.php** provides basic functionality like XML wrappers and a way to **analyse parameters** (see Fig. 5).
- **command.php** provides the ability to call the CEWebS (**prepare** and **doCommand / doDownload**) and generates also the information that is used for debugging purposes.

- **transform.php** holds the base class for all things related to the transformation of the data from the CEWebS. **transform_document.php** and **transform_file.php** hold derived classes for data returned from *doCommand* or *doDownload*.
- **transform.xml** and **transform.dtd** are used for generating the debugging output of the document structure (see Fig. 6)
- **index.php** and **download.php** are the same file (*download.php* is a link to *index.php*) and handle the two cases *doCommand* and *doDownload*. By calling one of them, the user can either download a file or view a document.

The **Data** directory has to be present at every course location (directory), but not in the TE directory:

```
./Data/CacheD
./Data/CacheF
./Data/CacheU
./Data/CacheM
./Data/CacheDH
./Data/CacheDL
./Data/CacheDR
./Data/CacheFH
./Data/participants.xml
./Data/services.xml
./Data/name.html
./Data/contact.html
./Data/description.html
```

Most cache directories hold the cached data separate for every service, except *CacheM* and *CacheU*.

CacheD holds the documents, in the form of files with the cachekey as filename, and the time of the last modification (from the server) as file modification time. **CacheF** is the equivalent for files. **CacheU** holds the unique keys that are used as cookies to identify a user. **CacheM** holds a static version of the menu and also extracted, easily includable information for each menupoint (so that the services.xml has only to be parsed when it has changed). **CacheDH** and **CacheFH** hold the headers that were assigned to a specific document file. CacheDL and CacheDR hold the links and resources that were returned with the particular document.

name.html, **contact.html**, and **description.html** hold additional information that produce the title and footer lines for the final document. **participants.xml** holds the user data and passwords, **services.xml** holds the information about the services as described above.

4.3.6 Performance

To give the reader at least a vague idea of the performance of the TE I put together some benchmarks, measuring the performance of the current implementation. I measured the time that was needed to carry out the steps between **selected service (menuitem) online?** and **apply HTML template** (see Fig. 5). The time is printed (as described before) to the final page as part of the debugging facilities for developers. I got some interesting numbers for the benchmark.

I had the chance to run the test on two servers, our own department server and a server running at the Masaryk University in Brno. I want to thank Tomas Pitner, Pavel Drasil and Jan Pavlovic for making possible to carry out the tests. Both servers ran exactly the same software:

- PHP 4.3.10 with DOM XML (using libxml 2.6.10) and NUSOAP 0.6.8 (Revision 1.80)
- Two CEWebS using Ruby 1.8.2, Webrick 1.8.2, Soap 1.8.2 and ruby-xml-simple 0.1.6

The configuration of the servers was as follows:

Leonardo: our department server

- 2 x Intel(R) Xeon(TM) CPU 3.06GHz
- Kernel 2.4.21 SMP
- Debian Sarge

Kore: the server at the Masaryk University in Brno

- 2 x Intel(R) Pentium(R) 4 CPU 3.20GHz
- Kernel 2.6.10 SMP
- Fedora Core 3

I decided that the memory configuration was irrelevant for our test, so it is not mentioned above. Both servers should be quite equal, although I'd given Kore some advantages for having a kernel known for better overall performance. Not all tests could be run on both computers, because the WIKI used for the test, so far only runs in Vienna. All times are the average of 10 subsequent calls.

Results for the HelloWorld CEWebS:

```
Leonardo calling local Hello World: 108ms
Kore calling local Hello World:     142ms
Kore remote local Hello World:      252ms
```

Results for the the Scientiki CEWebS (the implementation of a WIKI web), delivering a big page (in fact the *Architecture* chapter of this thesis):

```
Leonardo calling local WIKI:          136ms
Leonardo calling local WIKI uncached: 470ms
Kore calling remote WIKI:             273ms
Kore calling remote WIKI uncached:    480ms
```

I am interpreting the findings as follows:

- The performance is quite good, the users do not even notice that web-services are called. This is also true for remote web-services, that do not run in the same LAN.
- Caching is a very effective (and necessary) way to reduce traffic and delivery time.
- Either Debian Sarge is more optimised than Fedora Core 3, or the Xeon processor really makes a big difference for our tasks.

I think by using PHP5 with the new built in SOAP and XML facilities, performance can be improved even further, because NUSOAP's marshalling/demarshalling is known to be slow and memory consuming, especially when large amounts of data are transfered (e.g. file transfers).

4.4 The Delivery Interface

The Delivery interface (see Fig. 9 is the one part of the architecture that allows the users to interact with the system (Report and Administration interfaces are used only by applications for administration and controlling/grading). The interface is shared by all CEWebS and was kept simple to allow for easy creation of new services. A WSDL is provided, which was designed with a maximum compliance in mind. The WSDL file has been tested with:

- Apache Axis 1.1, http://ws.apache.org/axis/
- Microsoft .NET Framework 1.1., http://www.microsoft.com/Net/Basics.aspx
- Ruby 1.8.2 soap module, http://www.ruby-lang.org
- Perl SOAP::Lite 0.6.4 module, http://www.soaplite.com/

The Delivery interface of a CEWebS is basically given by the requirements of the TE, as elaborated in the previous chapter. There are basically three methods. **prepare** has to be called before everything else, **doCommand** and **doDownload** are used to get documents or binary files from the CEWebS. All three methods essentially share the same parameters with the exception of **modified**. The modified parameter is only present in *doCommand* / *doDownload*.

I am using a simple **Hello World** CEWebS to show how the Delivery interface works. Our CEWebS returns just a simple string, but in different languages. There exist multiple instances of the CEWebS with their own localisations (for the same language two instances of the CEWebS can return different strings). The string should be cached in the TE.

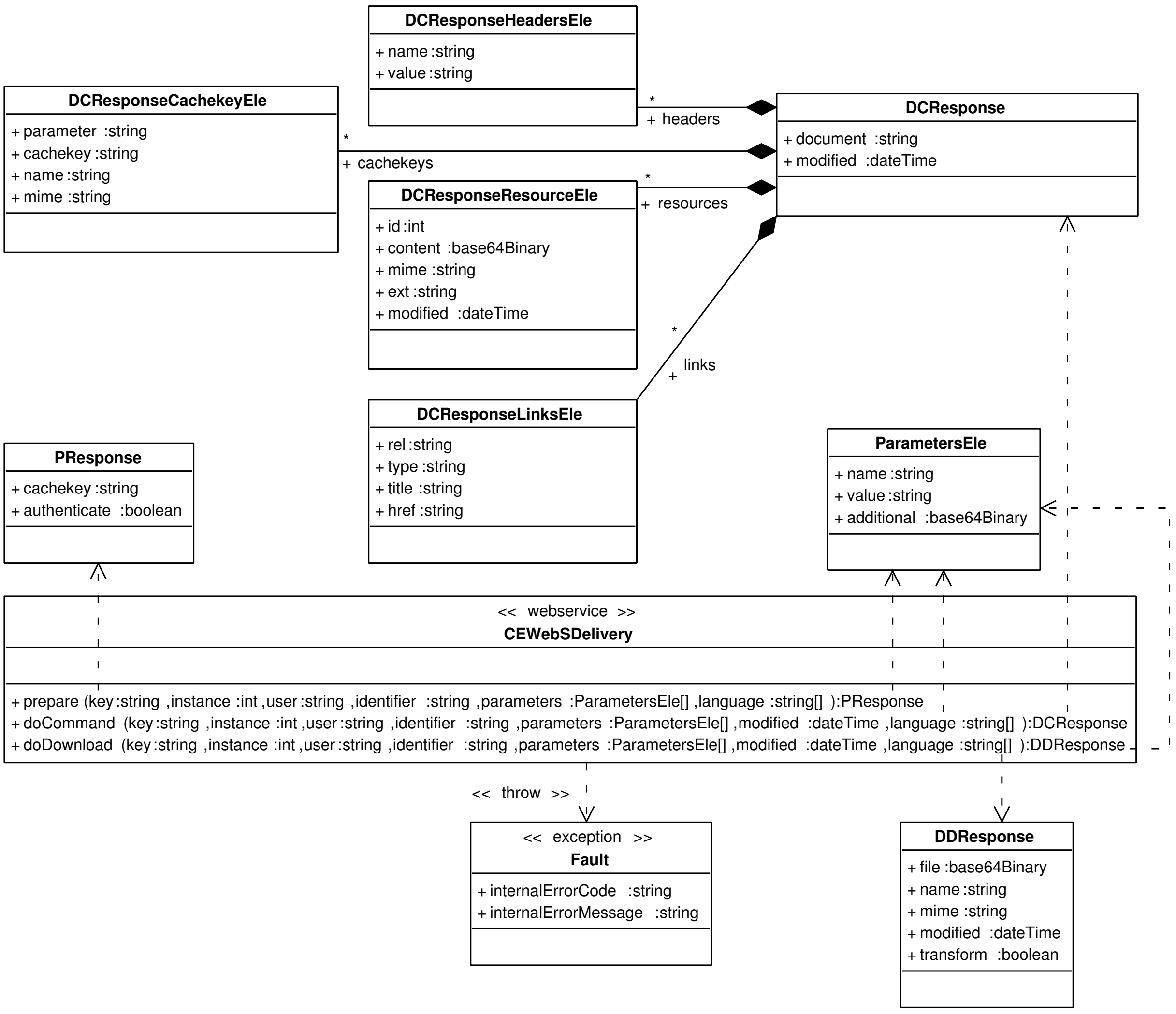


Figure 9: The Delivery Interface

The first thing that the TE calls is the **prepare** function to check if the service demands caching (in our case it will) and / or authentication. I will now explain the parameters, the method is called with, in more detail:

- **key** holds a unique identifier. The CEWebS can use this key and compare it to a locally saved version to find out if the calling TE is really allowed to do so. It acts as a poor mans certificate.
- **instance** holds a numeric value, that identifies the instance. This can also be ignored by the CEWebS if it implements only one instance. Instances are a way to realise e.g. multiple independent discussion forums, and to allow multiple independent TEs to include them.
- **user** holds a string that is supplied by the TE to identify the current user (can also be empty if the user is not yet identified). There is no password checking involved in the CEWebS itself, it trusts the TE that it securely identifies the users. All CEWebS have the information about the users that have access to the TE saved locally. They can use **user** to get more information (by default this includes forename, surname and email address as we will see later).
- **parameters** holds all parameters that the TE collected, either via HTTP/GET or via HTTP/POST. In the current implementation uploaded files are not part of the call to prepare. The parameters each have a **name** and a **value**. The attribute **additional** is reserved to hold the content of a file.
- **language** is an array of strings that holds the language preferences that are set in a users browser. This would be e.g. [”en-us”, ”en”, ”de”, ”cz”]. The format of the language string is subject of RFC 2616 [9], the array is just passed to the CEWebS from the **http-accept** HTTP header. The CEWebS has to select a language from the array, for which it has a localisation available, or else fall back to an internal default language.

An object with two attributes is returned that holds a **cachekey** and a boolean value that denotes if authentication for the given combination of instance, command (...) is desired. *cachekey* holds a string that functions as a key to a per service cache in the TE. If an empty string is returned, the CEWebS demands that the document is NOT cached. The key should be constructed to reflect at least the unique combination of language, instance, and command. I'll give some examples to explain the concept:

- In our service we have several instances and several languages
- Lets assume the *Hello World* string is printed when the parameter command has the value **show**
- Because there is only one command (*show*) the output of our service can be uniquely identified by a combination of *instance* and *language*
- It is wise to calculate a hash of the two values to enhance security and also guarantee that no special characters are included. That can prevent problems when the TE is using the string as a filename for a disk-based cache.

An example implementation of *prepare* for our *Hello World* CEWebS would look like:

```
def prepare(key,instance,user,identifier,parameters,language)
  case identifier
    when 'show'
      PResponse.new(Digest::MD5.hexdigest(
                      instance.to_s +
                      Language.select(instance,language)),
                    false)
    else
      raise Fault.new('NotFound','command is not available')
  end
end
```

As you can see the method only consists of a big *case* that selects the return for every command, according to the principles described above.

With the *cachekey* the TE can now supply a **modified** to subsequent calls - *doCommand* or *doDownload* (the *modified* is a value that is associated with the cachekey in the TE). When calling one of these methods the TE also adds uploaded files to the *parameters* array. In the case of a file upload, *name* holds the name of the form element, *value* the name of the file and *additional* the contents of the file.

When doing a call to *doCommand* the localised string is loaded from a data-source and returned to the user. All collected information is contained in an object of type DCResponse. The TE should interpret an empty attribute **document** as signal, that it can use the version from the cache (this also includes **headers** and **links**). **resources** and **cachekeys** should be handled separately, as returning no document does not mean, that embedded images could not change, or uploaded files should not be put to cache.

We will now concentrate on the details of the DCResponse object:

- **document** holds a document in a custom XML format, as described in the previous chapter, or an empty string
- **modified** holds a dateTime string denoting when a document / link / header changed. A document with its headers and links is seen as an entity, so there is only one *modified* to describe them all. When an embedded image changed and has to be sent back to the TE, the modification date for *document*, *headers*, and *links* stays the same, as resources are independently controlled. The document has not to be delivered when just a resource changed, given the case that the document is still valid.
- **headers** can hold a set of name / value pairs that are sent by the TE as HTTP headers before the final page is delivered. This allows the web-service to take full control of the delivery process. The possibilities are again described in RFC 2616. It can for example do page-redirects with the *location* header ([:name => ”location”, :value => ”http://xxxx”] to cause the effect that the users browser immediately changes to a different location after receiving a page. This can be used to redirect the user after submitting a form, so

that when he/she hits the reload button the form is not submitted again (cause he/she is already no longer on the page that was the target for the HTTP/POST).

- **resources** is an array that can hold images or objects embedded in the document. The delivery of a resource depends also on the *modified* parameter of the method. They are handled the same way as documents. When present they are saved to cache, when not present it is assumed that they are still valid. All *resources* are always saved to the cache. They are referenced from the document by links of the form *CEWebS://Data/CacheDR/doc/1.png*. To describe a *resource* the following attributes are mandatory:
  - the **id** (name) of the string (1 in the above example)
  - the *'content* of the resource (e.g. content of the image file)
  - the **mime** type of the resource (image/png in the above example)
  - the extension (**ext**) of the resource (png in the above example)
  - **modified**, a dateTime string that denotes the last change date of the resource
- **links** can hold information to construct *<link/>* elements in the *<header>* of a final page. This is useful when one wants to provide simple logical information how pages relate to each other. This can be used by browsers to get an idea how single pages are connected and to show an alternate navigation (this is only implemented by conceptual browsers). It is also useful to provide a user with the information that there is an alternate RSS feed for the page. The attributes have to be set according to XHTML rules. An example to link to an RSS feed would be:
  - **rel**: alternate
  - **type**: application/rss+xml
  - **title**: Discussion
  - **href**: CEWebS://self/download/rss&amp;CEWebS_id=42
- **cachekeys** is an array that holds the keys for files that were handed to the *doCommand* method. This makes it possible that the TE can cache the files immediately after uploading them. When other users request the files they are already in the cache. To make this possible the following attributes have to be set:
  - **parameter**: name of the form element that was responsible for the file upload
  - **cachekey**: the key that should identify the file in the cache
  - **name**: the filename that is associated with the file (has not to be the same as the name of the file submitted by the user)
  - **mime**: the mime-type for the file. In the current implementation the mime-types are selected automatically according to the content of the file. So even submitting a file with a wrong extension leads to a correct mime-type and possibly a file rename (new extension).

A sample implementation of our *HelloWorld* service can be found below:

```
require 'Delivery/ImplShow'
def doCommand(key,instance,user,identifier,parameters,modified,
            language)
  case identifier
    when ’show’
      cmd = Delivery::ImplShow.new(instance,modified,language)
      return DCResponse.new(cmd.document,cmd.modified)
    else
      raise Fault.new(’NotFound’,’command is not available’)
  end
end
```

As you can see, the method again consists only of a construct that returns the *DDResponse* for each command. In our current implementation the response is constructed by creating an object for each command, that decides if it is necessary to return a document and provides the necessary data if so.

The *doDownload* method works exactly the same as the *doCommand* method, but it returns a **DDResponse** object that can be described as follows:

- **file** contains the content of the file. If it is empty, it is assumed that the version from the cache is up-to-date.
- **name** contains a filename for the file.
- **mime** contains the mime-type that should be supplied with the download (to allow the users computer to automatically select an application for the file)
- **modified** contains a dateTime string denoting when the file has last be changed
- **transform** contains a boolean value that tells the TE if the content of the file should be searched for CEWebS:// links. If true all CEWebS:// links are translated to point to the actual TE. This is useful e.g. when the CEWebS provides an RSS feed that should link back to the contents of the CEWebS. Because the RSS XML document is a file and not a document in the CEWebS sense it is acquired through *doDownload*.

4.5 The Administration Interface

The *Administration* interface (see Fig. 10) has to be implemented by every CEWebS. The functions implementing this interface are responsible for the following:

- Creating and deleting instances
- Assigning users to a particular instance
- Providing information about what data (configuration) is required by the service to function properly
- Setting and querying this information

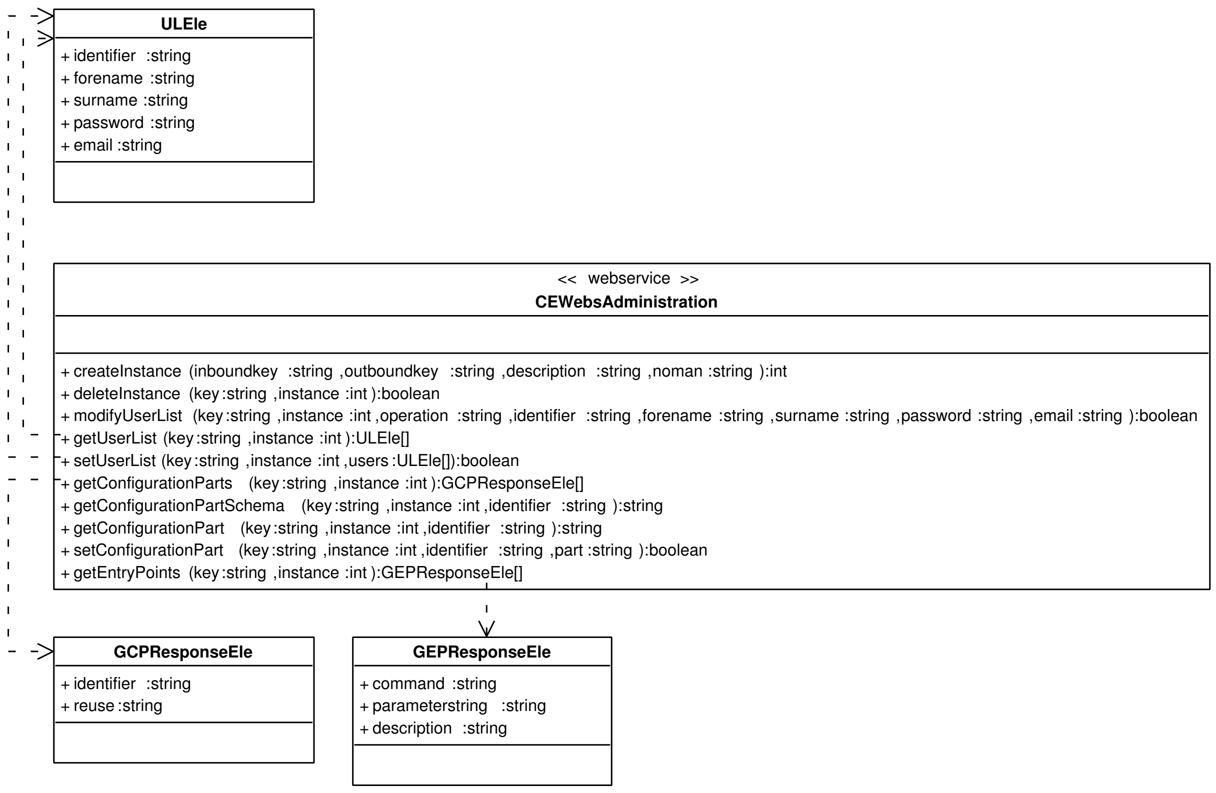

ULEle
+ identifier :string
+ forename :string
+ surname :string
+ password :string
+ email :string
<< webservice >>
CEWebsAdministration
+ createInstance (inboundkey :string ,outboundkey :string ,description :string ,noman :string):int
+ deleteInstance (key :string ,instance :int):boolean
+ modifyUserList (key :string ,instance :int ,operation :string ,identifier :string ,forename :string ,surname :string ,password :string ,email :string):boolean
+ getUserList (key :string ,instance :int):ULEle[]
+ setUserList (key :string ,instance :int ,users :ULEle[]):boolean
+ getConfigurationParts (key :string ,instance :int):GCPResponseEle[]
+ getConfigurationPartSchema (key :string ,instance :int ,identifier :string):string
+ getConfigurationPart (key :string ,instance :int ,identifier :string):string
+ setConfigurationPart (key :string ,instance :int ,identifier :string ,part :string):boolean
+ getEntryPoints (key :string ,instance :int):GEPResponseEle[]
GCPResponseEle
+ identifier :string
+ reuse :string
GEPResponseEle
+ command :string
+ parameterstring :string
+ description :string

Figure 10: The Administration Interface

- Querying the service for entry points. Entry points are the same to CEWebS as index files are to websites. For a CEWebS it is useful to tell a user what *command*s would be wise to use when displaying the service. For example it would not be useful to display the target of a form when the user enters the CEWebS. Instead it would be useful to display a form where the user can enter data or a survey list that shows all records.

To keep the *Administration* interface separated from *Delivery* has the advantage that a system administrator can very easily set up a restriction to only allow *AMan*'s to access it. Presumably a CEWebS is accessed by many different TEs (the different instances), but only by few AMan's. The separation has also the advantage that the interface can be shut down separately for a CEWebS, so that it can act as an archive, where the configuration can't change any longer.

I now want to describe the methods included in the the interface in more detail.

createInstance: This method is used to create and initialise a new instance of a CEWebS. **inboundkey** contains a key, that has to be supplied with every call to the service, if the caller wants to gain access to the now created instance. **outboundkey** must be supplied by the service if it wants to contact the *NoMan*. The contact information is contained in **noman** in the form of a link to a WSDL file. The **description** string is used to supply a more verbose description. In the case of a problem (e.g. when the mapping is lost in the AMan) the instance should be recognisable by this description. The method returns an integer that identifies the newly created instance.

deleteInstance: This method deletes a particular instance. It needs the numeric instance identifier and the assigned inbound**key** as parameters. It returns a boolean to signalise success.

From now on caller has to supply the correct instance / inbound-**key** combination for the method to return data or execute a requested operation.

The following three methods deal with the setting of users. One of the purposes of an *AMan* is to set all users immediately after creating a new instance. The rest of the configuration can be dependent on the user data (e.g. if a CEWebS implements parts of courses, the users are divided into groups and teams). After the initial user import, the users list should not change again as whole. So there are two different methods to deal with these use-cases. One method is to set the whole list of users, and one to add / change / delete single users. In the following I will describe the functions that deal with user-data:

setUserList: This method is used to set a a large quantity of users at once. A boolean value is returned, that signalises if the operation was successful or not. Each user is represented by an object **ULEle**, which has the following attributes:

- **identifier**
- **forename**
- **surname**
- **password**
- **email**-address

modifyUserList: is a method that concentrates on single users. It has all the attributes of *ULEle* as parameters. There is an additional parameter **command** that tells the CEWebS what to do with the parameters. *command* is an enumeration of the following values:

- **ADD_OR_CHANGE** - when supplied, the data for a specified user (identified by the string *identifier*) is updated or added (if the user not yet exists).
- **DELETE** - the user identified by *identifier* is deleted. The values apart from *identifier* can be empty.

getUserList: For an **AMan** it could be sometimes necessary to check what users are present in a CEWebS (e.g. if an AMan wants to synchronise all services with a different datasource). The Method returns an array of the above mentioned **ULEle** objects.

Setting the users is (as already mentioned) the second step that is involved in creating a functional instance of a CEWebS. The next step is to supply the service with the necessary configuration. It is recommended that a CEWebS is always initialised with a default configuration set (during **createInstance**). So the following functions serve the purpose of querying and modifying this default configuration:

getConfigurationPart: The configuration is divided into parts than can be set individually. This method returns an array of **GCPResponseEle** objects that consist of **identifier** and **reuse** attributes. **identifier** is a unique identifier for the configuration part. There has to be noted that also the order in which the parts are returned can be important. A part with a higher index in the array, can depend on the information that is included in a part with a lower index. The *reuse* string is particularly useful for the *NoMan*. All CEWebS that share configuration parts with the same identifier and the same value in *reuse* are seen as interested in this type of information. So when a service publishes such a part to the ”blackboard” all other CEWebS in the same TE (that share the same configuration part with the same **reuse** string) are notified.

getConfigurationPartSchema: For every configuration part (identified by the string **identifier**) the CEWebS can return an XML Schema that describes the data in more detail. This schema can be used by the AMan to create a wizard the lets users add missing information. To support the AMan in the creation of the wizard, the XML Schema is enriched with information by using the **<annotation><appinfo/></annotation>** schema component. Specifying the exact information that has to be supplied is considered a future exercise.

getConfigurationPart, setConfigurationPort: After getting information about the structure of a particular part, the *AMan* can query the information itself. After changing the information the user can ask the AMan to save it back to the CEWebS.

When finishing these steps, the CEWebS should be fully functional. The remaining method has to be called to find a *command* that can be used by the TE to access the service. Lets imagine a CEWebS supports students in uploading multiple documents. When the students are divided into several groups, it can be useful, to show an intermediate page, where the students can select the group they belong to. When there is only one group of students, this page is useless, and the upload form can be immediately presented to the user. So ...

getEntryPoints: returns an array of useful *entry points* that can be used directly as menu points for a TE, or can be supplied (as part of a configuration) to any other CEWebS inside the TE, to allow for interlinking. The method returns an object **GEPResponseEle** that has the following attributes:

- **command**: The command that is necessary to access the functionality.
- **parameterstring**: A set of parameters in the form of **name=value** separated by **&**'s as described in the chapter about CEWebS links.
- **description**: A description to help the user select the right entry point.

4.5.1 The Service Manager (SerMan)

The sole purpose of the **Service Manager** (SerMan) is to register CEWebS (see Fig. 11). So it provides the functionality of a UDDI to the *Administration Website*. The services are registered by providing:

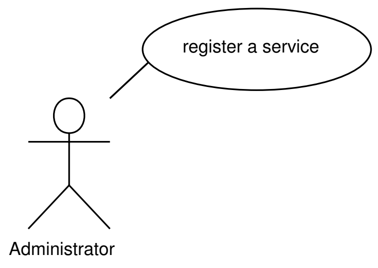


Figure 11: Use-cases For The Service Manager

- A name (short description).
- The location (WSDL) of the *Delivery* interface.
- The location (WSDL) of the *Administration* interface.
- Optional: the location (WSDL) of the *Report* interface.

In Fig. 12 one can see a screenshot of the current implementation of the SerMan. So far there are 18 services registered. Even the SerMan itself and the other CEWebS that the **Administration Website** consists of, are registered.

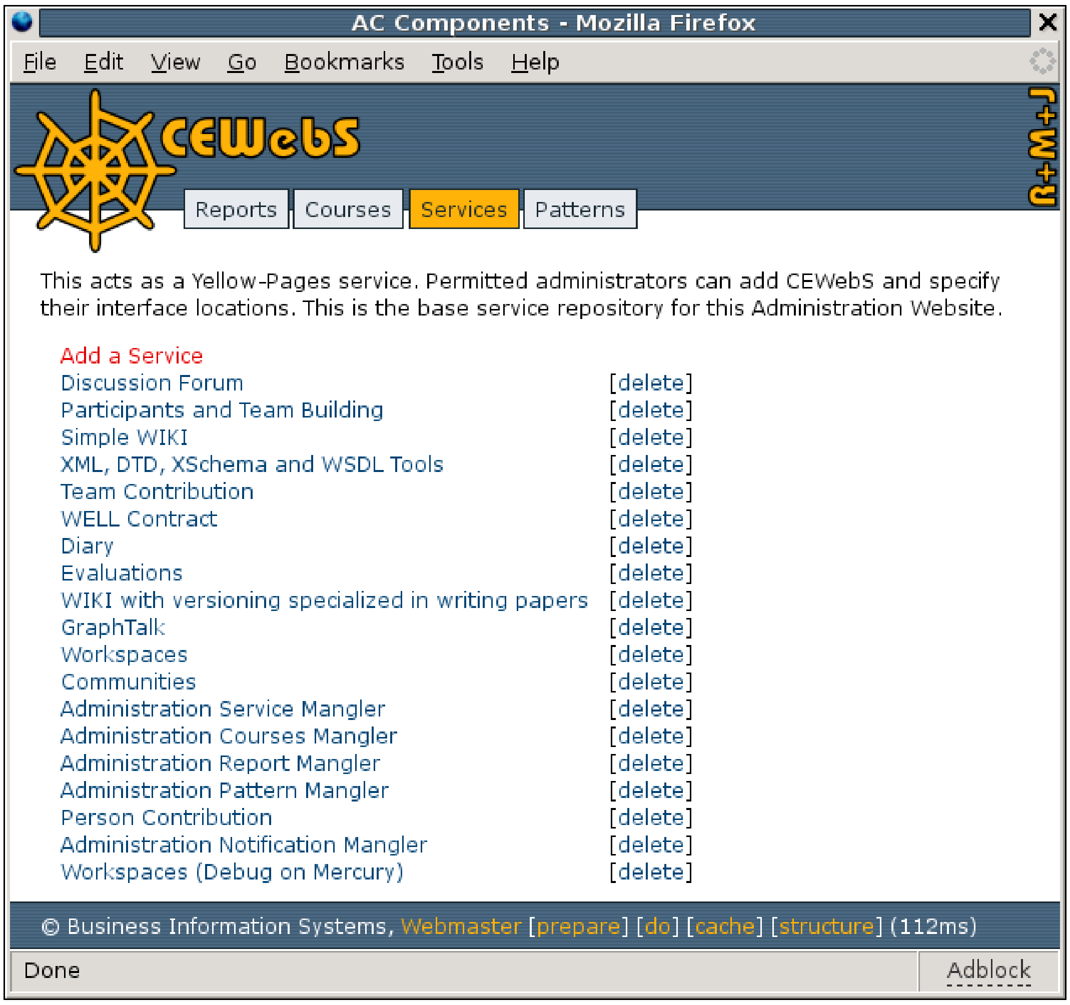
The information that is collected by the SerMan is shared with the AMan and ReMan CEWebS.

4.5.2 The Administration Manager (AMan)

The purpose of the AMan is to set up and manage courses (see Fig. 13). This includes ...

- registering courses locations (WSDL)
- initialising courses with information like course-name, webmaster, title-string (for HTML page)
- adding CEWebS to the course
- configuring a CEWebS
- setting the text of the menuitem that calls a CEWebS (as explained before)
- changing a menuitem to be invisible/visible for the user
- changing menuitem to be unusable (show an error message) / usable for the user

The user interface (UI) is designed to be as simple and straight forward as possible. When the administrator enters the webpage he/she can see a list of all courses that have been registered for the current term (see Fig. 14). The operations an administrator can perform include ...


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Reports Courses Services Patterns
R+M+J

This acts as a Yellow-Pages service. Permitted administrators can add CEWebS and specify their interface locations. This is the base service repository for this Administration Website.

Add a Service	
Discussion Forum	[delete]
Participants and Team Building	[delete]
Simple WIKI	[delete]
XML, DTD, XSchema and WSDL Tools	[delete]
Team Contribution	[delete]
WELL Contract	[delete]
Diary	[delete]
Evaluations	[delete]
WIKI with versioning specialized in writing papers	[delete]
GraphTalk	[delete]
Workspaces	[delete]
Communities	[delete]
Administration Service Mangler	[delete]
Administration Courses Mangler	[delete]
Administration Report Mangler	[delete]
Administration Pattern Mangler	[delete]
Person Contribution	[delete]
Administration Notification Mangler	[delete]
Workspaces (Debug on Mercury)	[delete]

© Business Information Systems, Webmaster [prepare] [do] [cache] [structure] (112ms)

Figure 12: The Service Manager

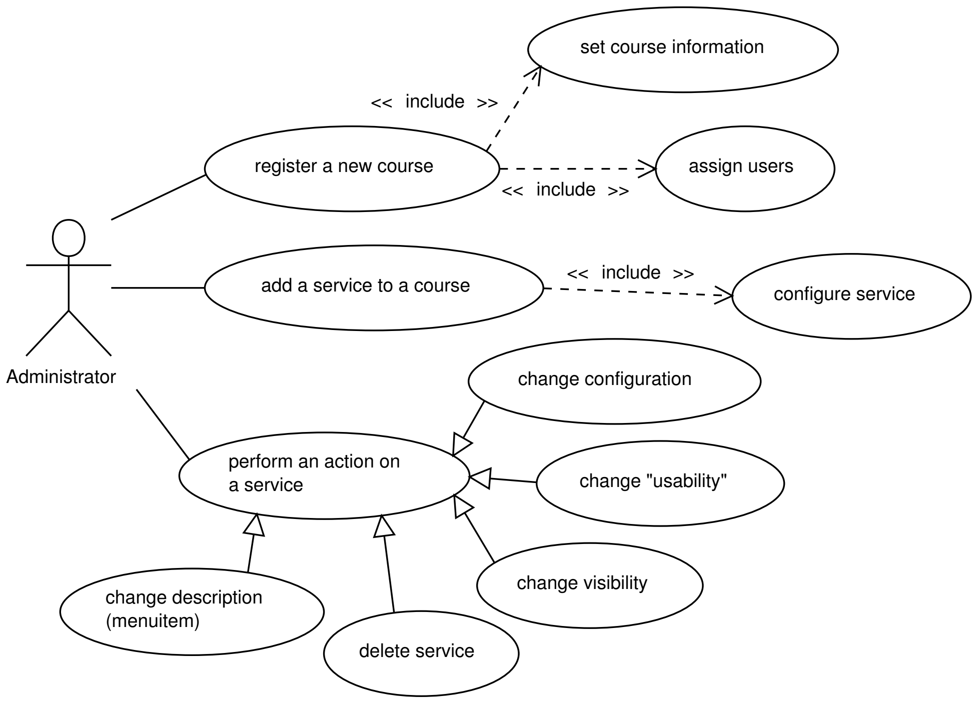
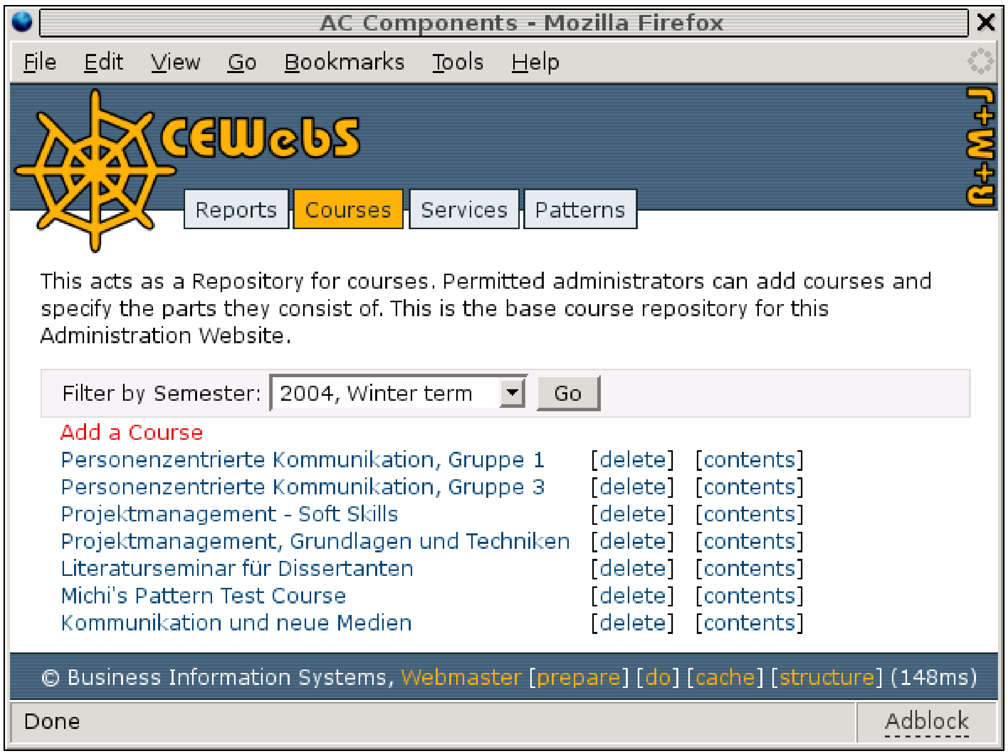


Figure 13: Use-cases For The Administration Manager

- adding a new course (**Add a Course**)
- deleting a course (**delete**)
- changing a course's preferences (**by clicking on the name**); the preferences include the course location (WSDL), contact information (webmaster), unique name (regarding the TE), text for the menuitem, and information about the term
- changing to the list of CEWebS that are including in the course (**contents**)

In Fig. 15 one can see, how a ready-to-use course looks from an administrator's point of view. The administrator has the chance to ...

- add / delete / import / synchronise users (**Manage Users**)
- add new (instance of a) CEWebS to the current TE
- **delete** CEWebS from the current TE (and hence also delete the instances in the CEWebS); deleting is not the right way if one just wants the students to no longer see a menuitem; deleting means loosing all data accumulated by the service
- change the information that is provided to the TE; this includes text of the menuitem, the order (position of the menuitem), an error message when the connection fails, and the command (+parameters) that is used when the menu item is activated.
- change the state of the menuitem to visible/invisible


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Reports Courses Services Patterns
R+M+J

This acts as a Repository for courses. Permitted administrators can add courses and specify the parts they consist of. This is the base course repository for this Administration Website.

Filter by Semester: 2004, Winter term Go

Add a Course

Personenzentrierte Kommunikation, Gruppe 1	[delete]	[contents]
Personenzentrierte Kommunikation, Gruppe 3	[delete]	[contents]
Projektmanagement - Soft Skills	[delete]	[contents]
Projektmanagement, Grundlagen und Techniken	[delete]	[contents]
Literaturseminar für Dissertanten	[delete]	[contents]
Michi's Pattern Test Course	[delete]	[contents]
Kommunikation und neue Medien	[delete]	[contents]

© Business Information Systems, Webmaster [prepare] [do] [cache] [structure] (148ms)

Done Adblock

Figure 14: The Administration Manager - The Courses

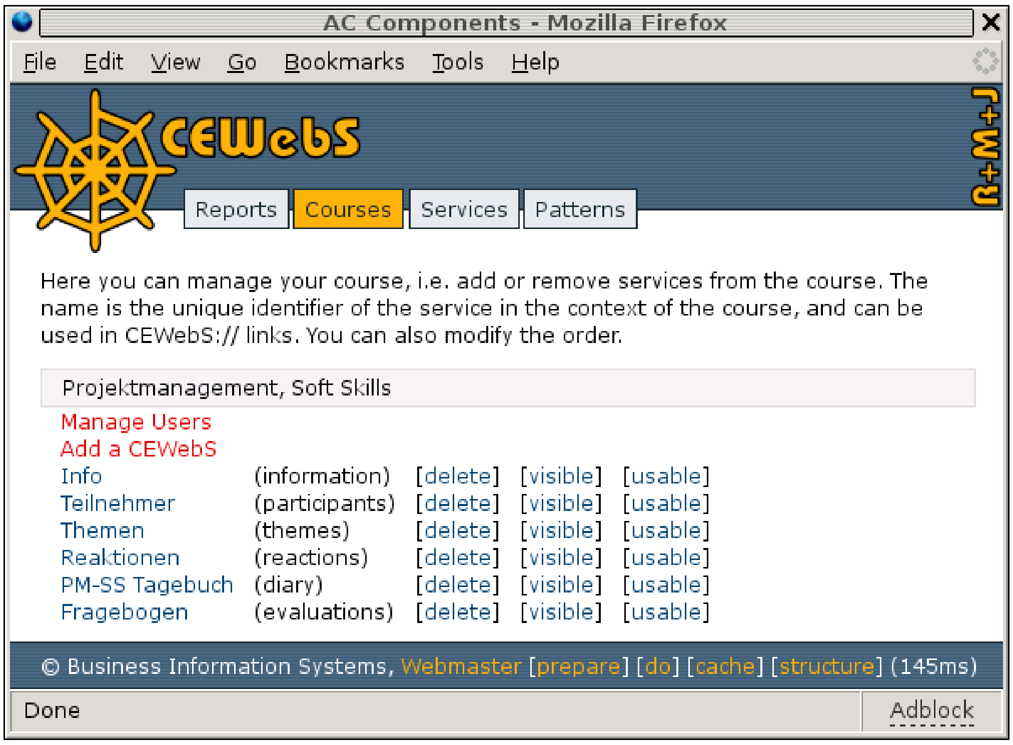
- change the state of the menuitem to usable/unusable (the menuitem is visible but the error message is always shown, no matter if a connection is possible or not)

4.6 The Maintenance Interface

The Maintenance interface (see Fig. 16) is to the TE, what the Administration is to a CEWebS. It has the role of:

- Initialising the TE.
- Changing keys and forcing a deletion of the cache.
- Get or set the users of the TE.
- Set the CEWebS contained in the TE.
- Set configuration parts.

Setting users and configuration parts works identical to the methods introduced in the previous chapter. A difference is, that for the TE having no instances, the according parameter is not necessary. An important thing to be mentioned is that when embedding the TE into an existing platform, the TE can be used in two ways:


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
R+M+J
Reports Courses Services Patterns

Here you can manage your course, i.e. add or remove services from the course. The name is the unique identifier of the service in the context of the course, and can be used in CEWebS:// links. You can also modify the order.

Projektmanagement, Soft Skills

Manage Users
Add a CEWebS

Info	(information)	[delete]	[visible]	[usable]
Teilnehmer	(participants)	[delete]	[visible]	[usable]
Themen	(themes)	[delete]	[visible]	[usable]
Reaktionen	(reactions)	[delete]	[visible]	[usable]
PM-SS Tagebuch	(diary)	[delete]	[visible]	[usable]
Fragebogen	(evaluations)	[delete]	[visible]	[usable]

© Business Information Systems, Webmaster [prepare] [do] [cache] [structure] (145ms)

Done

Adblock

Figure 15: The Service Manager - The Services in a Course

- If the platform that holds the TE already implements a sophisticated user management and already holds the users, the TE can be used as source for a user list. The AMan does not need to set any users, it only needs to extract them via **getUserList** and then distribute them to the CEWebS via *setUserList*. The implementation of the TE requires a connection to the platforms user repository and also has to implement a callback to the NoMan to signalise when user-data has changed.

- The platform that holds the TE has to have the ability to merge the users that are passed by the AMan with an existing set of users.

In both cases the TE has to have a very close connection to the embedding platform. In our test implementation the user data is held in simple XML files:

```
<?xml version='1.0' encoding='ISO-8859-1'?>
<persons>
  <person name='Motschnig' id='mo' email='a@x.y' pass='a'/>
  <person name='Derntl'    id='de' email='b@x.y' pass='b'/>
  <person name='Mangler'   id='ma' email='c@x.y' pass='c'/>
  <person name='Bauer'     id='ba' email='d@x.y' pass='d'/>
</persons>
```

Setting configuration parts is exactly the same as for a CEWebS. In our test implementation *getConfigurationParts* returns:

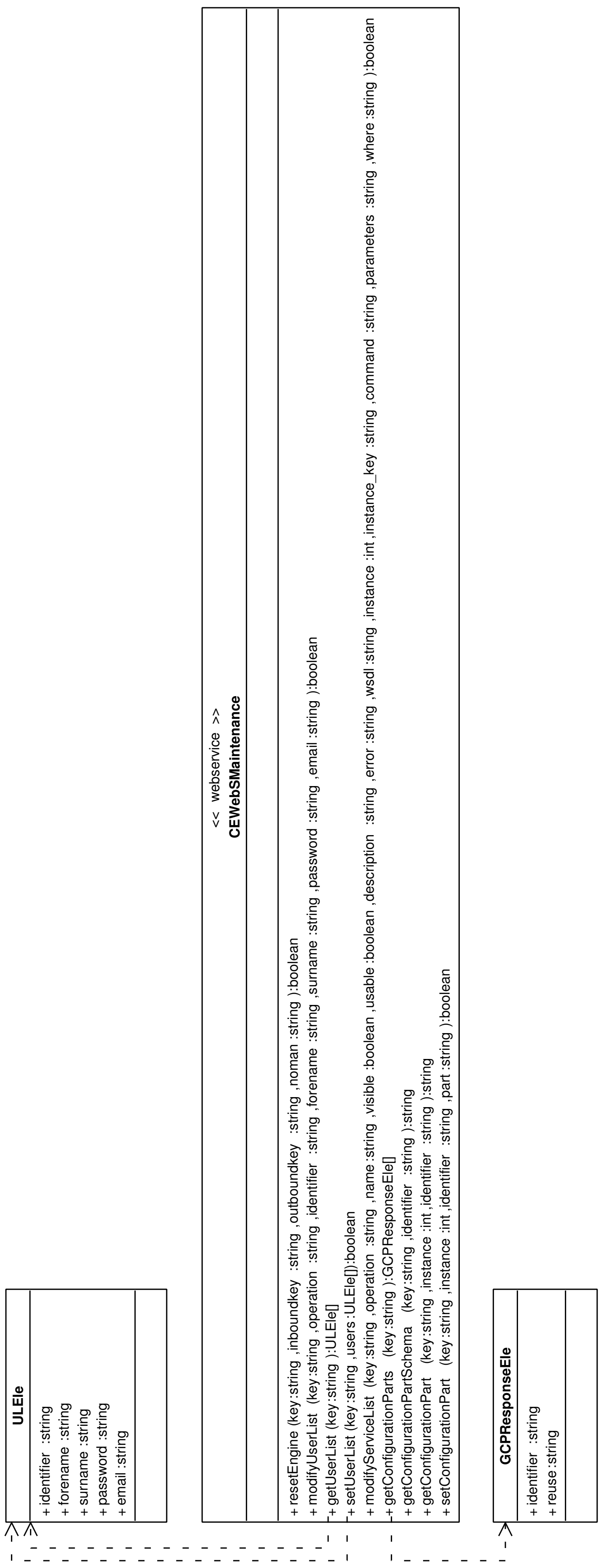

ULEle
+ identifier :string
+ forename :string
+ surname :string
+ password :string
+ email :string
<< webservice >>
CEWebSMaintenance
+ resetEngine (key:string ,inboundkey :string ,outboundkey :string ,noman :string):boolean
+ modifyUserList (key:string ,operation :string ,identifier :string ,forename :string ,surname :string ,password :string ,email :string):boolean
+ getUserList (key:string):ULEle[]
+ setUserList (key:string ,users :ULEle[]):boolean
+ modifyServiceList (key:string ,operation :string ,name:string ,visible :boolean ,usable :boolean ,description :string ,error :string ,wsdl :string ,instance :int ,instance_key :string ,command :string ,parameters :string ,where :string):boolean
+ getConfigurationParts (key:string):GCPResponseEle[]
+ getConfigurationPartSchema (key:string ,identifier :string):string
+ getConfigurationPart (key:string ,instance :int ,identifier :string):string
+ setConfigurationPart (key:string ,instance :int ,identifier :string ,part :string):boolean
GCPResponseEle
+ identifier :string
+ reuse:string

Figure 16: The Maintenance Interface

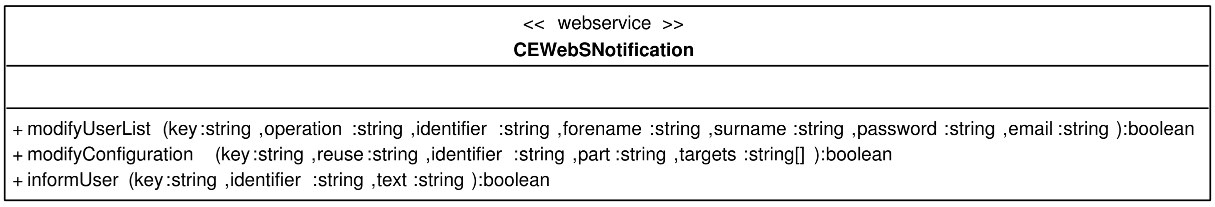

<< webservice >>
CEWebSNotification
+ modifyUserList (key :string ,operation :string ,identifier :string ,forename :string ,surname :string ,password :string ,email :string):boolean
+ modifyConfiguration (key :string ,reuse :string ,identifier :string ,part :string ,targets :string[]):boolean
+ informUser (key :string ,identifier :string ,text :string):boolean

Figure 17: The Notification Interface

```
[:identifier => 'description', :reuse => ' '],
[:identifier => 'name',        :reuse => ' '],
[:identifier => 'contact',     :reuse => ' ']
```

The Schema returns a structure that allows for the setting of one string for each of the parts, which results in three HTML files that are included at runtime into the final page, as shown before.

More interesting is the function that is used to initialise the TE. **resetEngine** takes **key** as parameter, except when it is called for the first time. **inboundkey** serves as verifier for each call from the AMan, **outboundkey** has to be supplied with every call to the NoMan. In the current implementation it is used:

- When called for the first time, to initialise the TE, in other words to create all necessary files and cache directories, as outlined in a previous chapter.
- For every subsequent call to clean the cache and reset the keys. As the TE is by nature more endangered by security breaks, the keys should possibly be changed on a regular basis.

modifyServiceList is the last method that is included in the interface. It is used to set all aspects of the services included in the TE. Like *modifyUserList* it has a parameter **operation** that can be set to **ADD_OR_CHANGE** or **DELETE**. The method changes *services.xml* that was described in more detail before.

4.7 The Notification Interface and the Notification Manager (NoMan)

The **Notification** interface (see Fig. 17) enables multiple CEWebS to exchange data. As explained in previous chapters, the AMan sets the WSDL location of a *Notification Manager* (NoMan) for each CEWebS and the TE so that they can ...

- exchange information about users that are allowed to use the TE and therefore also the CEWebS inside.
- hand over important messages for users, so that they can be delivered in an aggregated manner.

- exchange configuration parts, or, in other words, pieces of data that are considered useful by more than one CEWebS.

The NoMan is an implementation of the *Notification* interface, without a user interface (UI). It shares all information about the available TEs (and CEWebS they include) with the AMan. With this information it can distribute the messages to the CEWebS inside a particular TE. All methods in the interface use the parameter **key** to identify a service and check to which TE it belongs. So *key* has to be unique for all instances of the CEWebS inside an AMan (not only for a TE) . The key is supplied to the CEWebS and the TE as the **outboundkey**.

The method **modifyUserList** is basically the same as the *modifyUserList* from the *Administration* and *Maintenance* interfaces. It has the same basic parameters, that identify a user: **identifier**, **forename**, **surname**, **password** and **email** as well as the **operation**-string that tells if a user should be added/changed or deleted. In fact this method is translated to a *modifyUserList* call for all CEWebS and the TE.

The method **informUser** is very simple. Apart from the **key** to identify the CEWebS (and hence the TE) it has the parameters **identifier** to identify a particular user and **text** to deliver a message to this user.

The most important and interesting method is **modifyConfiguration** as it provides the ability to exchange configuration parts. The name "configuration parts" is maybe not very clear here, as it implies that the information has to be provided to the CEWebS in order to work properly. But there are always two sides: one CEWebS can collect or generate information (e.g. when the user provides a name for a project he / she has to elaborate during a course) that acts as configuration for a second service (e.g. after the project name is present, the user can upload files). So there has to be the ability to define configuration parts as *user-definable* or *passive* (when they are set by the service during work). These passive configuration parts are the ones that are also subject to exchange through the NoMan. The *modifyConfiguration* interface has (apart from **key**) the following parameters:

reuse: A string that defines what CEWebS belong together or, in other words, form a group that is supposed to exchange configuration parts.

identifier: The identifier of the part as described in a previous chapter (the names can be obtained through a **getConfigurationParts** call to the *Administration* interface).

part: A piece of XML that represents the actual data.

targets: Provides the ability to send a message only to a particular CEWebS. Because the CEWebS do not know what other services are in the TE, a target (or list of targets) has to be supplied to a CEWebS during configuration. The administrator can explicitly connect two or more services to get data from each other. This is useful for example when there are two services of the same type in a TE, that trigger something for only a subset of the CEWebS (E.g. two team building facilities that supply team information to two different team collaboration facilities).

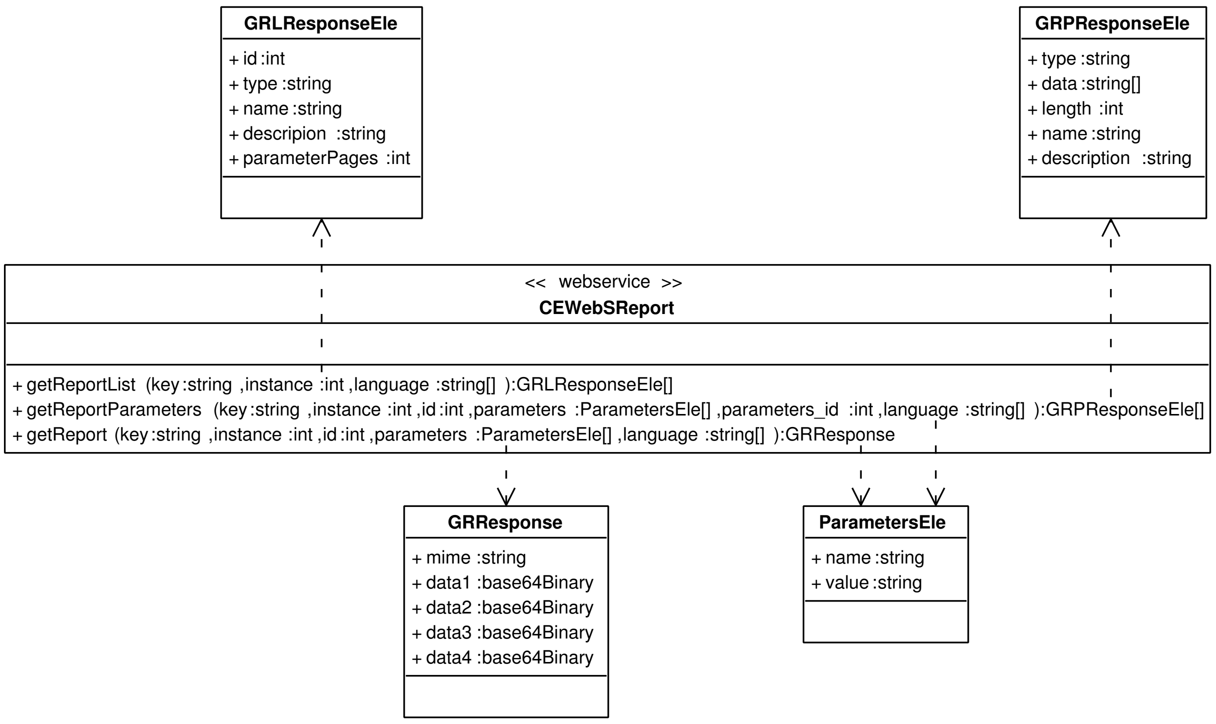


Figure 18: The Report Interface

I want to provide an example to make it more clear: In our case we have a set of CEWebS, that are designed especially for the task of supporting blended learning. So they all belong to the group (share the reuse string) **PCeL** which stands for **Person-Centered e-Learning** [23]. When a message is sent from a service, the NoMan calls **getConfigurationParts** for each CEWebS that is part of the same TE. When they report a part with a name that is identical to that provided by the sender and also have the same reuse string for it, this configuration part is set through **setConfigurationPart**.

This interface combines the ability to exchange data with the advantage of the whole system staying transparent. No CEWebS needs to know the location of any other CEWebS, and messages can be buffered to guarantee that the whole system is always up-to-date.

4.8 The Report Interface

The **Report** interface (see Fig. 18) is the only optional interface. It is maybe not useful for every CEWebS to implement it. The tool that calls the *Report* interface of a service is the **Report Manager** (ReMan). Together with the TE the ReMan is the most frequently used part of the architecture:

- The TE allows the students and the facilitator to interact. The data, that is collected during that interaction, is distributed over several CEWebS, that are responsible for only one aspect of this interaction.
- The ReMan allows the facilitator to look at the progress of the whole group or individual

students, by providing several views on the collected data.

For the *Report* interface to return data, the ReMan has also to supply the **key** on every call to a method. This *key* is the *inboundkey* that has been supplied during the creation of an **instance** through the *Administration* interface. The ReMan shares all data about created instances with the AMan, so it can provide the *key* if it wants to query an *instance*. Every report is bound to the data of a particular instance. The third parameter that is necessary for retrieving a report is the **language**-array, as supplied by the user's browser. This can be used to return a report that is tailored to the user's language preferences.

The *Report* interface consists of three methods, that have to be used together to get a result.

getReportList returns all reports that the service supports for a particular instance. Each report is represented by an object with the following attributes:

- **id**: The (per instance) unique identifier of the report.
- **type**: So far there exist two types of reports: **binary** and **email**. *binary*-type reports just return a downloadable object like an HTML page, an image or a PDF document. Reports of type *email* support the facilitator by providing the ability to write email to a group of people. For example there could be a CEWebS that provides the ability to write online reactions after a course. This CEWebS could implement a report of type *email* that lets the facilitator write an email to only students, that forgot to write their reaction.
- **name**: A short literal that can be used as menuitem when navigating through reports.
- **description**: A detailed description of the particular report.
- **parameterPages**: A CEWebS can ask the user several questions, before presenting a result. This parameter is used to denote the exact number of questions, so that a wizard can easily be created.

getReportParameters is used to get data about additional questions the user has to answer regarding the requested report. An example for such a question could be: *"Details about which student do you want to see?"*. There are additional parameters that have to be provided to this method. **parameters** is an array of objects containing **name** / **value** pairs. These objects hold the result of previous questions. When the method is called for the first time an empty set is provided. The value of **parameters_id** denotes the number of the question that is requested. As explained before, there can only be a total number of *parameterPages* questions. The data returned from this method is used to create a wizard. The input from the user has to be collected and provided to subsequent calls (via the *parameters* object). For these subsequent calls the CEWebS can rely on previous answers and present different questions to the user. The method returns an array of objects, that can be interpreted to create a form (page of a wizard). The attributes of the returned objects can be interpreted as follows:

- **type** is necessary to determine the form of data input. It helps to generate a wizard, that can utilise javascript to create comfortable input pages. Values include:

  - ”’enumeration”’: A drop-down combo is used. **data** is an array that holds multiple *id=value* pairs which are used to fill the combo.
  - ”’string”’: The user can enter arbitrary data in a single line input box, the first entry of *data* holds a default value.
  - ”’date”’: the user is forced to input a valid date, the first entry of *data* holds a default date.
  - **mask**: a series of input fields is created, according to a mask. The first entry in the array *data* holds character data, that is interpreted as label or separator, the second entry holds a regular expressions, that is used to check the user input. The third entry is interpreted as default value, the fourth entry as default length of the input field. E.g. [”, ’\d{2,4}’, ’0’, 4, ’.’, ’\d\d’ , ’42’, 2] forces the user to input between two and 4 numbers in the first input field, that has a default value 0, and has space for 4 alphanumeric characters. The point is printed as normal text, followed by a second input field that demands two numbers as input, is 2 characters wide, and has a default value of 42.

- *length* holds the size of the input element for enumeration and string *type*s.
- *name* is used as the name for the input field. For *date* and *mask* types multiple *name=value* pairs are generated.
- *description* is used as label in the resulting input page.

getReport takes the final **parameters** array (after collecting the results from each call to *getReportParameters*) as parameter. The result is an object that can be interpreted as follows:

- **mime**: This attribute holds the mime type, if the report is of type *binary*. Example: *”text/html; charset=utf-8”*.
- **data1**: For reports of type *binary* it holds the *filename*, for reports of type *mail* it holds a list of *recipients*.
- **data2**: For reports of type *binary* it holds the *filedata*, for reports of type *mail* it holds the *subject*.
- **data3**: For reports of type *mail* it holds the proposed *message-text*.
- **data4**: is reserved for later future usage.

4.8.1 The Report Manager (ReMan)

As stated before the ReMan is an important part of the CEWebS architecture. It covers the use-cases in Fig. 19. The current implementation is one of the oldest working parts in CEWebS (working since 2 years). When the administrator enters the ReMan, he / she can see a list of the courses in the current term, similar to the first screen for the AMan. In fact the data for creating the list, comes from the AMan. After selecting a course, the user can see the list of services that is contained in the course (in the TE), as illustrated in Fig. 20.

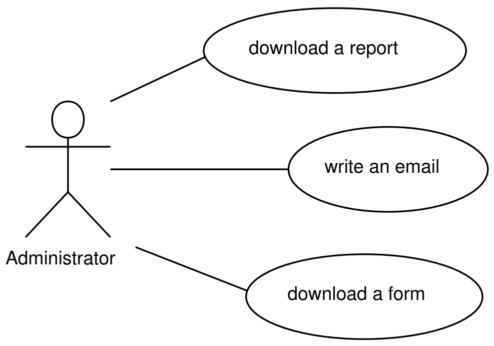
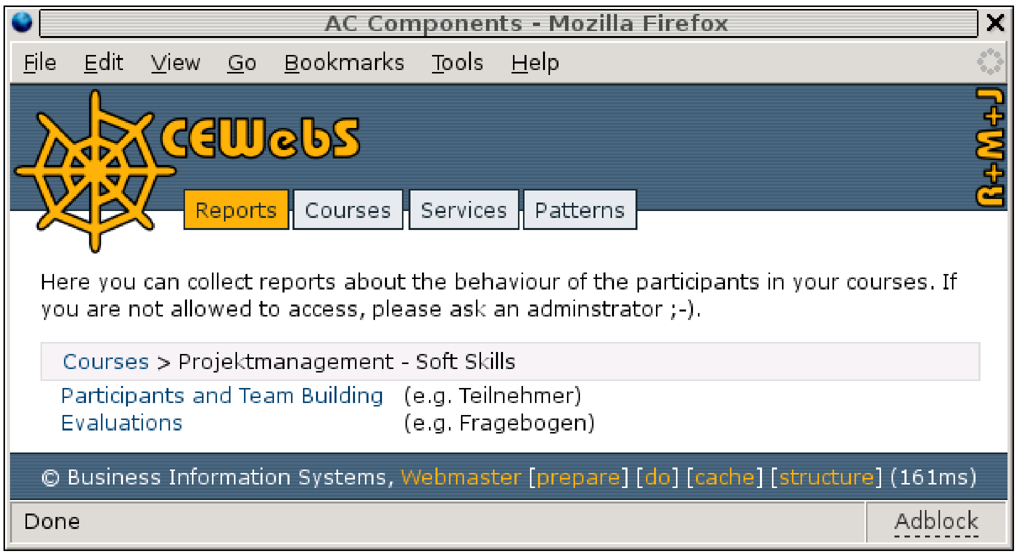


Figure 19: Use-cases For The Report Manager


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
R+M+J
Reports Courses Services Patterns

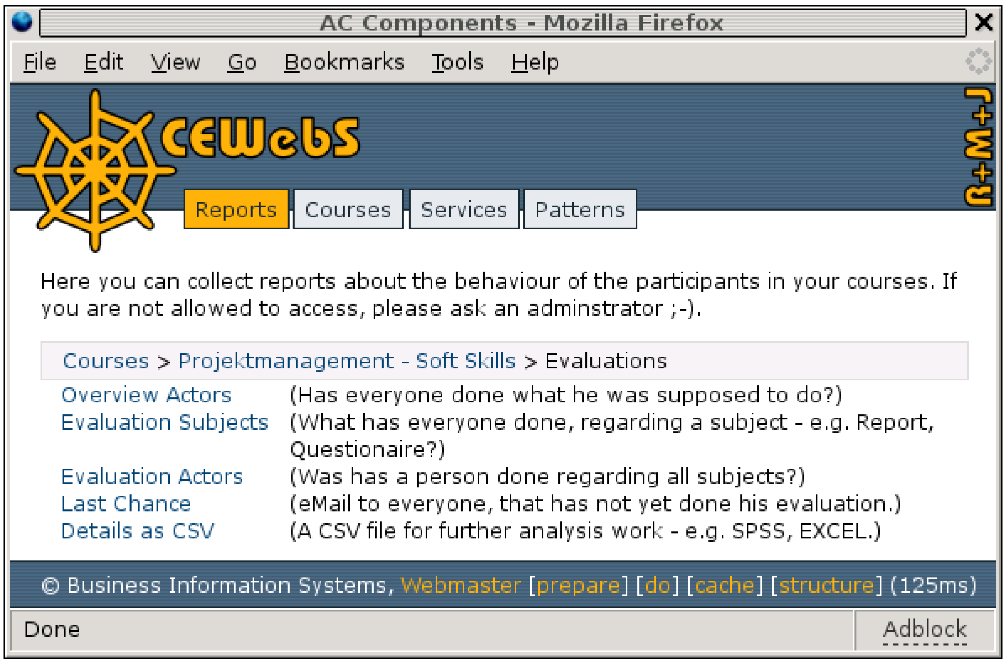
Here you can collect reports about the behaviour of the participants in your courses. If you are not allowed to access, please ask an adminstrator ;-).

Courses > Projektmanagement - Soft Skills

Participants and Team Building	(e.g. Teilnehmer)
Evaluations	(e.g. Fragebogen)

© Business Information Systems, Webmaster [prepare] [do] [cache] [structure] (161ms)

Figure 20: The Report Manager - A Course


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Reports Courses Services Patterns
R+M+J

Here you can collect reports about the behaviour of the participants in your courses. If you are not allowed to access, please ask an adminstrator ;-).

Courses > Projektmanagement - Soft Skills > Evaluations

Overview Actors	(Has everyone done what he was supposed to do?)
Evaluation Subjects	(What has everyone done, regarding a subject - e.g. Report, Questionaire?)
Evaluation Actors	(Was has a person done regarding all subjects?)
Last Chance	(eMail to everyone, that has not yet done his evaluation.)
Details as CSV	(A CSV file for further analysis work - e.g. SPSS, EXCEL.)

© Business Information Systems, Webmaster [prepare] [do] [cache] [structure] (125ms)

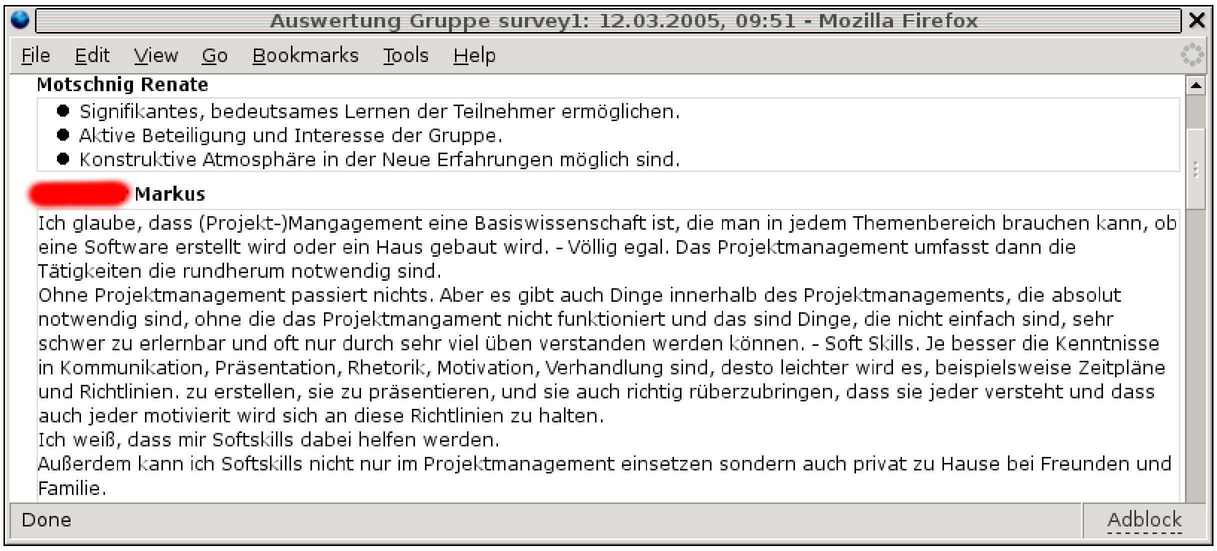
Done

Figure 21: The Report Manager - A Service

The list can be quite short, as not every CEWebS needs to provide a *Report* interface. The second point is, that a TE can hold several menuitems that point to the same CEWebS and CEWebS instance. E.g. the evaluation CEWebS seen in Fig. 20, is included two times in the same TE, pointing to a questionnaire at the beginning of the term, and to a different questionnaire for the end of the term, that are both contained in the same instance. These two occurrences of the same CEWebS and CEWebS instance, are aggregated to one occurrence in the report list, with the label of the menuitems put in brackets behind the name, so that the administrator can identify them.

Fig. 21 shows, what occurs, after the administrator selected the item *Evaluations*. He / she can see the list of reports that is available for the selected CEWebS. This list is obtained by a call to *getReportList*. *name* is selectable, and leads to the wizard, that is generated to ask the user for additional input, *description* is printed in brackets to clarify what the report returns.

In Fig. 22 one can see what is returned after finishing the wizard (that has been created through calls to *getReportParameters*). The return is a HTML document generated with the final call to *getReport*. In our example the aggregated text from the students reaction sheets for a particular course unit can be seen. It is now easy to print this report.


Auswertung Gruppe survey1: 12.03.2005, 09:51 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done
Adblock

Motschnig Renate

- Signifikantes, bedeutsames Lernen der Teilnehmer ermöglichen.
- Aktive Beteiligung und Interesse der Gruppe.
- Konstruktive Atmosphäre in der Neue Erfahrungen möglich sind.

[redacted] **Markus**

Ich glaube, dass (Projekt-)Mangagement eine Basiswissenschaft ist, die man in jedem Themenbereich brauchen kann, ob eine Software erstellt wird oder ein Haus gebaut wird. - Völlig egal. Das Projektmanagement umfasst dann die Tätigkeiten die rundherum notwendig sind.
Ohne Projektmanagement passiert nichts. Aber es gibt auch Dinge innerhalb des Projektmanagements, die absolut notwendig sind, ohne die das Projektmangament nicht funktioniert und das sind Dinge, die nicht einfach sind, sehr schwer zu erlernbar und oft nur durch sehr viel üben verstanden werden können. - Soft Skills. Je besser die Kenntnisse in Kommunikation, Präsentation, Rhetorik, Motivation, Verhandlung sind, desto leichter wird es, beispielsweise Zeitpläne und Richtlinien. zu erstellen, sie zu präsentieren, und sie auch richtig rüberzubringen, dass sie jeder versteht und dass auch jeder motivierit wird sich an diese Richtlinien zu halten.
Ich weiß, dass mir Softskills dabei helfen werden.
Außerdem kann ich Softskills nicht nur im Projektmanagement einsetzen sondern auch privat zu Hause bei Freunden und Familie.

Figure 22: The Report Manager - An HTML Report

5 Services

In this chapter I want to provide some insight into a selection of CEWebS we developed so far. I think they can be divided into two groups:

1. CEWebS that provide essential functionality that is already provided by other Learning Management Systems. We wrote them as proof of concept and to provide a basis for our courses. Then we fine-tuned them to fit better with the concepts of PCeL. To this group belong

   - **Scientiki**, a WIKI that has special ability to export to LaTeX and HTML and to support the creation of scientific papers. During development we had a strong focus on usability and to make the markup as self-explaining as possible.
   - **PersonContribution** and **TeamContribution** are used to collect students contributions. There has been a focus to design it in such way, that the contributors can't make mistakes. Providing overview about the contributors work so that they can learn from each other was another design goal.
   - **Evalution** is a CEWebS that deals with the structured input of information so it handles forms, questionnaires and surveys. Our approach is to make the data persistent for the user, whenever one enters a form / questionnaire for the second time the form is in exactly the same state as one left / saved it. Obviously this CEWebS has a strong focus on providing different reports.
   - **Discussion** provides the functionality to discuss topics. We put in special functionality to use WIKI markup, which includes the ability to easily publish pieces of source code. Improving usability compared to other forums was also a big issue.

2. CEWebS that we developed especially to support our courses, and enhance and ease interaction between the students and between the students and the facilitators. To CEWebS that belong to this category are:

- A **Diary** for single students or teams to easily keep track of their efforts.
- Facilities to build teams and form communities called **Participants** and **Community**.
- Various tools that focus Computer Science (CS) topics like *XML*, *Quality Assurance*, and *Modelling and Diagrams*.

Additional services are being designed and prototyped in advanced CS courses such as *"Web Engineering"*. In the following I will provide details and information about some of the so far successfully used services, about their adaption and mutation over time.

5.1 WIKIs

As reported above we were using *Dayta* for some semesters to support our courses. In *Dayta* (as well as in *WebCT*) it is possible to create files by using a built-in HTML editor. This approach has the following characteristics:

- Files have to be created in a specific location (e.g. a folder), one has to organise the files in order to avoid disorganisation
- Java has to be installed in order to run the editor
- A document can be created by means of WYSIWYG
- The resulting documents are HTML

Although this approach proofed to be very flexible I had a strong feeling that it had serious shortcomings:

- The plugin loaded slowly, the overall feeling when creating the document was sluggish
- The resulting HTML was not conforming to the XHTML specifications
- I had a feeling that the most elements for structuring text were useless for just writing simple text
- When creating text online one was stuck, one could either create one big HTML document and pasting it to a word document, or create a word document and paste it to the platform. When the document was distributed over multiple documents on the platform (which is very recommended web-usability-wise), this approach was not scaling very well.
- One could lose overview with multiple documents in multiple folders. The usual errors occurred, orphaned documents and stale links.
- When multiple people were working on the documents chaos reigned, documents were overwritten and deleted.

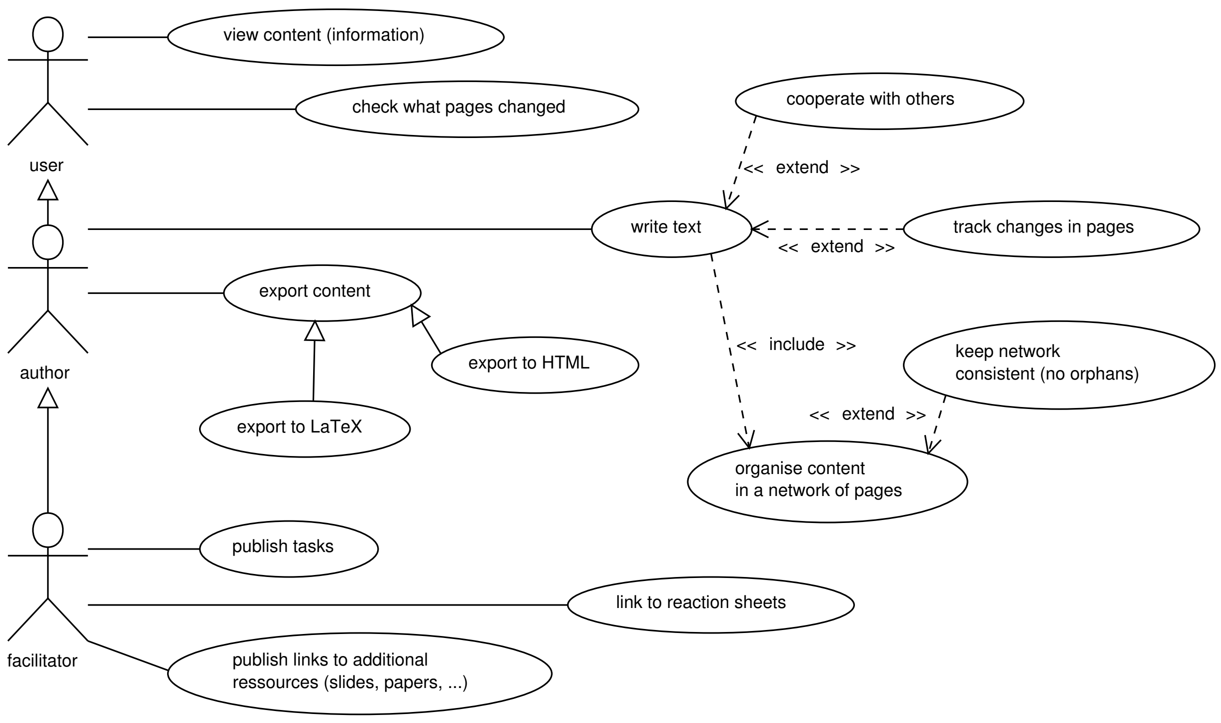


Figure 23: Use-Cases For a WIKI

So I decided to introduce a WIKI CEWebS that could solve the shortcomings. When thinking about the purpose of the CEWebS I identified the use-cases as illustrated in Fig. 23. I selected the name **Scientiki** for my WIKI to reflect a focus on scientific collaboration, for which I tried the WIKI to optimise. *Scientiki* is providing the following advantages over our previous approaches (some are WIKI related, some come from the implementation):

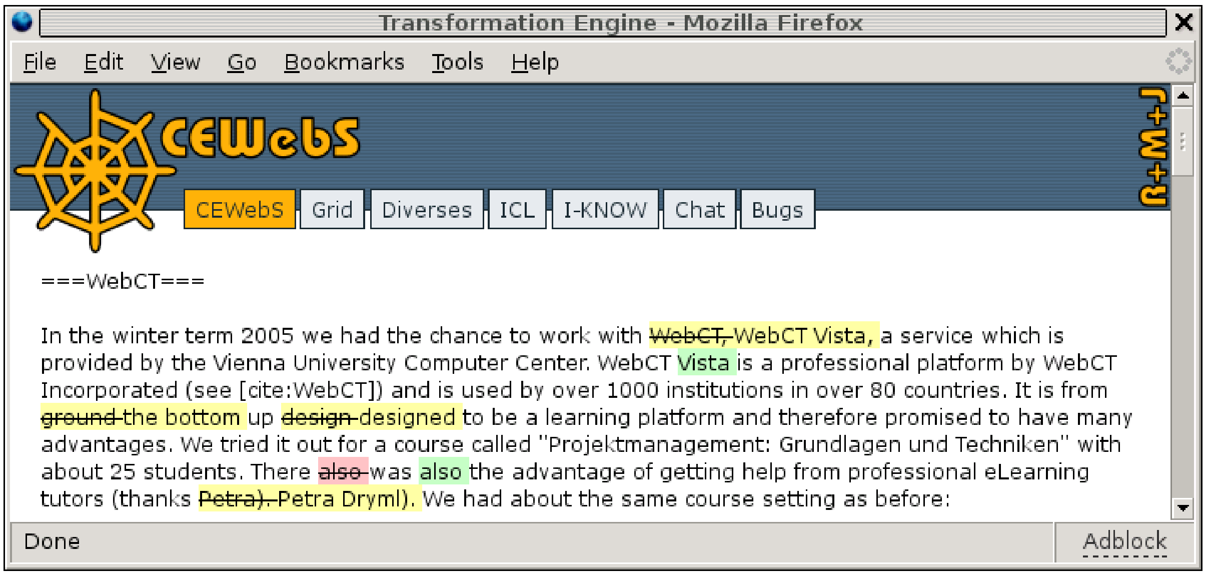
No plugin is required: The markup, although very simple was the element of uncertainty for me. I implemented the markup from the MediaWiki software [35] used by Wikipedia, stripping out all HTML-relics and changing to a more natural usage of newlines. I will describe the markup in more detail later.

Export: The markup is internally saved as it is entered, and two export formats are provided: HTML and LaTeX. Additionally it is possible to export a range of documents to one single document. So the advantages of both worlds are available: Simple navigation through small pieces of information and large printable documents.

Simple Markup: The markup is quite simple and not even covers the whole abilities of the above mentioned MediaWiki. The markup focuses on structuring the text, not on formating. The user decides that a text is a heading, he/she does not care about the style of the heading in the resulting page. I think the markup is suitable for most small to medium sized project, in fact it was suitable for writing this masters thesis.

A Document Network: By applying the WIKI principles one gets the advantage that all orphans are easily identifiable and no stale links are possible.

Focus on Cooperation: A WIKI focuses on cooperation. Many people work together and the WIKI assists this. The WIKI automatically resolves most conflicts when two people work


Transformation Engine - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
CEWebS Grid Diverses ICL I-KNOW Chat Bugs
Done
Adblock

===WebCT===

In the winter term 2005 we had the chance to work with ~~WebCT,~~ WebCT Vista, a service which is provided by the Vienna University Computer Center. WebCT Vista is a professional platform by WebCT Incorporated (see [cite:WebCT]) and is used by over 1000 institutions in over 80 countries. It is from ~~ground~~ the bottom up ~~design~~ designed to be a learning platform and therefore promised to have many advantages. We tried it out for a course called "Projektmanagement: Grundlagen und Techniken" with about 25 students. There ~~also~~ was also the advantage of getting help from professional eLearning tutors (thanks ~~Petra).~~ Petra Dryml). We had about the same course setting as before:

Figure 24: Diff Algorithm At Work

at the same document or warns the users when something is not resolvable. Scientiki is fully versioned by utilising Subversion [2]. By using an additional per word DIFF algorithm it is possible to provide the user with a very nice version tracking experience as shown in Fig. 24 (yellow text means replaced, red text means deleted, green text means added).

The markup as mentioned above differs slightly from MediaWiki. Making links to other location on the web works by inserting **"[http://www.ruby-lang.org|description]"**, mailto: links work by inserting **"[mailto:et@cewebs.org|description]"**. Creating new WIKI-keywords (new documents in the WIKI) is as simple as writing **"[[keyword]]"**. The resulting link is red when the linked document does not already exists. It turns to blue when content is added.

Headings work the same way as in MediaWiki: **"==heading=="**, **"===sub-heading==="**, **"====sub-sub-heading===="**. Other basics include:

/*WIKI syntax expression*/: A comment. The expression inside is not processed by the WIKI markup parser.

"emphasis": Two single quotes. Tells the parser to print the content in *italic letters*.

"'strong"': Three single quotes. Tells the parser to print the content in **bold letters**.

!!annotation!!: This element is excluded from export. With this element the users can put comments in the text, to further ease collaboration. This should not be used as versioning instrument.

Newlines are handled differently like they are handled in MediaWiki. Every newline has an impact on the result. To create a paragraph one can enter two newlines. More than two lines create a bigger space (although this space is not growing with more then three newlines).

Lists basically work like in MediaWiki with the exception that a newline is possible. A list is terminated when two newlines (a paragraph) occur.

*** a star:** A star at the beginning of a line starts a normal bullet-list. The list continues until a paragraph or a heading is found.

**** two stars:** Two stars at the beginning of a line start a sub-list. It can only exist inside a bullet-list or numbered list.

***** three stars:** Three stars at the beginning of a line start a sub-sub-list. It can only exist inside a bullet-sub-list or numbered sub-list.

# a hash: A hash symbol at the beginning of a line starts a numbered list. The list continues until a paragraph or a heading is found.

## two hashes: Two hash symbols at the beginning of a line start a numbered sub-list. It can only exist inside a bullet-list or numbered list.

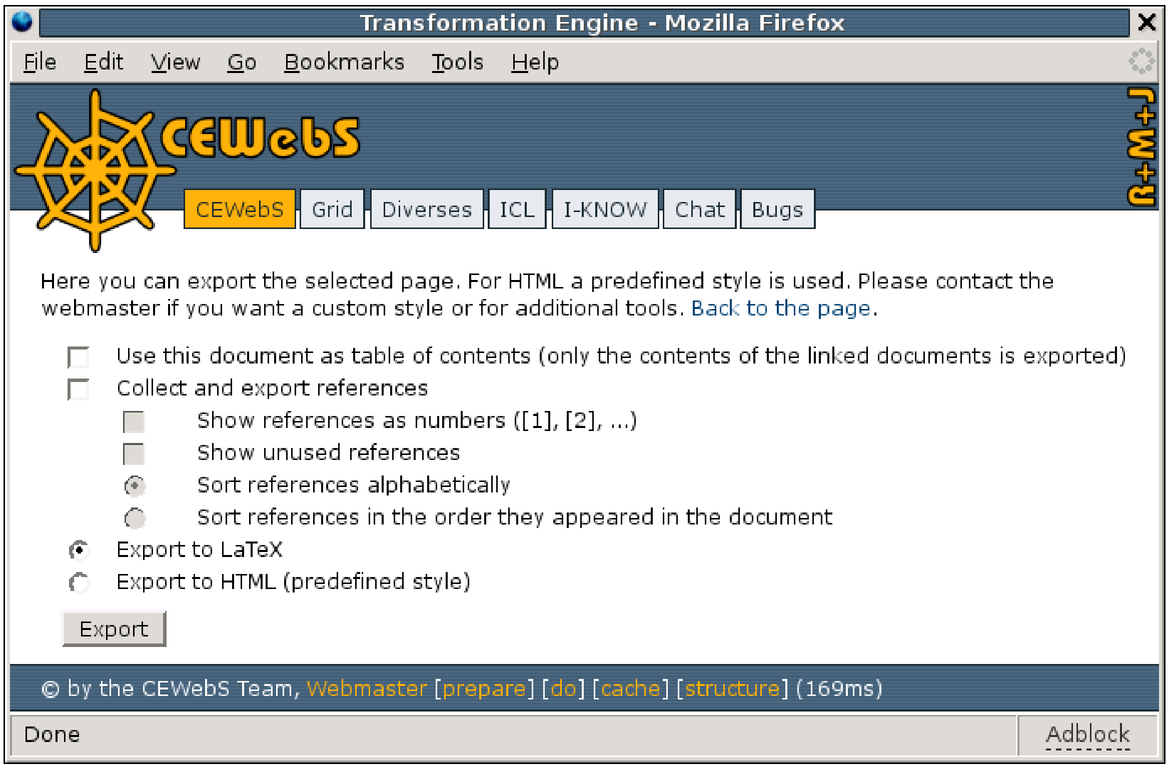
### three hashes: Three hash symbols at the beginning of a line start a numbered sub-sub-list. It can only exist inside a bullet-sub-list or numbered sub-list.

As already obvious these two list types can also be interspersed. The last list type is a definition-list. It is used by writing **";definition: description"** starting at the beginning of a line. This type comes in very handy when defining concepts or terms.

> A difference to MediaWiki markup is introduced by making it possible to create indented text blocks. This is especially useful when one has to add long quotes to his documents. An indented paragraph is denoted by two leading > characters (»).

A custom addition that to the best of my knowledge only exists in Scientiki is the literature references. They are inspired by the way LaTeX handles them. A reference can be placed on an arbitrary page in the WIKI, by using the **"* [ref:identifier|optional shortcut] reference text"** syntax (starting in a new line). This reference can than be cited everywhere in the WIKI by using **"[cite:identifier]"** or **"[cite:optional shortcut]"**. The export also assists this system by collecting the references from the whole WIKI and appending it to the exported document. So its wise to put all references to a central document that is not included in the export. When exporting the user can also select various options regarding the export of links (see Fig. 25).

Inserting images is modelled after adding attachments to an email. Images can be added to a document and then be used. Images from other documents are not (!) usable- Adding the images to the document is as easy as uploading them. When uploading a size restriction can also be applied. If an image exceeds this size, it is resized to meet the restriction. This is also useful for vector graphic images, as they often do not contain size information. By using ImageMagick [15] it is possible to throw over 90 major formats at the system. Images can be used inside the page by writing **"[image:Scientiki/name|description]"**. Every image on the page can then be referenced inside the text by using **"[fig:Scientiki/name]"**.


Transformation Engine - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
CEWebS Grid Diverses ICL I-KNOW Chat Bugs

Here you can export the selected page. For HTML a predefined style is used. Please contact the webmaster if you want a custom style or for additional tools. Back to the page.

- Use this document as table of contents (only the contents of the linked documents is exported)
- Collect and export references
  - Show references as numbers ([1], [2], ...)
  - Show unused references
  - [✓ Sort references alphabetically]
  - Sort references in the order they appeared in the document
- [✓ Export to LaTeX]
- Export to HTML (predefined style)

Export
© by the CEWebS Team, Webmaster [prepare] [do] [cache] [structure] (169ms)
Done
Adblock

Figure 25: Exporting Documents

Both, cites and references to images are clickable. For images the page scrolls to the image, for references one jumps to the page with the reference.

Tables also differ from MediaWiki. A table is created by starting a line with "|| ". At the end of each row the user has to put a "|-". Each row can span multiple lines in the input box. Each cell has to be separated by another " || ". By default the text is aligned left. This can be changed by using " |r ", " |c " or " |l " instead of " || ".

There is also the ability to add text that is printed in monospaced letters. This is especially useful when one wants to add source code to a document. Every line with a space as the first letter is printed monospaced, multiple successive lines following this rule are interpreted as block. Example:

```
people.each do |person|
  puts person + ' has an IQ of ' + rand(200)
end
```

Scientiki proofed to be a very powerful and useful instrument. It is used as blog to inform the students about changes, it is used to discuss issues and to write papers. Its flexibility will hopefully expand even further in the future, when we can include support for didactical models.

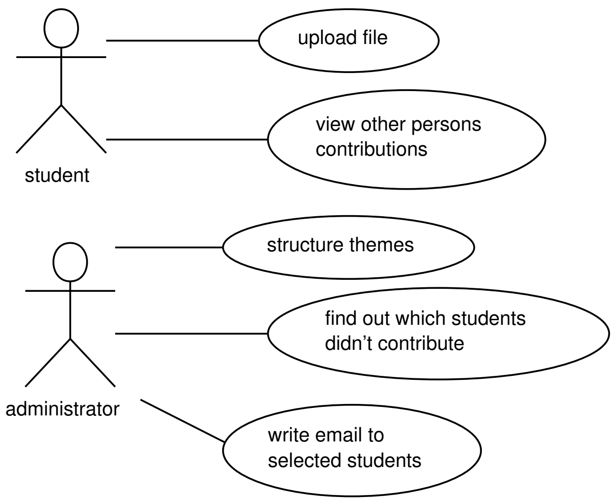

student
upload file
view other persons contributions
administrator
structure themes
find out which students didn't contribute
write email to selected students

Figure 26: Contributing / Uploading Material

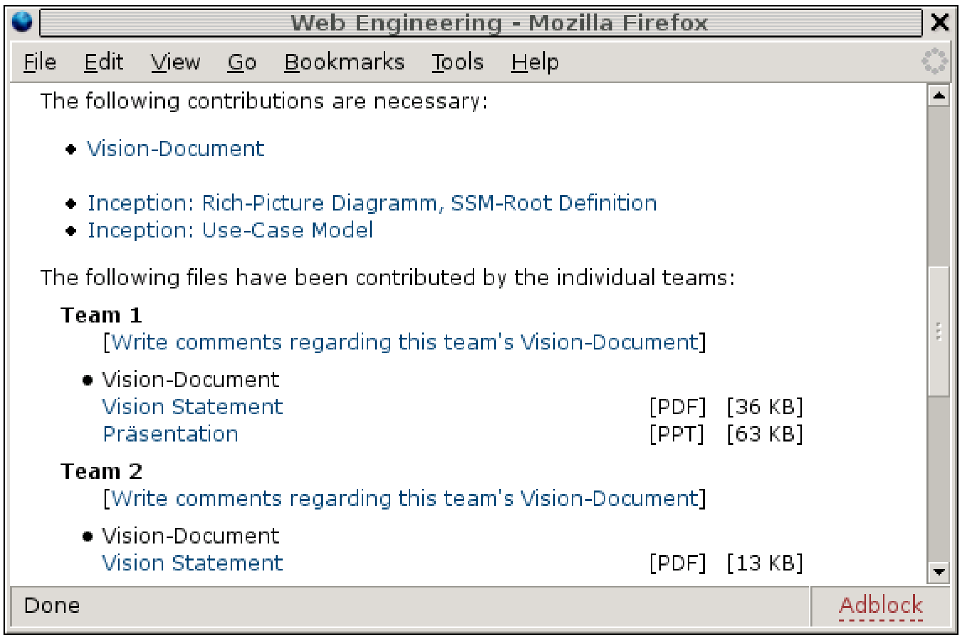
5.2 Team- and Person-Contributions

The idea to realise this CEWebS came from the necessity to collect material from students.

- Collecting material from students via email is complicated because the facilitator has to keep an overview over multiple versions and documents (maybe sent by multiple team members)
- In *Dayta* the problem for the facilitator was that he/she had to browse through directories to find the documents, which where not always easy to identify.
- In *WebCT* for the first time there was a central point where all the students could add their documents, but:
  - The process of posting and calling them back is in my eyes too complicated (as explained above)
  - Team members can not work together, only the member that posted the document can can call it back
  - Although the facilitator can request reports which provide a nice overview over the documents already submitted, the students can not request these. We found that by allowing the students to view the elaborations of others (given the situation that individual teams are working on their own projects) improves the quality and the overall satisfaction of the students.

So the decision was made to create two CEWebS that focus upon contributions by teams (**TeamContribution**) and single students (**PersonContribution**) respectively. We identified the use-cases shown in Fig. 26.

We will now take a look at the *TeamContribution* service. Under the precondition of being as simple as possible, we realised it consisting of only three pages for managing the whole process.


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Done
Adblock

The following contributions are necessary:

- Vision-Document
- Inception: Rich-Picture Diagramm, SSM-Root Definition
- Inception: Use-Case Model

The following files have been contributed by the individual teams:

Team 1
[Write comments regarding this team's Vision-Document]

- Vision-Document

Vision Statement	[PDF]	[36 KB]
Präsentation	[PPT]	[63 KB]

Team 2
[Write comments regarding this team's Vision-Document]

- Vision-Document

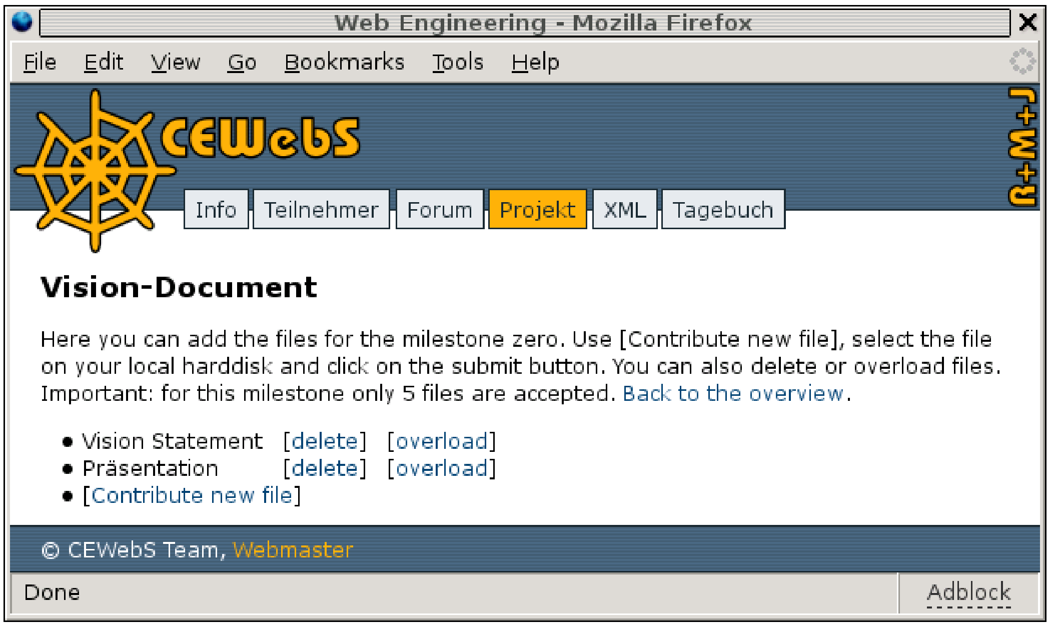
Vision Statement	[PDF]	[13 KB]

Figure 27: List of the Uploaded Material

The first page provides a detailed overview about what the individual teams contributed so far (see Fig. 27). In our case the details-page consists of various elements:

- At the top of the page a short introduction is given, explaining the project the students have to contribute to and also providing various links, where the students can find additional information.
- **”The following contributions are necessary:”** contains a list of all tasks to which the students have to contribute. This list is changing over the term, as new entries are added, and old ones are removed. Again the principle of showing only what is necessary is very important to us.
- The main section is lead by **”The following files have been contributed by the individual teams:”**. This section holds:
  - A scope for each team
  - Each teams scope is further divided into a space where links in brackets ([...]) occur, leading to other CEWebS providing peer-evaluation facilities. A parameter CEWebS_section is added to each link, to denote the team that is currently evaluated. In this space also the links to the results of the evaluation occur.
  - The second space holds the files for each task. When clicking on the filename, the file is opened or downloaded to the local hard-disk.

After selecting a task from the list, the user is presented with a page that allows for managing the files that a team contributed to the specific task so far (see Fig. 28). The management-page holds a more detailed description about a single task, and allows to add new files, delete files and overload (or rename) existing contributions.


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Info Teilnehmer Forum Projekt XML Tagebuch

Vision-Document

Here you can add the files for the milestone zero. Use [Contribute new file], select the file on your local harddisk and click on the submit button. You can also delete or overload files. Important: for this milestone only 5 files are accepted. Back to the overview.

- Vision Statement [delete] [overload]
- Präsentation [delete] [overload]
- [Contribute new file]

© CEWebS Team, Webmaster

Done Adblock

Figure 28: Manage Your Contributions

The last page that the user has to master is the upload-page where he / she has to select a file and add a description. After uploading a file, the updated management-page is presented to the user. The CEWebS also provides a report of type *email*, that can be used to write an email to all teams that have not yet contributed a file to a task.

The *PersonContribution* service is almost equal. The only difference is that instead of the teams, the names of the students are printed. An of course each user can only manage his/her own files.

5.3 *Evaluations*

Evaluation is a CEWebS that is specialised on structured input. In its current form it is suitable for *questionnaires*, *reaction-sheets*, *surveys* and in general, data from every template that can be generated by the service. We identified the use-cases that are shown in Fig. 29 for the *Evaluation* CEWebS. The most important thing was again usability. So from the user's point of view the service consists only of one page . In the normal mode (logged in users) the form is stateful. In other words, when the user saves his / her entries and then returns to the form at a later point in time, the form is in exactly in the same state as he / she left it. When saving, only the obvious *"Thank you for ..."* message is displayed. The form itself is used to inform the user what was filled in. With this behaviour there is only one page, that the user has to master.

I will use the XML that is used to create the form to further explain the different modes that this service can operate in:

```
<evaluations>
  <evaluation mode='person' subject='demo' description='Demo'
                                           details='anonymous'>
```

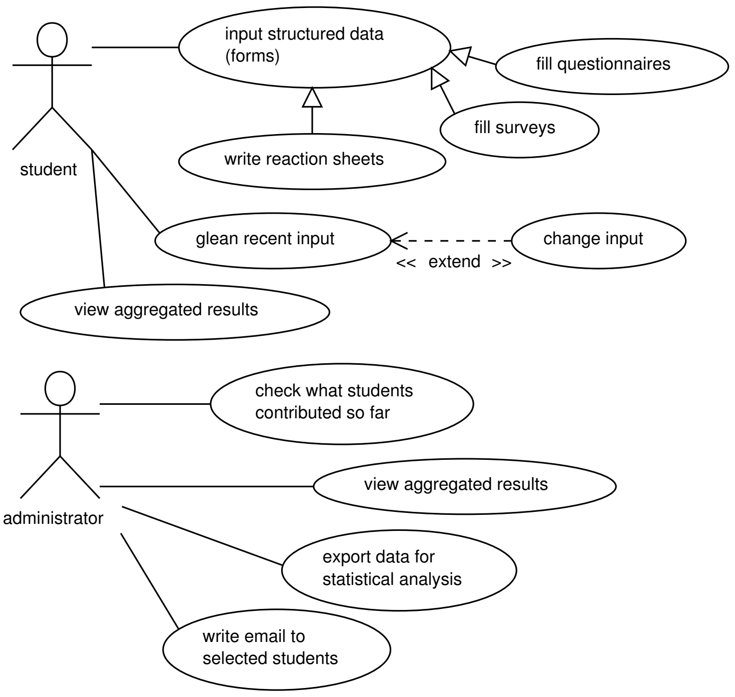


Figure 29: Collecting Forms-Input

```
    ...
  </evaluation>
</evaluations>
```

Each instance can hold multiple evaluations, each evaluation in an instance has to have a unique identifier **subject**. The attribute **description** is used in reports instead of the raw subject string, so that the user can easier identify the evaluation. The most important attribute is **mode**. It can have one of four values: **person**, **team**, **group**’ and **all**. These four values describe how the CEWebS behaves, when multiple users fill out the same form.

- In *person* mode every user has its own form (showing always the own entries).
- In *team* mode, when a team member enters a form he / she can see the input that other team members have saved so far. This allows for fine grained collaboration.
- *group* mode is essentially the same, but for larger groups of people or groups of teams.
- In *all* mode, all users that are authorised for the instance share the form and the already saved data.

The other aspect is aggregation of the input. An aggregated version of all users input can be presented (e.g. results of a peer-evaluation). The modes are also essential for the aggregation. Instead of an input box, there is a list of answers for every question. Depending on the mode these answers come from single persons, teams, ... (and are labelled as such).

The attribute **details** controls if the name / team synonym is printed or not. When *details* has the value **anonymous**, no information about the contributing actor is shown in the aggregated results.

Every evaluation can consist of multiple language (**lang**) groups, that hold the data. For every language there must exist an **intro**, and one or more **block** elements. Every block element has to have at least one element annotation:

```
<lang id='en'>
   <intro>Demonstration of the possibilities.</intro>
   <block type='....'>
     <annotation>.......</annotation>
   </block>
 </lang>
```

The block elements holds the information about the form elements. There exist only eight types of blocks, that have been proved to be sufficient to create almost every form of questionnaire. This sounds in fact over-simplified, particularly when one takes into account that in fact there exist only four usable elements (as shown below). All elements are high-level components. That means they are not input-fields, check-boxes, ..., but compound structures with a description, multiple elements and optional auto-generated javascript attached. In the following I will explain all different block types and how they are used.

The first two elements are used to structure the resulting form. They provide the ability to insert headings and additional explanations (aka. labels). **annotation** is used as heading- or label-text.

```
<block type='heading'>
  <annotation>Heading</annotation>
</block>
<block type='label'>
  <annotation>Label</annotation>
</block>
```

All following elements are able to collect input. The three *textarea* elements only differ in height. *largetextarea* is realised to be about twelve lines high, *textarea* about five lines, and *simpletextarea* exactly one line. *annotation* is used as label. The attribute **use** affects the javascript generation. If use is *'required* then the user is forced to provide some data in the input field.

```
<block type='largetextarea' use='required'>
  <annotation>Large textarea</annotation>
</block>
<block type='textarea' use='required'>
  <annotation>Normal textarea</annotation>
</block>
<block type='simpletextarea'>
  <annotation>Small textarea</annotation>
</block>
```

graduation proved to be the most often used element. It provides the ability to group together multiple questions with an equal set of answers. It is in my eyes a method for making a questionnaire very compact and concise. It consists of questions and a set of radio-buttons, one for each answer. The XML looks as follows:

```
<block type='graduation' use='required'>
  <annotation>A graduation</annotation>
  <group>Scale</group>
  <selection>0</selection>
  <selection>1</selection>
  <selection span='4'>2 to 5</selection>
  <question type='text'>Do you like CEWebS?</question>
  <question type='entry'>Own contribution</question>
</block>
```

The element has the following characteristics:

- <**group**> can hold text that is printed above the question, just besides the scale. It can be used for example to describe the scale.
- The scale is specified by multiple <**selection**> elements. They can also hold a **span** attribute that makes the scale span multiple radio buttons.
- the questions itself are contained in <**question**> elements. They can be of type **text** or **entry**. In the case of *text* elements, the content is printed as label, when an *entry* element occurs, the content is used as default value for a textbox which can be changed by the user.

listCheck and **listRadio** belong to the same family. The difference to *graduation* is, that there is no scale, instead the questions are selectable items. For *listRadio*, one *question* can be selected, for *listCheck* the user can select (or check) any number of questions.

```
<block type='listCheck' use='required'>
  <annotation>A checkable list</annotation>
  <group>Detailed description</group>
  <question type='text'>Option 1</question>
  <question type='text'>Option 2</question>
  <question type='entry'>Own option</question>
</block>
<block type='listRadio' use='required'>
  <annotation>Selectable options</annotation>
  <group>Detailed description</group>
  <question type='text'>Option 1</question>
  <question type='text'>Option 2</question>
  <question type='entry'>Own option</question>
</block>
```

In Fig. 30 one can see a sample output, generated from the XML pieces above. The *larg-textarea* and *textarea* elements are bigger in real life, but have been reduced in height to make the screenshot smaller.

For this service the reports are (from a facilitators point of view) more interesting than the user-visible parts. The reports allow to have different aggregated views on the collected data. To help a facilitator to overview the activities of the students we created the following reports:

Overview Actors: The facilitator has to select one of the evaluations (identified by *subject*), to see a list of all actors (persons, teams, groups or all) and what they evaluated. The list of the things a student evaluated seems quite useless, because the facilitator already selected the subject. But there is also the concept of sections (as explained in the *Contributions* chapter). For each subject there can be an arbitrary number of these sections. For example when the student has to peer-evaluate other students' contributions, the subject is *"peer"* but the things the student evaluated are *"Team 1"*, *"Team 2"*, *"Team 3"* and so on.

Evaluation Subjects: This is interesting if the facilitator wants to see all entries that have been entered for a particular subject. He / she can again select the *subject*. In the next step the facilitator has to select for what section he / she wants to see results (**All** is also possible). The last decision he / she has to make is, if the names of the students / teams should be added to the report. This is e.g. useful when the report is sent to partners that should not see names.

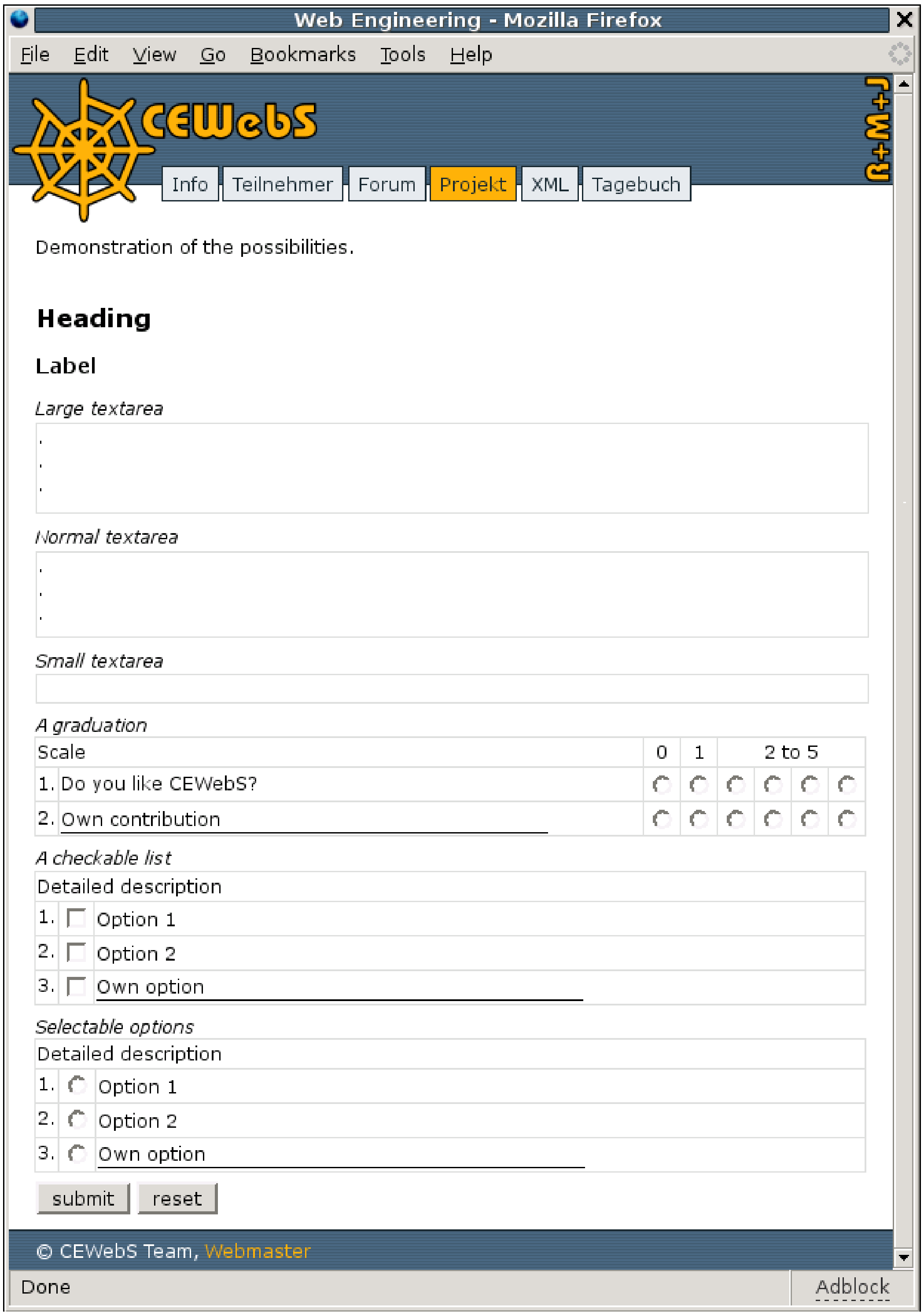
Evaluation Actors: This report is useful when the facilitator wants to see all entries a student made. For our peer-review example the report would consist of the names of the teams a student reviewed and the data he / she entered. After selecting a *subject*, the facilitator can decide for what actor (person, team, group, all) he / she wants to get the report.

Last Chance: This report is of type *"email"*. It is tied to the *"Overview Actors"* report in that way, that the *list of things* is counted and and a facilitator can select the email addresses for all students that evaluated less than *X* things. E.g. for a reaction-sheet (where no sections are used) a good value for *X* is one, because the facilitator wants to select all people that have not yet written a reaction. For a peer-evaluation where at least four other contributions have to be evaluated, *X* has to be four. The resulting mail-form looks like shown in Fig. 31.

Details as CSV: Useful for exporting the data to a statistical analysis package. For a given subject and section all data is returned as CSV file. The file holds one line for every actors, starting with the name or description of the actor (a person's name or team description), a section name (if sections where used), followed by all the input the actor has made, separated by semicolons.

5.4 Discussion Forums

Designing a forum that satisfies all students proved to be somehow difficult, as people tend to like what they already use. The students turned out to be very critical about the forum we


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Info Teilnehmer Forum Projekt XML Tagebuch
R+M+J

Demonstration of the possibilities.

Heading

Label

Large textarea

Normal textarea

Small textarea

A graduation

Scale		0	1	2 to 5			
1.	Do you like CEWebS?	○	○	○	○	○	○
2.	Own contribution	○	○	○	○	○	○

A checkable list

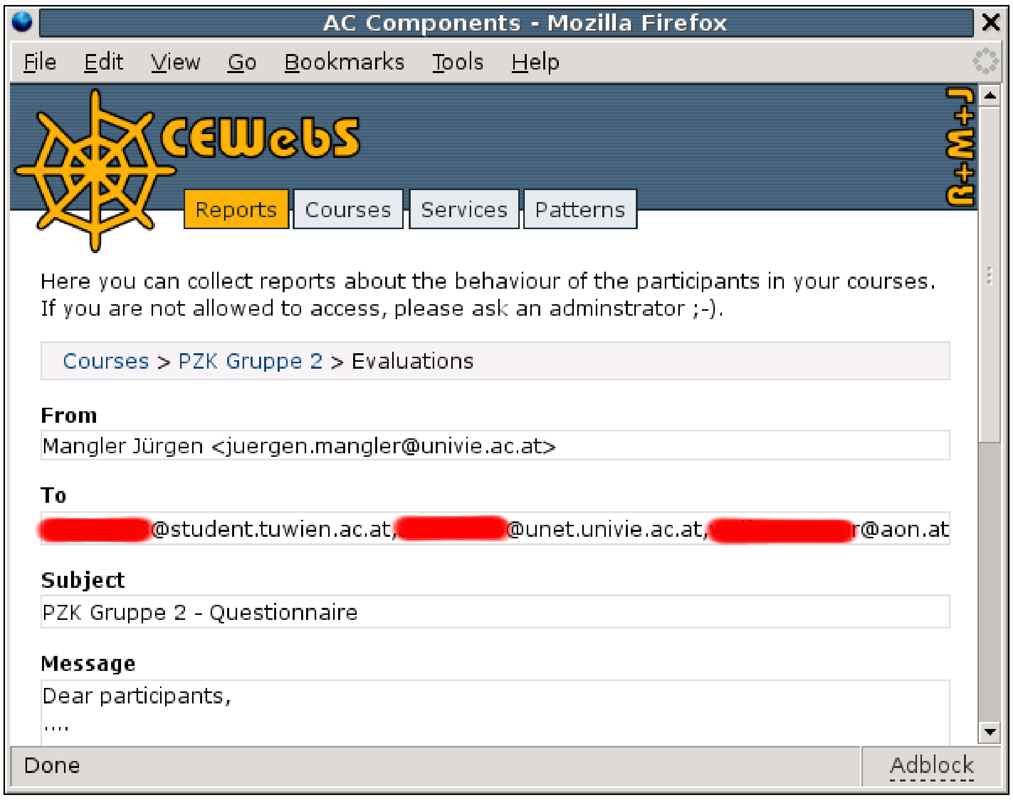
Detailed description		
1.	☐	Option 1
2.	☐	Option 2
3.	☐	Own option

Selectable options

Detailed description		
1.	○	Option 1
2.	○	Option 2
3.	○	Own option

submit
reset
© CEWebS Team, Webmaster
Done
Adblock

Figure 30: An Example Form Showing All Possibilities


AC Components - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Reports Courses Services Patterns
Done
Adblock

Here you can collect reports about the behaviour of the participants in your courses.
If you are not allowed to access, please ask an adminstrator ;-).

Courses > PZK Gruppe 2 > Evaluations

From
Mangler Jürgen <juergen.mangler@univie.ac.at>

To
[redacted]@student.tuwien.ac.at,[redacted]@unet.univie.ac.at,[redacted]r@aon.at

Subject
PZK Gruppe 2 - Questionnaire

Message
Dear participants,
....

Figure 31: An Form For Contacting Students

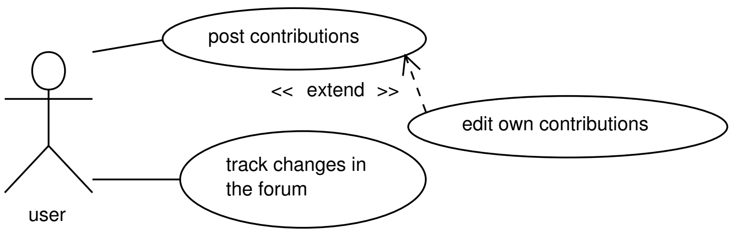
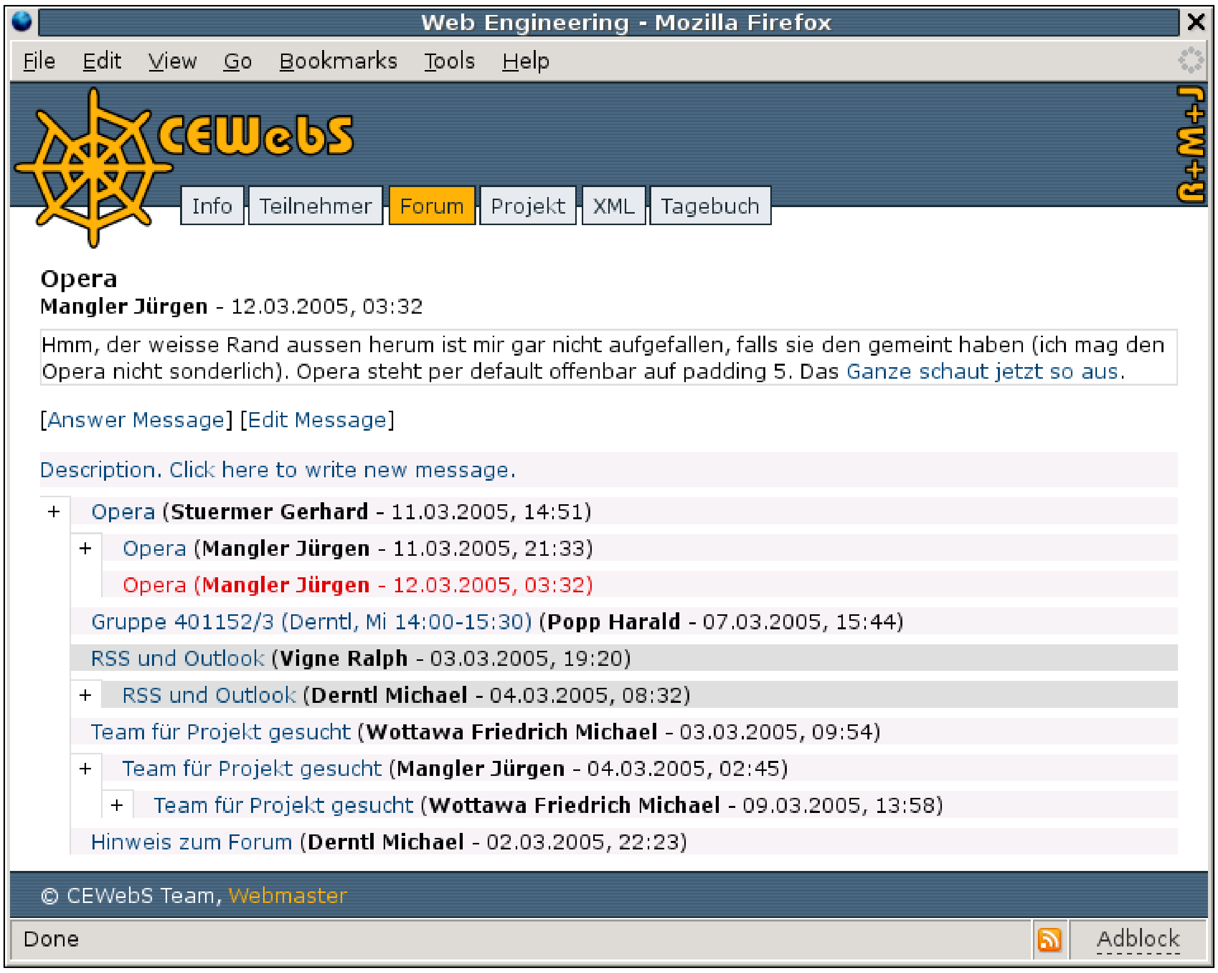


Figure 32: Discussing Topics


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Info Teilnehmer Forum Projekt XML Tagebuch

Opera
Mangler Jürgen - 12.03.2005, 03:32

Hmm, der weisse Rand aussen herum ist mir gar nicht aufgefallen, falls sie den gemeint haben (ich mag den Opera nicht sonderlich). Opera steht per default offenbar auf padding 5. Das Ganze schaut jetzt so aus.

[Answer Message] [Edit Message]

Description. Click here to write new message.

```
+ Opera (Stuermer Gerhard - 11.03.2005, 14:51)
   + Opera (Mangler Jürgen - 11.03.2005, 21:33)
     Opera (Mangler Jürgen - 12.03.2005, 03:32)
  Gruppe 401152/3 (Derntl, Mi 14:00-15:30) (Popp Harald - 07.03.2005, 15:44)
  RSS und Outlook (Vigne Ralph - 03.03.2005, 19:20)
   + RSS und Outlook (Derntl Michael - 04.03.2005, 08:32)
  Team für Projekt gesucht (Wottawa Friedrich Michael - 03.03.2005, 09:54)
   + Team für Projekt gesucht (Mangler Jürgen - 04.03.2005, 02:45)
      + Team für Projekt gesucht (Wottawa Friedrich Michael - 09.03.2005, 13:58)
  Hinweis zum Forum (Derntl Michael - 02.03.2005, 22:23)
```

© CEWebS Team, Webmaster

Done Adblock

Figure 33: A Typical Discussion

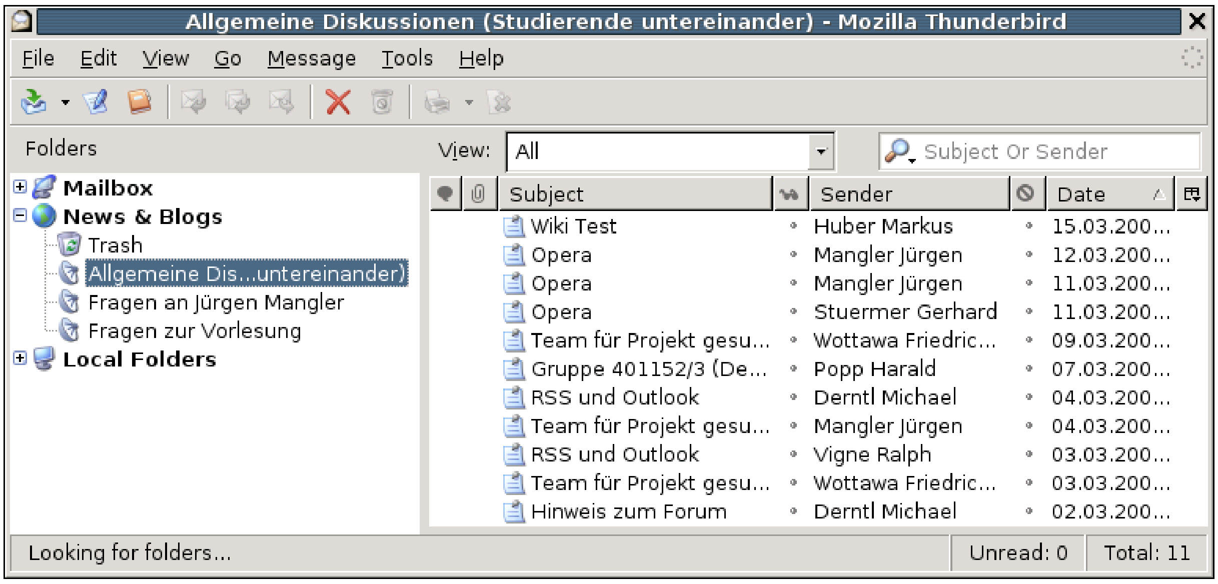
introduced. The use-cases for a forum are quite simple (as can be seen in Fig. 32), but to implement one can get quite complicated. We identified three major systems that forums use to visualise a discussion:

Threaded: The user sees a tree representation and can click on the subjects to see the messages.

List: All messages are show in a list, the user can jump to following or preceding messages.

Threaded with messages included: The tree also holds the messages. This can be combined with a rating system to show not all messages or else the page holding the forum will be very long.

All systems have their advantages and are used on big websites with many users. My personal preference is a clickable threaded view as I think it is the most concise form. But the students were surprisingly favouring the simple list form (which I in fact hate and refuse to implement). I think their preference comes from a forum they use every day, and that is focused on discussing several aspects of our branch of study.



Allgemeine Diskussionen (Studierende untereinander) - Mozilla Thunderbird

File Edit View Go Message Tools Help

Folders

- Mailbox
- News & Blogs
  - Trash
  - Allgemeine Dis...untereinander)
  - Fragen an Jürgen Mangler
  - Fragen zur Vorlesung
- Local Folders

View: All — Subject Or Sender

Subject	Sender	Date
Wiki Test	Huber Markus	15.03.200...
Opera	Mangler Jürgen	12.03.200...
Opera	Mangler Jürgen	11.03.200...
Opera	Stuermer Gerhard	11.03.200...
Team für Projekt gesu...	Wottawa Friedric...	09.03.200...
Gruppe 401152/3 (De...	Popp Harald	07.03.200...
RSS und Outlook	Derntl Michael	04.03.200...
Team für Projekt gesu...	Mangler Jürgen	04.03.200...
RSS und Outlook	Vigne Ralph	03.03.200...
Team für Projekt gesu...	Wottawa Friedric...	03.03.200...
Hinweis zum Forum	Derntl Michael	02.03.200...

Looking for folders... Unread: 0 Total: 11

Figure 34: RSS Version of The Discussion in a Mailbox

Again I wanted to focus on making a forum that is easy to use, and has not to be explained to new users. I selected a threaded form with the thread always visible at the bottom of the page. The top-level nodes of the thread are sorted in reverse order, so that the newest branches are always at the top (see Fig. 33). Another import point is the RSS integration (currently RSS 2.0). This feature can be used as follows:

- To subscribe to a forum with a mail-program (as can be seen in Fig. 34). If this possibility is chosen, the user has the entries in a list, and can sort them according to his / her own preferences. New messages are marked the same way as new mail messages would be. Once in the mailbox, they can be deleted like ordinary messages, and do not occur again with the next synchronisation. When a message is clicked, the forum entry is shown as message text, admittedly *reply* is not working directly from the mail programs we tested so far.
- When using with a browser that is able to handle RSS as bookmark folder, the last ten messages are shown (in reversed order, with the newest as first message).

The web version has the following characteristics:

- The entry point is a page that shows a list of all topics (plus a link to the RSS for each topic).
- When the user selects a topic he / she can see an introduction that explains the purpose of the particular forum.
- For every topic the threaded list is always visible at the bottom of the page.
- The root element of the thread is a link to post a new top-level message or question (*”Description. Click here to write new message.”*).

- For the message the user currently views, the subject in the thread is emphasised (bold and red with our current stylesheet).
- When the user answers, the message text is still displayed on the top of the page, the lower part (the threaded-list) is replaced by input-elements (subject, message text). So the user can easily copy parts of the message he / she is answering to and can re-read the message during the creation.
- After submitting a message, the user is automatically transfered to his / her answer. He / she has then the ability to change the message for ten minutes.
- All unread messages are marked (as can be seen in Fig. 33, the current stylesheet defines a dark gray background for them).

An additional advantage is, that the user can use parts of the WIKI syntax to format the messages. Allowed markup includes **strong**, *emphasis*, comments and annotations. Pasting source code, lists and indented text blocks are also allowed. Additionally all sorts of links (http://, mailto:, CEWebS://) are working. Headings, tables, images and literature references are not possible.

We have also added two reports to make the statistical analysis of a topic possible. The reports were first requested by *Christine Bauer* who did an usage analysis for one of our courses. I want to thank her for specifying them and testing their usefulness.

The first report returns a printable document for a whole topic. All messages are printed in threaded form in one big list. This is useful if one wants to read through a whole topic to recapitulate what was going on.

The second report returns all entries as CSV file. Every line in the file represents a message. The file has the format:

```
date;time;author;index number;level in the thread;text
```

The **index number** of the message is just a unique (to a topic) identifier for a message. The **level in the thread**' is a number that tells if the message is top-level (0), a direct answer to a top-level message (1) or something else (2, 3, ...).

5.5 Keeping a Diary

Keeping a project-diary is a useful instrument of tracking the own expenditure of time. When used as an instrument in project-teams it can help all members to follow the changes and track the results that have been achieved so far. On the other hand for the facilitator it can provide a means to find out early if something is going wrong.

Apart from the very simple use-cases, depicted in Fig. 35, we identified also the following functional requirements:

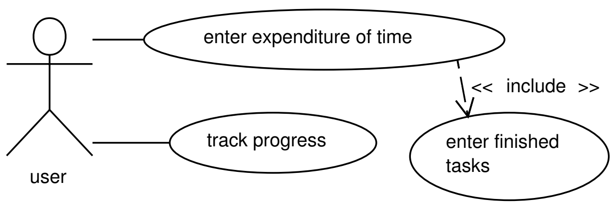


Figure 35: Track Progress

- The most important thing is, that the diary must be easy to use. If it is not easy to make an entry, or making it consumes to much time, it will most likely not be made. So ease of use and an uncluttered interface are a necessity.
- If the members of a team have not access to a centralised document, several versions will arise and potential inconsistencies and duplicated data will occur. So for our loose teams of students a web-based solution is necessary, as this is the only way to keep them in sync.

I suppose to add specialised components like the *diary* to be one of the genuine advantages that the CEWebS architecture provides. In other learning platforms we tried to realise a diary by creating forums that were only accessible by individual teams, but obviously this is not a good solution, as there have to be rules (like the format of the start-time and duration) and there are no analysis tools available.

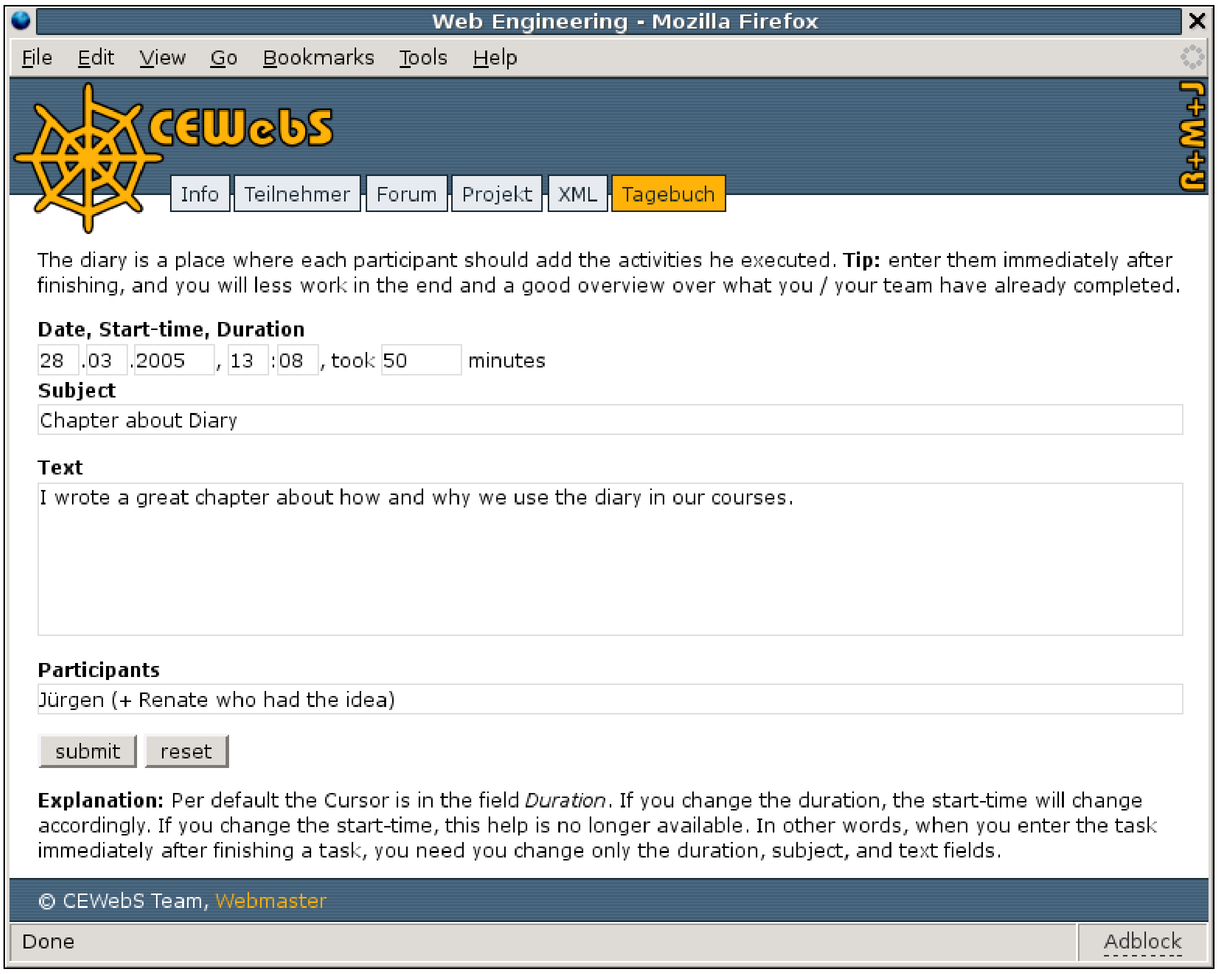
The resulting CEWebS on the other hand is very simple, highly specialised and easily extendable (by introducing other fields like *Importance* or *Category*). I consists of only two user-visible pages, one that presents a list of all tasks to the user and a second for creating / changing entries.

The list-page provides the ability to add new tasks, delete, and modify them. There exist also two convenience shortcuts *Alt+N* and *Alt+L* to create new entries or reload the list. When an entry is deleted the user sees immediately the updated list. There are two modes, a single-user mode and a team mode. In single-user a user sees only his / her own entries, in team mode a team can work together to keep the diary up-to-date.

Creating new entries has to be very simple. We realised the input form as shown in Fig. 36. When the user enters the form the duration field is already active. The default duration can be set for every instance and the start-time is set according to the actual time minus the duration. When the user changes the duration the start-time is modified to fit the change. On the other hand when the user changes the start-time, changing duration does no longer trigger any modifications. It is also possible to make the start-time and duration fields invisible for an instance.

Subject and **Text** are mandatory, **Participants** is an optional field where the user can write the names of people that took part in the execution of the task. The author of the task is implicitly part of this list, as only logged-in users can use the diary.

The facilitator can request a simple report, that enlists all entries of an actor (person / team) on a single page. This long form can then be stored as the base-documentation of a project and can help the facilitator and the students to discuss the project (and the grade).


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
R+W+J
Info
Teilnehmer
Forum
Projekt
XML
Tagebuch

The diary is a place where each participant should add the activities he executed. **Tip:** enter them immediately after finishing, and you will less work in the end and a good overview over what you / your team have already completed.

Date, Start-time, Duration

28 .03 .2005 , 13 :08 , took 50 minutes

Subject

Chapter about Diary

Text

I wrote a great chapter about how and why we use the diary in our courses.

Participants

Jürgen (+ Renate who had the idea)

Explanation: Per default the Cursor is in the field *Duration*. If you change the duration, the start-time will change accordingly. If you change the start-time, this help is no longer available. In other words, when you enter the task immediately after finishing a task, you need you change only the duration, subject, and text fields.

Figure 36: Create a New Diary-Entry

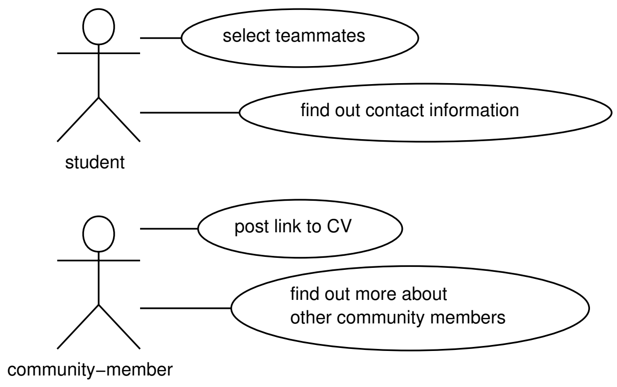


Figure 37: Build Teams and Keep In Contact With a Community

5.6 Team-Building and Communities

The two services that allow for team-building (**Participants**) and communities (**Community**) were designed with different goals in mind. Their commonality is to show the names and addresses of people that are authorised for TE. We identified the use-cases shown in Fig. 37.

Participants is a service for students. All students belong to a group and the task this CEWebS facilitates is the building of teams. The process is self-contained and usually the students have one week to form the teams (adding a forum for them to allow for interaction is wise though). When a student enters the page he / she sees a list of all users. After logging in (through clicking on a link that points to the password-protected version of the same page) the page has the following characteristics:

- User IDs and email-addresses are visible.
- There is a compartment named **Pool** where all students' names reside.
- When a student clicks on a desired team-mates name, a new team is created and the two people are assigned to it.
- When a user that is already in a team clicks on the heading of the pool, he / she is removed from his / her current team and put back to the *Pool*.
- When a user clicks on the team-name of any other team he / she is moved to this team (either from the *Pool* or his / her current team).
- When a user is already in a team and clicks a person from the *Pool*, this person is also added to the team.
- A person cannot add people to his / her team, that are already in some other team (people CAN move to other teams like mentioned above, but only voluntarily).

The facilitator has two possibilities. He / she can set a maximum size of people per team and when he / she considers the process to be over he / she can lock the interface:

- The maximum size of people is global, that means that a team can only consist of a maximum number of students (no exceptions here). When a user tries to change to a team that is already full (or when he / she is selected from the pool) nothing happens. It is wise though to tell the students about the restriction or to add it to the introduction text.
- The facilitator can lock the service after some time. This changes the behaviour slightly:
  - No student can move back to the *Pool*.
  - No student that is already in a team can change to a different team.
  - Students in the *Pool* can still move to teams and can also be selected from the *Pool*.

This service has many advantages over the methods mentioned in previous chapters (e.g. collecting sheets of paper with names). The students can form the teams without a pressure, additionally no time is wasted during face to face (f2f) meetings. No typos or errors can occur, the whole process is transparent to both the students and the facilitator.

The **Participants** service has some functionality (reports) built-in, that cannot be labelled report but can be considered as convenience stuff, that fits best with this particular CEWebS. This includes printing data-sheets, where all teams and persons are contained in matrices. This comes in handy for a facilitator during a f2f meeting, he / she can take notes about contributions of students. There is also a report that enables writing mails to a whole group or single teams.

Future improvements may include a simple grading / messaging system, where the facilitator can write private messages to a student or add brownie points and notices to a particular team or person.

The **Community** service on the other hand has no functionality to build teams. It focuses on getting a community started:

- A list of all members is provided.
- Every person can set a link, that is the target for clicking on his / her name. This can be used to link to a CV, a blog or something similar.
- The email-address for every person is printed behind the name, it is clickable and can also be copied to a mail program.
- A link that opens the standard mail program with the addresses of all people as receivers is provided. This comes in handy when one wants to drop a quick note to everybody.

Setting the link to personal information is very easy. There is just one input field and a submit button. As said before this service should enable a small community to get / keep in touch. If the community grows I would suggest using something more sophisticated like a mailing list.

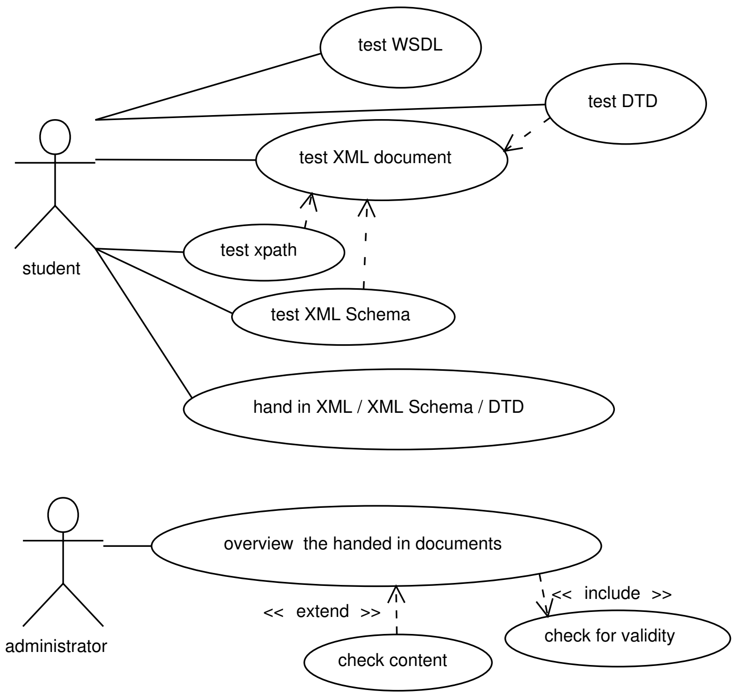


Figure 38: Various XML Related Helper-Functions

5.7 XML, BugTracker, GraphTalk

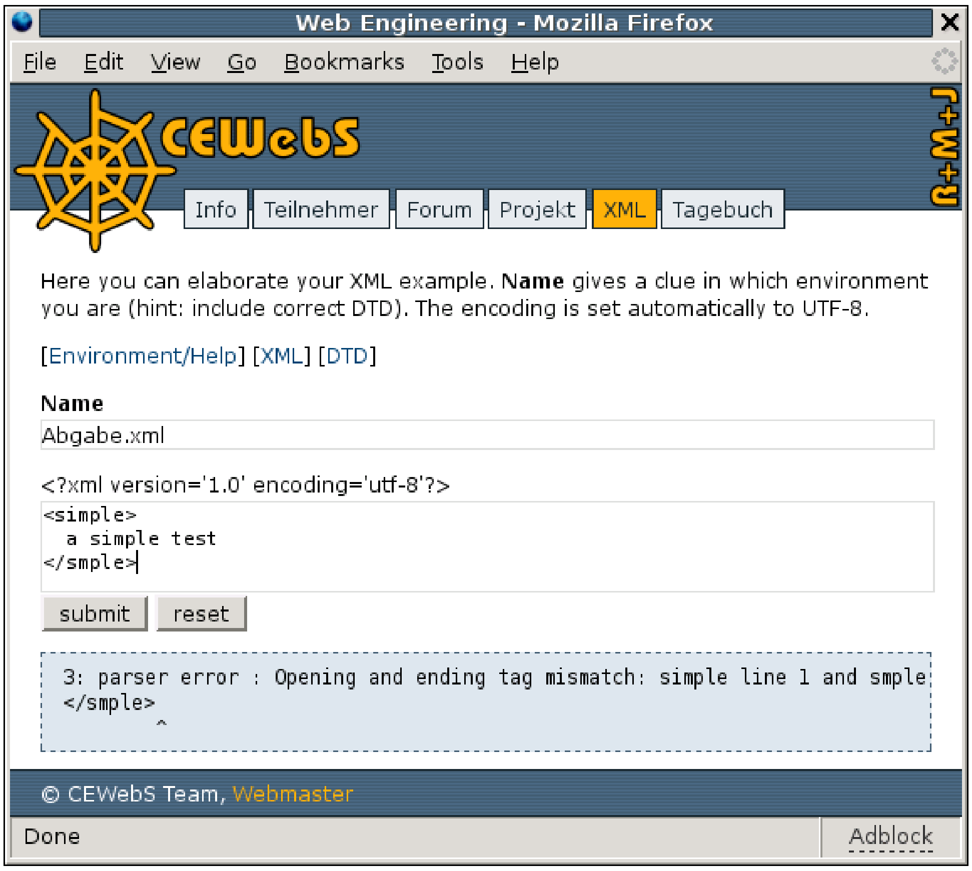
These CEWebS are closely related to our current courses and the field of Computer Science (CS) in general. They provide functionality that can not be found in any other platform. They also show why developing an architecture that helps embedding / using arbitrary services, programming languages and techniques is definitely an advantage.

5.7.1 XML

The first CEWebS called **XML** helps to handle XML related homework. It is not team-oriented but provides an XML test-bed for every student where he / she can try out different XML related technologies without the necessity to install tools on his / her local computer. For the facilitator the CEWebS has the advantage that he / she can track the students' progress and that he / she can concentrate on the semantics of the XML-files (the syntax checking is done by CEWebS). He / she can request detailed reports that show what pieces are syntactically correct and provide an aggregated view on the pieces. This is useful for a concluding dialogue with a student.

We identified the use-cases as depicted in Fig. 38. The CEWebS has the following abilities:

- There are several environments a student can use. An environment is a space where pieces of XML can be stored and tried out. When the student is sure everything works he / she can move it to the contribution-environment. Only the version in the contribution-environment will be graded.


Web Engineering - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Info Teilnehmer Forum Projekt XML Tagebuch

Here you can elaborate your XML example. **Name** gives a clue in which environment you are (hint: include correct DTD). The encoding is set automatically to UTF-8.

[Environment/Help] [XML] [DTD]

Name

Abgabe.xml

<?xml version='1.0' encoding='utf-8'?>

```
<simple>
  a simple test
</smple>
```

```
3: parser error : Opening and ending tag mismatch: simple line 1 and smple
</smple>
        ^
```

© CEWebS Team, Webmaster

Figure 39: The XML Tool in Action

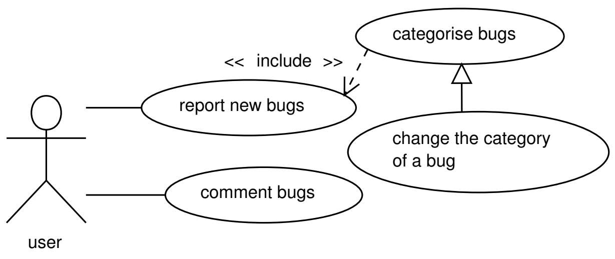


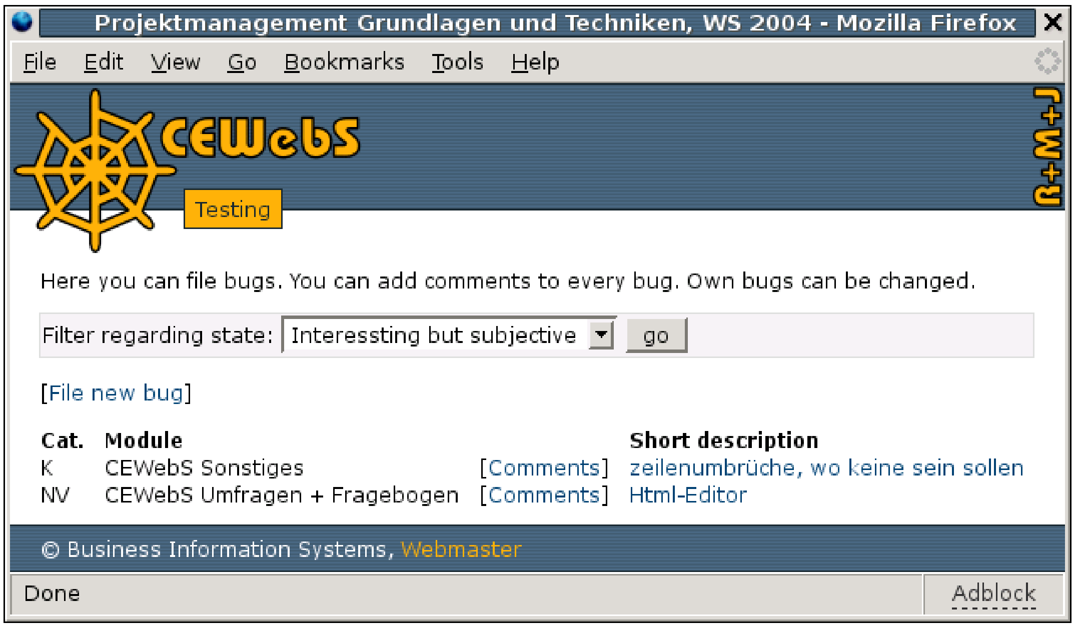
Figure 40: Report Bugs

- The user gets an **XML** environment (see Fig. 39, the input box is reduced in size to keep the screenshot small) where he / she can try out XML documents. The parser tries to provide the user with helpful information about possible errors. If everything is okay, it states *”Wellformed document”*.
- The **DTD** environment is similar. But there are some traps hidden in DTD. First the user has to include the correct DTD (as shown in the box below **Name**) in the *XML*. The DTD has to be syntactically correct and it has to match the *XML* structure. Again the output for the error-case is designed to be helpful for beginners.
- The **Schema** environment does basically the same as the *DTD*. It checks Schema against the *XML*. An inclusion in the *XML* is not allowed, as it would break the *DTD*. The Schema is automatically applied to the document root. For the *Schema* environment helpful output by the parser is even more important.
- The **XPath** environment allows to test XPath expressions against the *XML*. The result is then printed, or highlighted in a static version of the document.
- The **WSDL** environment is completely independent from the rest of the environments. It checks WSDL files, and identifies errors, by consuming a web-service that exposes the .NET [20] WSDL parser. Thanks to Michael Derntl for maintaining the web-service that makes this environment possible.

This CEWebS encourages the students to play around with the technologies after they hear about them during the lecture. We think easy access to technologies and the *”just try it out”* philosophy helps keeping learning exciting and fun.

5.7.2 BugTracker

The second CEWebS is called **BugTracker**. It has been designed to support e.g. our entry level project-management course (*PM-GT - Projektmanagement Grundlagen und Techniken*). Its purpose is to provide an easy bug-reporting tool for the students’ projects. They have to systematically test the project of a peer-team and report the bugs through the web-interface. Again this helps to keep teams in sync, and to divide the work even if there is no time to meet f2f. Commercial or Open-Source bug-tracking tools are often very complicated to use, so we tried to simplify the process. The use-cases in Fig. 40 where identified as being important.


Projektmanagement Grundlagen und Techniken, WS 2004 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Testing

Here you can file bugs. You can add comments to every bug. Own bugs can be changed.

Filter regarding state: Interessting but subjective go

[File new bug]

Cat.	Module		Short description
K	CEWebS Sonstiges	[Comments]	zeilenumbrüche, wo keine sein sollen
NV	CEWebS Umfragen + Fragebogen	[Comments]	Html-Editor

© Business Information Systems, Webmaster

Done Adblock

Figure 41: A List of Bugs, Sorted by Category

The CEWebS in its current implementation is not able to handle / manage more than say 200 bugs. For the service to scale to higher numbers an efficient search mechanism and a way to get notified about status changes and comments being added to bugs would be needed. In its current form it is able to file bugs, change their state and add comments to them.

When the user enters the service he / she can see an interface as depicted in Fig. 41. An administrator can add an arbitrary number of bug-states to the configuration, the first state in the configuration is the one that is shown per default. States in our configuration include (in exactly this order):

- **Open** - bugs that have not yet been resolved.
- **Squished** - bugs that have been resolved / fixed.
- **Already Existing** - bugs that have already been reported / duplicated bugs.
- **Interesting but subjective** - ideas or reported errors that belong to the category *”feature not bug”*. Also complaints about colours and page layout can be put to this category. It holds bugs that need further discussion.

The bugs can be filtered according to these states. For our configuration all *Open* bugs are shown in the default view. The list is sorted by modules and categories. Modules are the pieces of software where the bug occurred, and they are the primary sorting criteria. Categories include:

- **Error** - an error message occurs, or something just refuses to work.

- **Not available, missing functionality, Idea**
- **Disturbing, Usability** - something works but is not nice to use.
- **Minor flaw** - nice to have, but low priority.

Again the administrator can define his / her own categories, the ordering in the configuration file affects the ordering in the output list. In our case errors are considered the most important thing, followed by ideas and so on. There is also a link to the comments section of the bug and a short description, that is clickable and links to the detailed description of the bug.

When the user decides to add a new bug, he / she will see a form with the following input fields:

- **Date, Time - Regarding**: *Date* and *Time* can be set as well as the module where the error has been found (*Regarding*).
- **Author and Browser** will be set automatically, as only logged-in users can add bugs, and the browser can also be read out. This field is immutable.
- The **Short description**of the bug that will be shown in the list.
- An **URL** that points to a location where the error occurs. If an URL can't be given (due to frames or for session reasons e.g. in WebCT) one should add here a short description how to provoke the error.
- Long **Description**
- **Effects** that occur because of the error. What things can no longer be accomplished?
- **Additional Comments** that fit into none of the above fields.

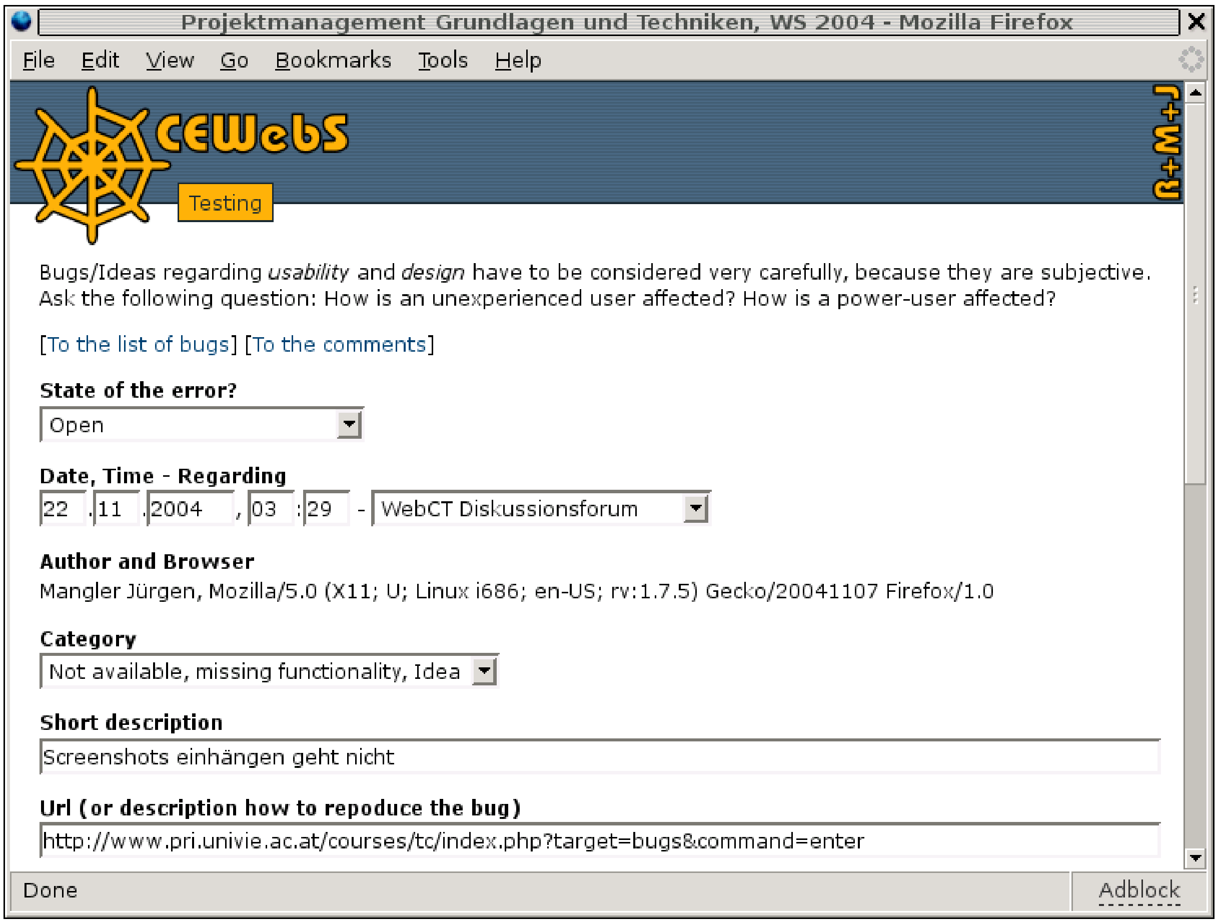
A user can change the bugs he / she reported (the facilitator can change all bugs) as can be seen in Fig. 42. All new bugs default to *Open* (or in fact the first state in the configuration), after first saving one, the user can also change the state (e.g. to *Already Existing*).

The last functionality that is included is the ability to add comments to every bug. The comments will be shown in a flat list, every comment can make use of the above mentioned simple WIKI syntax.

5.7.3 GraphTalk

The CEWebS called **GraphTalk** is a general purpose component for cooperative chatting and modelling as described in [17] and [18]. The use-cases have been identified as shown in Fig. 43 and it has the following features:

- Chatting with all users that are authorised for a particular TE.


Projektmanagement Grundlagen und Techniken, WS 2004 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
CEWebS
Testing
R+M+J

Bugs/Ideas regarding *usability* and *design* have to be considered very carefully, because they are subjective. Ask the following question: How is an unexperienced user affected? How is a power-user affected?

[To the list of bugs] [To the comments]

State of the error?
Open

Date, Time - Regarding
22 .11 .2004 , 03 :29 - WebCT Diskussionsforum

Author and Browser
Mangler Jürgen, Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.7.5) Gecko/20041107 Firefox/1.0

Category
Not available, missing functionality, Idea

Short description
Screenshots einhängen geht nicht

Url (or description how to repoduce the bug)
http://www.pri.univie.ac.at/courses/tc/index.php?target=bugs&command=enter

Figure 42: Owner of a Bug

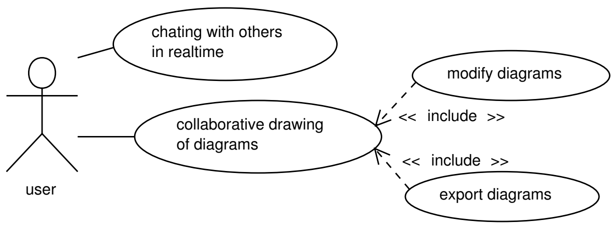
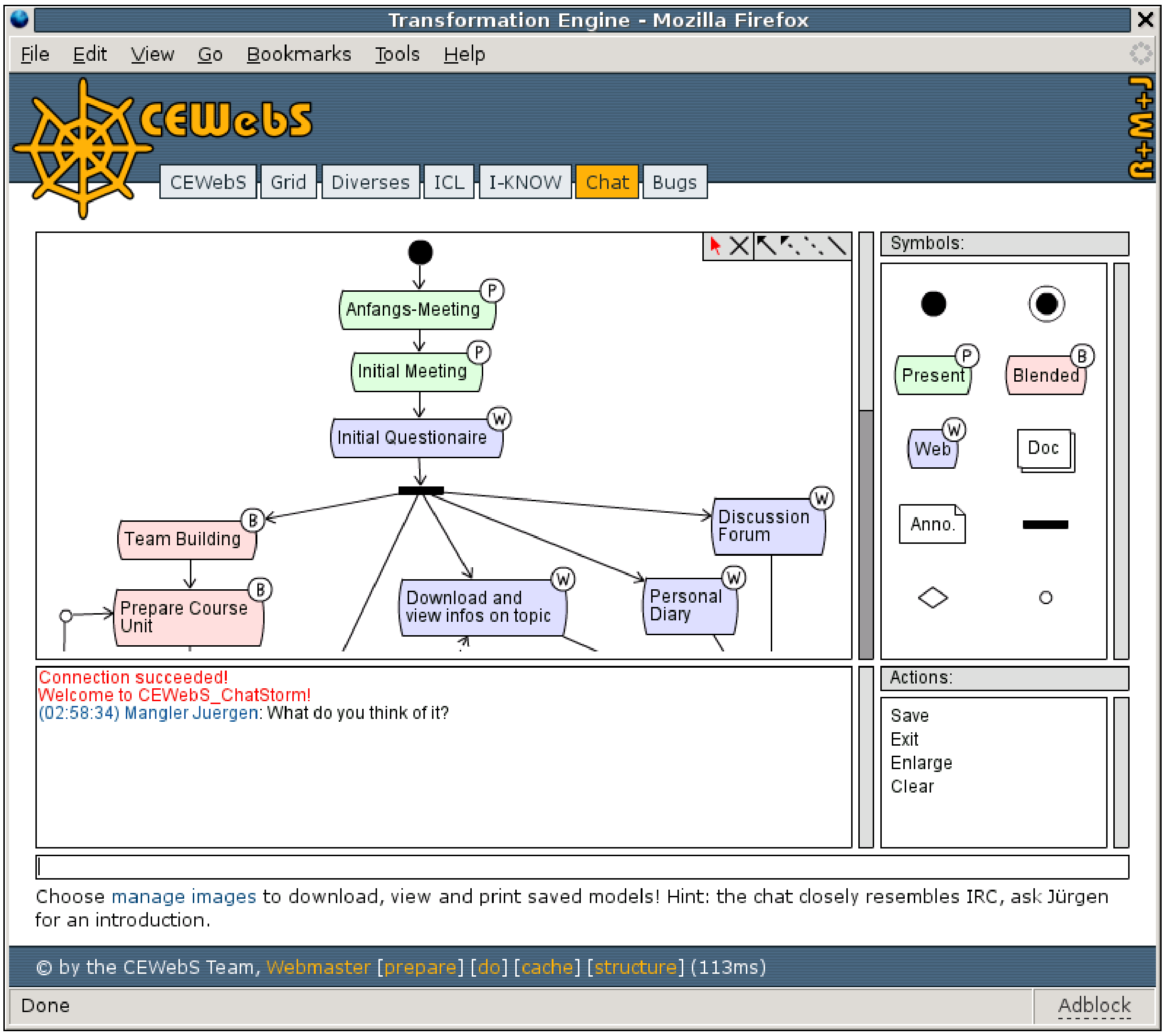

chating with others in realtime
modify diagrams
<< include >>
collaborative drawing of diagrams
<< include >>
export diagrams
user

Figure 43: Chatting and Modelling



Connection succeeded!
Welcome to CEWebS_ChatStorm!
(02:58:34) Mangler Juergen: What do you think of it?

Choose manage images to download, view and print saved models! Hint: the chat closely resembles IRC, ask Jürgen for an introduction.

© by the CEWebS Team, Webmaster [prepare] [do] [cache] [structure] (113ms)

Figure 44: A BLESS Model

- Inviting people to a whiteboard where not only drawing but true modelling is possible (including modifying elements, making connections between elements).
- It is easily extendable to include new sets of elements. Currently existing sets include elements for activity diagrams (BLESS modelling) (Fig. 44) as well as elements for use-case models and Rich-Picture diagrams.

6 Conclusion and Outlook

This thesis has shown our experience regarding the support of courses with learning technology. These experiences lead to the creation of CEWebS, which is a framework that defines interfaces and data-formats to allow for the integration of web-services into learning-platforms (or other applications in general) as well as the interaction of these services. This thesis has also given an overview over particular services we created and how they help us to create better courses and focus on interesting topics rather than on administrative tasks.

Currently we are working on new services including calendars with desktop integration, a front end for the Internet Relay Chat and services to make the grading process more transparent to the students throughout the whole course. There is also ongoing work to make the web-services easy to configure, even for people with little technical understanding.

We think that by making web-service technologies usable also for learning environments and by proposing cross-platform standards we can facilitate cooperation and interoperability inside and between organisations. Learning related web-services that provide interfaces (WSDL) [36] and can be used through the Simple Object Access Protocol (SOAP) [38]. By utilising this approach current static learning platforms can be gradually replaced by dynamic networks of web-services. We envision the genesis of virtual organisations [10] that exchange learning services, content and didactic elements through the means of web-services.

By making the source-code available online we hope to create a community, that helps us to propagate the idea. There will also be a strong focus towards cooperation with others to eventually make the virtual organisations possible, to share and to learn from others.

Researching new technologies, finding test-cases, testing interoperability and providing interested parties with rules for applying web-services based technologies could spur exchange and collaboration and enrich the learning experience and possibilities for both, facilitators and learners.

7 Attachment: Installation

For all services introduced in this thesis, the following prerequisites apply:

- Ruby >= 1.8.2
- Ruby XML Simple (http://raa.ruby-lang.org/project/ruby-xml-simple/)
- Webrick (included in Ruby 1.8.2)
- Soap4R >= 1.5.2 (included in Ruby 1.8.2)

For *Scientiki* there is additionally needed:

- rmagick > 1.7
- libalgorithm-diff-ruby1.8 (from Taniguchi Takaki)
- Subversion > 1.0

The *XML* service needs also:

- libxml >= 2 (plus xmllint utility)
- libxslt >= 1 (plus xsltproc utility)
- libxerces >= 2.3 (plus xerlint utility)

TeamWorkspaces, a web-service by Michael Derntl that allows for web-based file-management, needs the .NET Framework >= 1.1.

The TransformationEngine is totally independent from everything else and needs:

- PHP >= 4.3 (not PHP 5) with dom-libxml
- The current implementation also depends on environment variables that are only set by Apache > 1.3

Packaging is not yet available but will most likely include:

ServicesBase-X-X-X.tgz: All WSDL interfaces.

ServicesBase_Ruby-X-X-X.tgz: Helper programs and common ruby classes that are needed by all services. The most important file is **server.rb** which starts the web-server for a any Ruby CEWebS. It depends on *ServicesBase*.

TransformationEngine-X-X-X.tgz: Depends on *ServicesBase*.

CEWebS_NAME-X-X-X.tgz: All services will be provided in separate packages. All Ruby CEWebS will depend on *ServicesBase_Ruby*.

Installing CEWebS will be as simple as putting *ServicesBase*, *ServicesBase_Ruby* and all services *CEWebS_NAME* into the same directory. Changing into a *CEWebS_NAME* directory and issuing a **./server.rb start &** will then start a particular CEWebS.

The Ruby CEWebS are realised in a very modular fashion. Every service includes the same set of directories. If a directory is present, the *server.rb* will dynamically load all functionality included in the directory.

- The **Administration** directory holds the implementation of the *Administration* interface.
- The **Delivery** directory holds the implementation of the *Delivery* interface.
- The **Report** directory holds the implementation of the *Report* interface. If this directory is not present, the particular interface will not be loaded / available.
- The **Data** directory holds all instances and the data they collected. There exists at least a directory **Default** that holds a working default configuration. When copied to a directory with a numeric name (e.g. 1) a working instance of the same name will exist (no restart of the CEWebS is required to use it). The configuration is held in a file called **custom.xml**, the users are saved in a file called **participants.xml**.
- The **Logs** directory holds several logs with access as well as performance information.

Every Ruby CEWebS has a **server.port** file in its main directory that holds the port it listens on (obviously this file can be changed). The default access URLs are:

- http://domain:port/Administration/wsdl
- http://domain:port/Delivery/wsdl
- http://domain:port/Report/wsdl

The *server.rb* is very flexible and allows also to start services only local (listening only to localhost). Setting up the TE has been explained in previous chapters. Most likely there will be a prepackaged version for Debian Sid and Ubuntu Hoary.

References:

[1] BSCW Team, Fraunhofer FIT, OrbiTeam Software GmbH: ”BSCW”. http://bscw.fit.fraunhofer.de/ [last access February 25, 2005]

[2] Collins-Sussman B., Fitzpatrick B., Pilato C.: ”Subversion. Next-Generation Open Source Version Control”. http://svnbook.red-bean.com/ [last access September 17, 2004], O’Reilly Media Inc. (2004)

[3] Corkill, D: ”Blackboard Systems”. AI Expert 6(9) 40-47 (September 1991), http://dancorkill.home.comcast.net/pubs/ai-expert.pdf [last access February 22, 2005]

[4] Curbera, F., Nagy, W., Weerawarana, S.: ”Web Services: Why and How;”. OPSLA 2001, Workshop on Object-Oriented Web Services, Florida, USA, (2001) http://www.research.ibm.com/people/b/bth/OOWS2001.html [last access February 25, 2005]

[5] Derntl, M., Motschnig, R.: ”BLESS - A Layered Blended Learning Systems Structure”. Proceedings of I-Know 2004, Graz (2004)

[6] Derntl, M., Motschnig, R.: ”Patters for Blended, Person-Centered Learning: Strategy, Concepts, Experiences, and Evaluation”. Proceedings of SAC 2004 2004, Nikosia, Cyprus (2004)

[7] Derntl, M., Motschnig, R.: ”A Pattern Approach to Person-Centered e-Learning Based on Theory-Guided Action Research”. Proceedings of Networked Learning Conference (NLC) 2004, Lancaster, UK (2004)

[8] European Union, Information Society: ”Free and Open Source Software”. http://europa.eu.int/information_society/activities/opensource/ [last access February 16, 2005]

[9] Fielding et al.: ”RFC 2616”. http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html [last access February 16, 2005], part of Hypertext Transfer Protocol (HTTP/1.1)

[10] Foster I., Kesselman C., Tuecke S.: ”The Anatomy of the Grid: Enabling Scalable Virtual Organisations”. International Journal Supercomputer Applications, 2001

[11] Free Software Foundation: ”Violations of the GPL, LGPL, and GFDL”. http://www.fsf.org/licenses/gpl-violation.html [last access February 16, 2005]

[12] Free Software Foundation: ”General Public License (GPL)”. http://www.fsf.org/licensing/licenses/gpl.html [last access February 16, 2005]

[13] Free Software Foundation: ”Lesser General Public License (LGPL)”. http://www.fsf.org/licensing/licenses/lgpl.html [last access February 16, 2005]

[14] ILIAS open source, University of Cologne: ”Ilias”. http://www.ilias.uni-koeln.de/ios/ [last access February 19, 2005]

[15] ImageMagick Studio LLC: ”ImageMagick”. http://www.imagemagick.org [last access February 19, 2005]

[16] Macromedia Corporations: ”Macromedia Flash”. Information and plugins available at http://www.macromedia.com/software/flash/ [last access June 15, 2004]

[17] Mangler J., Bauer C.: ”A CEWebS Component for Facilitating Online Cooperative Brainstorming Processes”. Proc. of the ICL 2004, Klagenfurt (2004), available at http://www.pri.univie.ac.at/ mangler/Papers/2004-ICL-chatstorm.pdf [last access February 19, 2005]

[18] Mangler J.: ”GraphTalk”. Software Engineering Homework (2005), available at http://www.pri.univie.ac.at/ mangler/Papers/2005-SE-graphtalk.pdf [last access February 19, 2005]

[19] Microsoft Coorporation, ”Component Object Model, ActiveX”. http://www.microsoft.com/com/ [last access February 19, 2005]

[20] Microsoft Coorporation, ”Microsoft .NET Framework”. http://www.microsoft.com/net/ [last access February 19, 2005]

[21] Microsoft Coorporation, ”eXtensible Application Markup Language (XAML)”. http://longhorn.msdn.microsoft.com/lhsdk/core/overviews/about%20xaml.aspx [last access February 19, 2005]

[22] Motschnig, R., Holzinger, A.: ”Student-Centered Teaching Meets New Media: Concept and Experiences”. IEEE Educational Technology and Society, 5, 4 (2002), 160-172

[23] Motschnig, R., Derntl, M.: ”Cooperative Person-Centered e-Learning: Moving from Exams to a Web-Engineering Knowledge Base”. Proceedings of Interactive Computer-Aided Learning (ICL) 2003, Villach (2003)

[24] Mozilla Organisation: ”XML User Interface Language (XUL)”. http://www.mozilla.org/, http://xulplanet.com/tutorials/whyxul.html [last access February 19, 2005]

[25] Nielsen, J.: ”The Top Ten New Mistakes of Web Design”. http://www.useit.com/alertbox/990530.html [last access February 19, 2005], Alertbox (May 30, 1999)

[26] Open Mobile Aliance (OMA): ”Wireless Markup Language (WML)”. http://www.phone.com/dtd/wml11.dtd, http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html [last access February 19, 2005]

[27] Plone Foundation: ”Plone”. http://plone.org/ [last access February 19, 2005]

[28] Python Software Foundation: ”Python”. http://www.python.org/ [last access February 20, 2005]

[29] RSS-DEV Working Group: ”RDF Site Summary (RSS) 1.0”. http://web.resource.org/rss/1.0/spec [last access February 16, 2005]

[30] Sun Microsystems, Inc.: ”Java”. http://java.sun.com/ [last access February 16, 2005]

[31] Sommerville, Ian: ”Software Engineering, 4th. ed.”. Addison-Wesley, Workingham, England (1992)

[32] WebCT Incorporated: ”WebCT”. http://www.webct.com/ [last access September 17, 2004]

[33] Wikimedia Foundation: ”ActiveX”. http://en.wikipedia.org/wiki/ActiveX [last access September 17, 2004], Wikipedia

[34] Wikimedia Foundation: ”Regular Expressions”. http://en.wikipedia.org/wiki/Regular expressions [last access September 17, 2004], Wikipedia

[35] Wikimedia Foundation: ”Wikipedia: How to edit a page”. http://en.wikipedia.org/wiki/Wikipedia:How_to_edit_a_page [last access September 17, 2004], Wikipedia

[36] W3C: ”Web Services Description Language (WSDL) 1.1”. http://www.w3.org/TR/wsdl [last access February 16, 2005]

[37] W3C: ”Extensible Markup Language (XML)”. http://www.w3.org/XML/ [last access February 16, 2005]

[38] W3C: ”SOAP Version 1.2 Part 0: Primer”. http://www.w3.org/TR/2003/REC-soap12-part0-20030624/ [last access February 16, 2005]

[39] W3C: ”The Extensible Stylesheet Language Family (XSL)”. http://www.w3.org/Style/XSL/ [last access February 16, 2005]

[40] W3C: ”Platform for Privacy Preferences (P3P) Project”. http://www.w3.org/P3P/ [last access February 16, 2005]

[41] Zope Coorporation: ”Zero Objects Programming Environment”. http://www.zope.org/ [last access February 19, 2005]

List of Figures: