

SubCVS:

Merging version control systems in a servlet-based repository browser

January 16, 2008

Author: A.S.S. Oemraw
aoemraw@hotmail.com

Advisor: B. Kienhuis
Leiden Institute of Advanced Computer Science
Niels Bohrweg 1, 2333 CA Leiden, The Netherlands
kienhuis@liacs.nl

University of Leiden
Rapenburg 70, P.O. Box 9500, 2300 RA Leiden, The Netherlands

In this project, my goal is to improve and extend the functionality of SubCVS, a servlet-based manager for the CVS and SVN current version systems.

Table of contents

| | |
|--|----|
| Introduction | 3 |
| Aims and Objective | 4 |
| Methodology | 5 |
| UML of the old SubCVS..... | 5 |
| UML of the new SubCVS | 6 |
| Motivation and implementation of the UML model | 7 |
| Enhanced features | 10 |
| Interface..... | 10 |
| Database | 13 |
| Refactoring & extending classes | 14 |
| LoadModule and AJAX | 17 |
| New features..... | 19 |
| Automatic checkout..... | 19 |
| Automatic updates..... | 19 |
| Recent updates..... | 20 |
| Module Management..... | 22 |
| SubCVS Manager..... | 22 |
| Casestudy | 24 |
| Conclusion..... | 25 |
| Discussion | 26 |
| References | 27 |

Introduction

Version control (also known as revision control, source control or source code management, SCM) is the management of multiple revisions of the same unit of information. It is commonly used, but finds most of its application in software development. In software development documents and especially files containing source code are subject to a lot of change. It is therefore necessary to keep track of these changes, as every change could lead to an improvement, but also to errors or problems. One can imagine the situation where programmers need to remove code they have added, when the newly added code causes problems.

To keep track of the changes to a file, or even all the files in a project, version control systems can be used. These systems identify changes made to files and store a version of the file for each change on a repository. Files get a revision number and this number is incremented with each new revision (change). There are several version control systems. Two systems that are well known are CVS (Concurrent Versions System [1]) and SVN (Subversion [2]), they are both used within Liacs[3].

A project can be managed by CVS or SVN (or any other version control system). Its files and older revisions of the files can be viewed from the version control system's repository. When a person is working with different projects, it is quite possible that he has to work with several version control systems (as each project can be controlled by a different version control system). This can be time consuming. Thus came the concept of SubCVS.

SubCVS is a servlet-based browser application that supports both CVS and Subversion repositories. Using SubCVS an user can browse and manage both CVS and SVN repositories from a web-browser, which makes it possible for users to keep track of projects from any machine connected to the Internet.

Aims and Objective

SubCVS, implemented by Maarten Vijfhuizen, provided a web application to browse repositories with support for CVS and SVN repositories, making it a very welcome addition to a software engineers set of tools. More about SubCVS can be read in “SubCVS: Merging version control systems in a servlet-based repository browser”, by Maarten Vijfhuizen [4].

SubCVS has the following features:

- Checkout or update a local copy of specified projects.
- Browse the contents of a project. If needed, a local checkout is done first.
- Open, add or delete files. For CVS, the '-kb' option can be specified when adding a binary file. Subversion auto-detects this.
- Explore the 'log' of a file, which contains information about all revisions of the file since it was imported into the version control system.
- View a specific revision of a file as ASCII.
- Compare two specific revisions of a file.

SubCVS provides all the basic functionality we would expect. It makes it possible for users to browse and manage their CVS and SVN repositories. The concept of a browser for multiple version control systems has been realized in SubCVS. However, the current implementation of SubCVS can still be enhanced and extended. Our goal is to continue the development of SubCVS and add some extra features and improvements.

In the current implementation of SubCVS, repositories are listed in an external text file. If the user wishes to add a repository, he has to add it to the list of modules in this text file. When the servlet loads, the modules are listed and the user can browse the individual modules and view the files, etc. The application checks out the latest version of the files, and the user can update modules, but this has to be done manually. Also, except from manually requesting a compare operation between 2 versions of a file, the user does not see what files are updated, nor what changes are made. These are some points that we could improve. What we would like, is to make SubCVS more intuitive, automatic and independent.

The aspects we would like to enhance in SubCVS:

- Make SubCVS database driven
- Build a new interface
- Refactor and extend classes

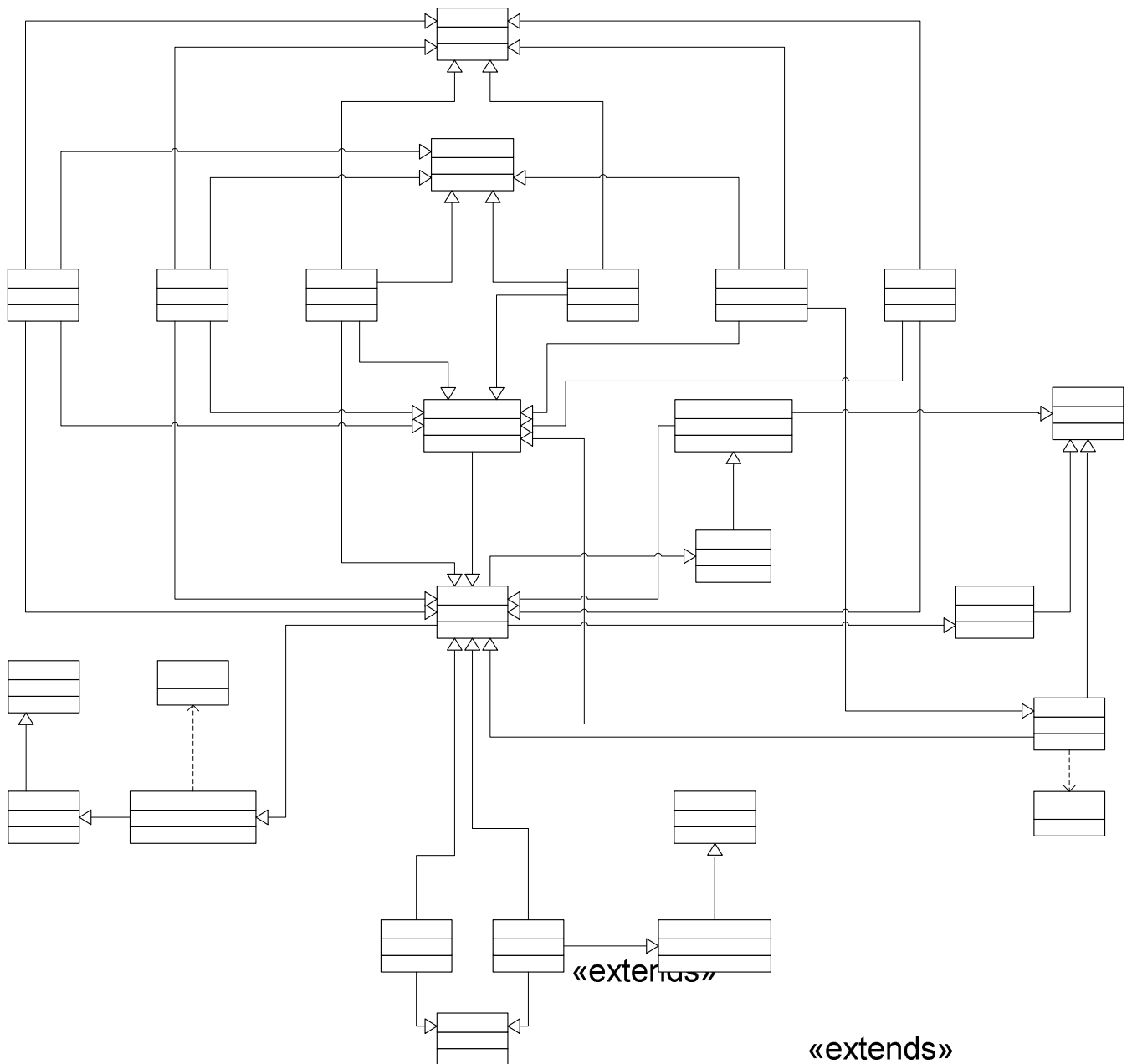
The aspects we would like to add to SubCVS:

- Automatic updates of the repositories in SubCVS
- User receives notification of updates when the servlet is loaded
- Adding, editing and removing CVS/SVN repositories from the servlet
- Add a manager for SubCVS settings

UML of the new SubCVS

Although the old SubCVS structure was good, it was necessary to change the structure to implement the new features enhancements. The old structure was able to support the basic functionality, but the structure was difficult to extend to support the new features. Extending the old structure would lead to very complicated structure and code.

The new structure needed to support the old features, as well as the new features, and provide the option for future extensions. The UML of the new structure can be seen in Figure 2.



Although the new model might look complicated at first glance, examining it will show that the underlying structure is quite simple.

If we compare this to the new model, we can see that the part containing the basic functionality is still present and unchanged. This is quite logical, as the enhancements and new features added to SubCVS do not change the basic functionality.

If we focus on the other part, we see that the presentation and organization part has been changed drastically.

With the application being extended, it would be very difficult and complicated to present all features and information using a single servlet. Also, by using a single servlet for the whole application, the servlet would have a very big workload, which would make it slow and more subject to problems. As a solution, the application has been split in several servlets, each having its own specification. If we would replace the different servlets in the new UML model with a single servlet, it would resemble the old UML model more.

All servlets have the same basic layout, menu, header, footer, etc. on a page. The code for these parts of a website is collected and stored in the HtmlTemplates class. This way we can reuse the code. For that reason each servlet is connected with the HtmlTemplates class.

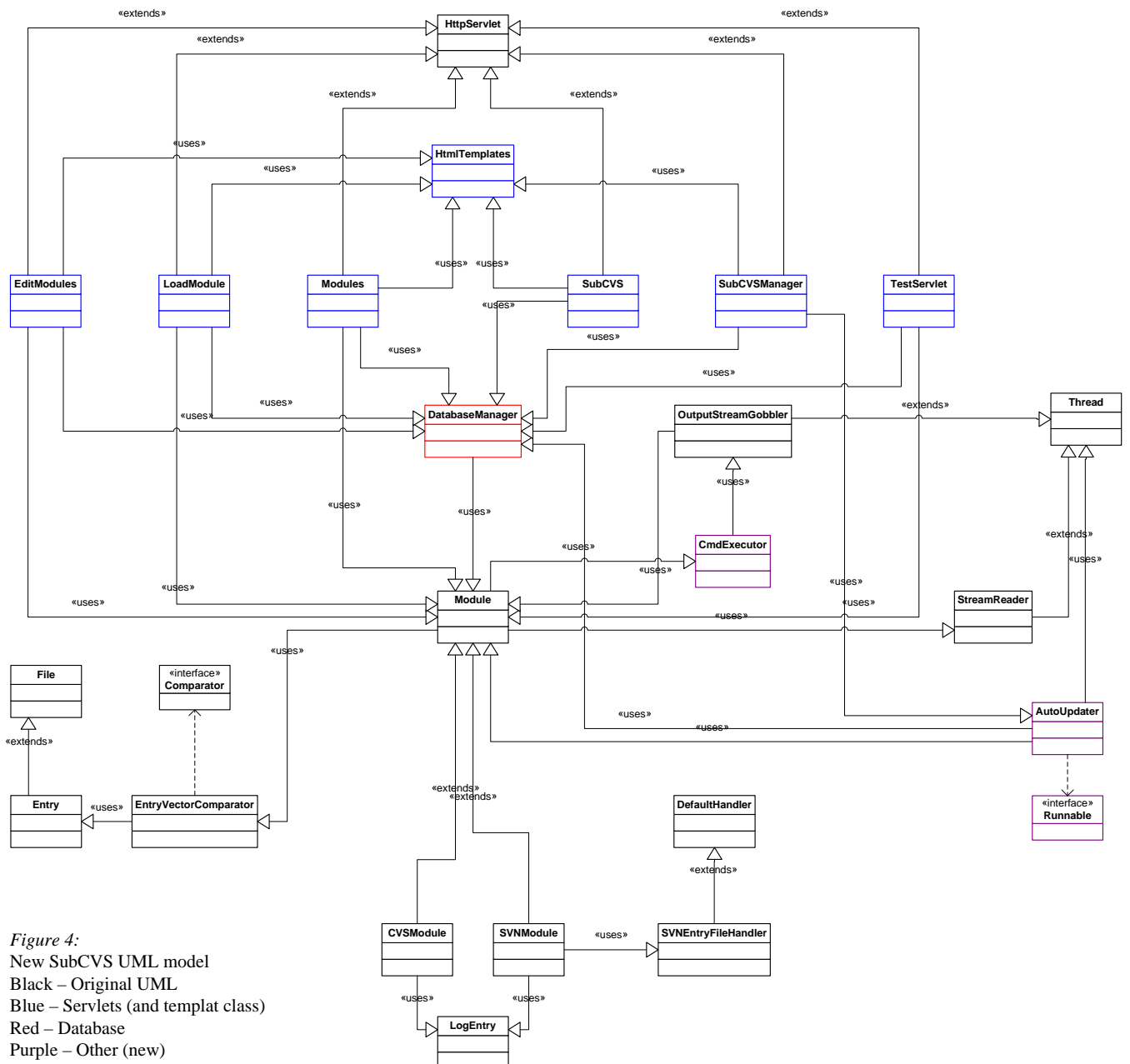


Figure 4:
 New SubCVS UML model
 Black – Original UML
 Blue – Servlets (and templat class)
 Red – Database
 Purple – Other (new)

As every servlet works with one (or multiple) modules, each servlet has a connection with the DatabaseManager class. The DatabaseManager class plays an important role in the application. In the original design, modules managed with SubCVS were stored in a text file. This has been removed and replaced by the use of a database to manage the modules. The DatabaseManager class contains the functionality to interact with the database to retrieve and/or store information.

Other additions to the model include classes that implement the auto-update feature and an executor class. In the original model each shell command that was to be executed was implemented locally. In the new model a CmdExecutor class has been introduced to execute all shell commands. As the application has more functionality also executes more shell commands, the CmdExecutor class is used to execute and facilitate the commands.

Enhanced features

Interface

SubCVS provides the user with a very simple interface. This interface provides the user with the means to make full use of the SubCVS features, but it could be improved. The interface could be made more user-friendly and pleasing to the eye. To make the application more intuitive and aesthetic, a new interface was designed and implemented. Figure 5 shows the original SubCVS interface.

The screenshot displays the original SubCVS interface. On the left, there is a 'Modules' section with links to various CVS and SVN repositories. Below this, there are several attention messages and a 'Your last visit' section. The main part of the interface is a table listing files with columns for Name, Size, Rev., Modified, and Author. Below the table, there is a form for adding files to the directory, including a 'Log message' field and a 'Binary file (-kb option)' checkbox. The browser address bar at the bottom shows the URL: http://localhost:8080/subcvs_admin/index?mod=1&action=browse&dir=

| Name | Size | Rev. | Modified | Author |
|--|------|-------|---------------------|--------|
| .cvsignore | 23 | 1.1 | 2006-01-19 14:10:13 | |
| temp.log | 3684 | 1.1 | 2007-04-10 15:37:28 | |
| Demo.java | 2164 | 1.23 | 2006-12-13 14:33:19 | |
| kernel_lpsolve_LpSolve.h | 2717 | 1.1 | 2006-01-19 14:10:13 | |
| lpsolve.exp | 2856 | 1.1 | 2006-01-19 14:10:13 | |
| LpSolve.java | 4873 | 1.2 | 2006-12-13 14:36:19 | |
| lpsolve_interface.c | 6557 | 1.1 | 2006-01-19 14:10:13 | |
| makefile | 737 | 1.1 | 2006-01-19 14:10:13 | |
| test1.txt | 3430 | 1.105 | 2007-05-16 16:48:35 | |
| test2.txt | 98 | 1.27 | 2007-05-16 16:49:03 | |
| test3.txt | 43 | 1.15 | 2007-05-16 16:49:03 | |

Figure 5
Original interface of SubCVS

In the new interface the following changes were made:

- *Introduction of a logo*
A SubCVS logo was designed and is displayed at the top of each page.
- *New layout of the page*
The main menu, placed on the left, has been replaced by horizontal menu on top of the page. An additional submenu has been added on the left.
- *A new color scheme*
The new interface comes with a new color scheme using white and different tints blue and gray.
- *Main servlet has been split in multiple servlets*
The original SubCVS made use of a single servlet, processing all the requests. This servlet has now been split in multiple servlets. The main servlet has been simplified and each servlet now focuses on more distinct requests.
Reasons for this approach is to make the application more modular, abstract and reducing the workload of the main servlet by distributing it over several servlets.

The changes resulted in a new interface with not only a better look and feeling, but also faster, more reliable and better control of workflow. As the original interface, the new interface is also XHTML 1.0 strict. Figure 6 shows a screenshots of the new userinterface. Figure 7 and 8 show the modules managed by SubCVS and how a module is browsed.

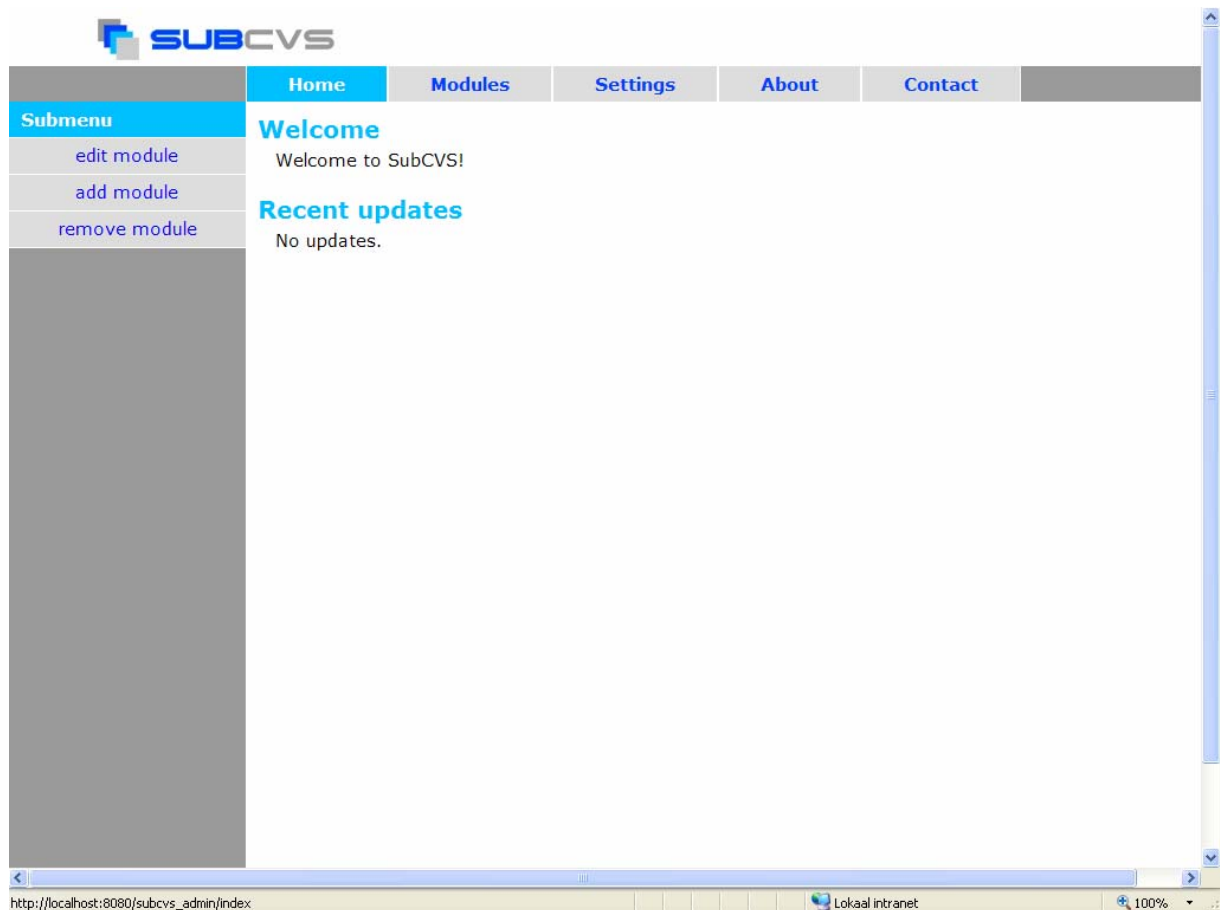


Figure 6
New homepage of SubCVS

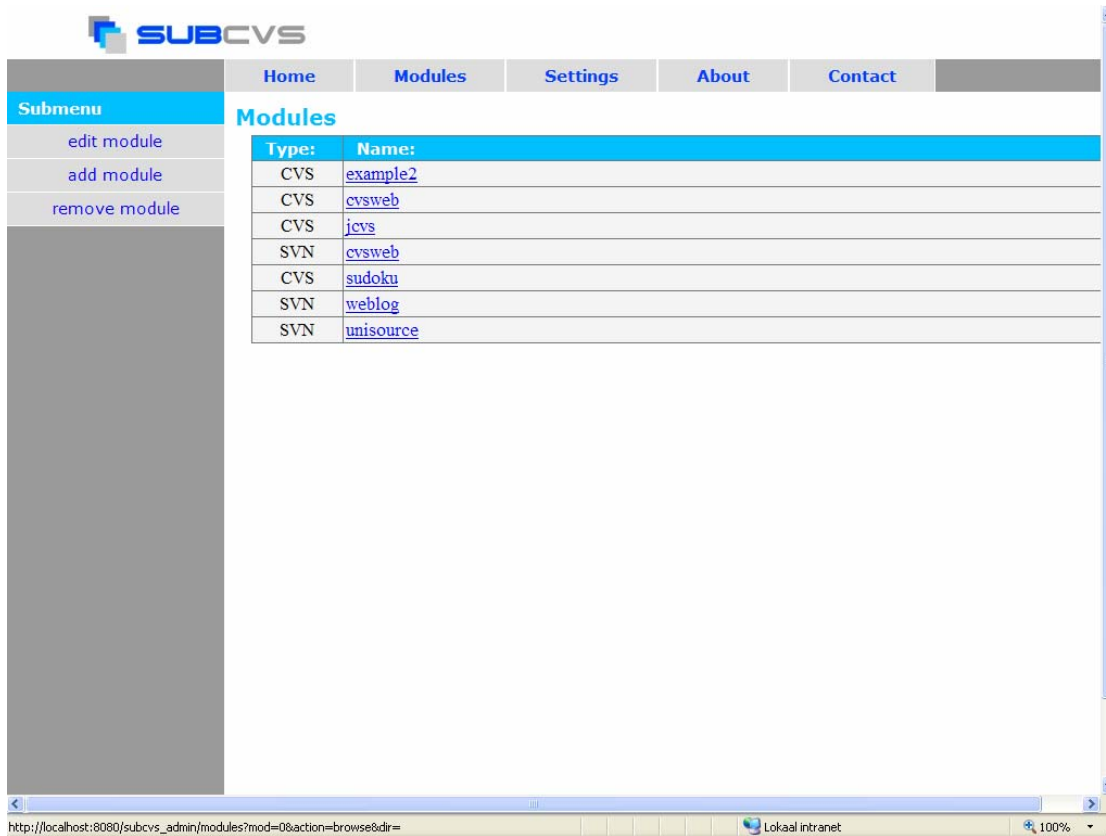


Figure 7
Modules managed by SubCVS

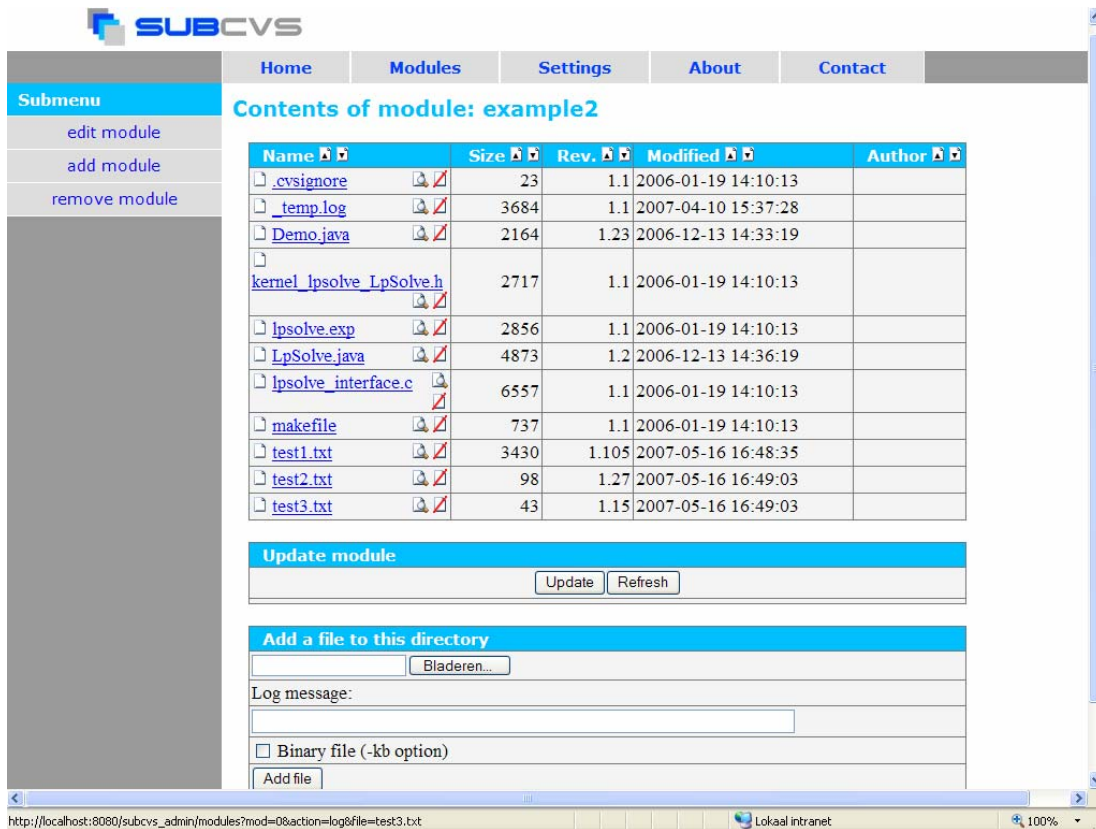


Figure 8
Browsing a module

Database

For storing information about the modules (CVS and SVN repositories) managed with SubCVS, a text file is used: modules.txt. Each line in modules.txt contains all the information SubCVS needs about a module. When a repository needs to be added to SubCVS, the user has to open modules.txt and add a line with information about the module. This has to be done in a special order and format. When the servlet initiates, or rebuilds it modules-list, it reads modules.txt and stores all the information about the repositories in a 'modules-vector'. Though this approach of storing modules is working, it is not very efficient. Much better would be the use of a database. A database has the following advantages:

- Databases can store very large numbers of records efficiently (they take up little space)
- It is very easy and quick to find information in a database
- It is easy to add new data and to edit or delete old data
- Data can be searched easily
- More than one person can access the same database at the same time
- Security may be better than storing in separate files
- Data can be taken into other applications, as other applications can easily access the database

For SubCVS it was decided to use a HSQL-database [5]. HSQLDB is a SQL relational database engine written in Java. It has a JDBC driver, fast database engine which offers both in-memory and disk-based tables, and supports embedded and server modes. It is best known for its small size, ability to execute completely in memory, its flexibility and speed. HSQLDB software is also open source and completely free to use.

For interaction with the database a DatabaseManager class was added to SubCVS. The DatabaseManager class provides simple functions to insert, remove and retrieve data from the database. Subsequently SubCVS needs a running HSQLDB server. This server can be local, but may also be deployed on another machine. Figure 9 shows a screenshot of how modules can be added to SubCVS. Next to adding modules, users can similarly edit or remove modules from SubCVS.

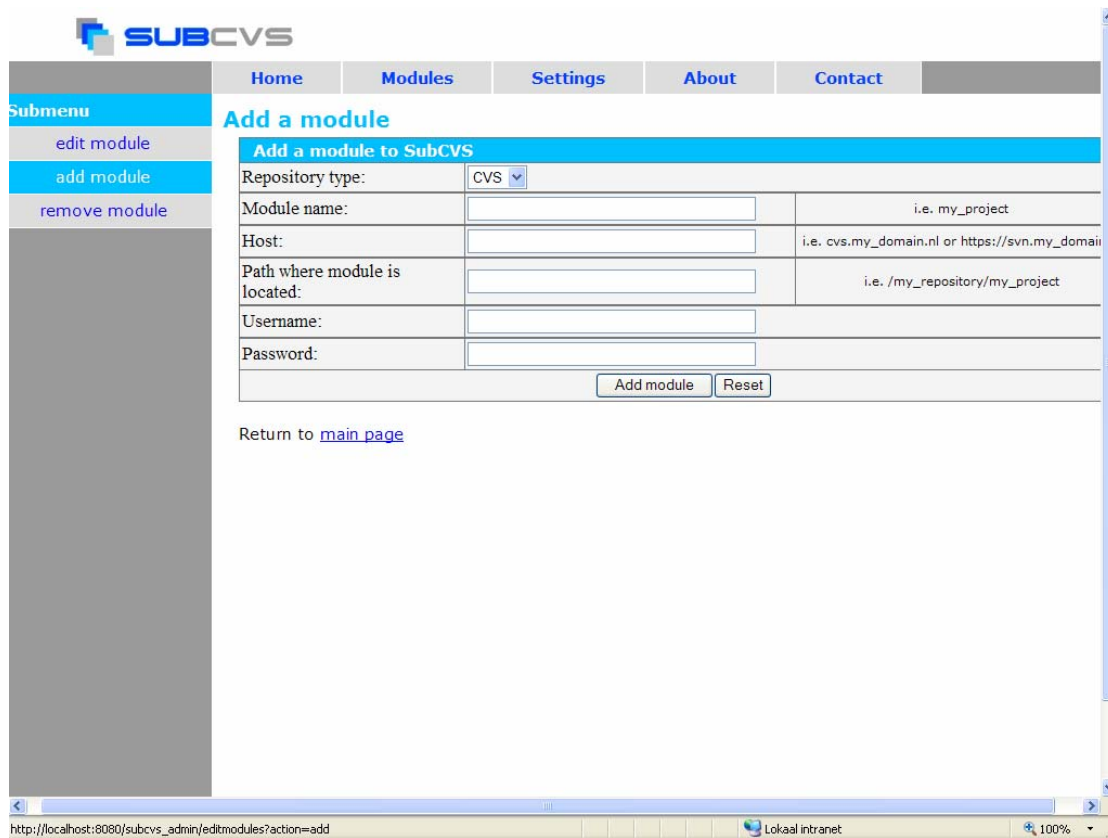


Figure 9
Adding a module to SubCVS

Refactoring & extending classes

As stated before, SubCVS originally used a single servlet to handle all requests. This servlet has been split and now multiple servlets are used, each handling a specific feature of SubCVS. SubCVS now consists of the following servlets:

- *SubCVS.java*
The main servlet, hosting the index-page of the application, the about and contact page.
- *EditModules.java*
EditModules is called when modules are added, removed or edited within SubCVS.
- *Modules.java*
This servlet is used for browsing a module, viewing the files of the module, adding/removing files from the module and updating the module (manually).
- *LoadModule.java*
The LoadModule is used on the background for AJAX interaction when updating modules from the Modules servlet. The user does not get to this servlet from SubCVS, unless the servlet is called manually. More about the LoadModule and AJAX can be found in the next section.
- *TestServlet.java*
TestServlet is a very basic servlet used to print basic information about the SubCVS. This includes information like: system properties, cvs version and svn version. Figure 10 shows a screenshot of the TestServlet.

- *SubCVSManager.java*

The SubCVSManager provides the user with the ability to change some settings of the SubCVS application. The user can start/stop the auto-updates, and change the time between the updates.

--TEST SERVLET

--System Properties:

| | |
|-------------------------------|--|
| Operating System | Windows XP |
| Operating System Version | 5.1 |
| Operating System Architecture | x86 |
| user.dir | C:\Program Files\Apache Software Foundation\Tomcat 5.5 |
| user.home | C:\Documents and Settings\AWR |
| user.name | SYSTEM |
| file.separator | \ |
| Local Root Directory | C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\subcvs_admin |
| Local Root Path | C:\Program Files\Apache Software Foundation\Tomcat 5.5\webapps\subcvs_admin\checkout |

--CVS Properties:

| | |
|-------------|---|
| CVS version | Client: Concurrent Versions System (CVS) 1.11.22 (client) Server: Concurrent Versions System (CVS) 1.12.9 (client/server) |
|-------------|---|

--SVN Properties:

| | |
|-------------|---|
| SVN version | Client: Concurrent Versions System (CVS) 1.11.22 (client) Server: Concurrent Versions System (CVS) 1.12.9 (client/server) |
|-------------|---|

--CVS checkout:

| |
|--|
| cvsw update: Updating cvswweb |
| cvsw update: Updating cvswweb/.settings |
| cvsw update: Updating cvswweb/WEB-INF |
| cvsw update: Updating cvswweb/WEB-INF/classes |
| cvsw update: Updating cvswweb/WEB-INF/classes/com |
| cvsw update: Updating cvswweb/WEB-INF/classes/com/ice |
| cvsw update: Updating cvswweb/WEB-INF/classes/com/ice/cvsc |
| cvsw update: Updating cvswweb/WEB-INF/classes/com/ice/jcvslet |
| cvsw update: Updating cvswweb/WEB-INF/classes/com/ice/jcvslet/images |
| cvsw update: Updating cvswweb/WEB-INF/classes/com/ice/util |
| cvsw update: Updating cvswweb/WEB-INF/classes/org |
| cvsw update: Updating cvswweb/WEB-INF/classes/org/ice |

Figure 10
Screenshot of the TestServlet

Next to adding and Refactoring servlets, also classes have been added, extended and refactored. These changes involve the following classes:

- *CmdExecutor.java*
Executing commands was previously done from the Modules servlet, through the function `execute()` and `executeCmd()`. These processes would call the relevant system functions, and create a process to execute the commands. This implementation was sufficient for the original SubCVS application, but with the extension made, a separate class was necessary. By creating a separate class for executing commands, we can reuse code more efficiently. Also, by doing the command executions in a separate thread, the application is more robust.
- *OutputStreamGobbler.java*
The `OutputStreamGobbler` is used in combination with `CmdExecutor` to catch outputs from the execution of commands. Previously the `StreamReader` class was used for this. The `OutputStreamGobbler` is a more extended version of the `StreamReader` (written by M. Vijfhuizen) as it also catches and handles certain commands (like updates).
- *HtmlTemplates.java*
`HtmlTemplates` is basically a library class, which holds function for printing standard HTML code which is used frequently.
- *Module.java, CVSmodule.java and SVNmodule.java*
These classes have been refactored to suit changes made to other classes and servlets. Also functions have been added to these classes that are needed for the new features of SubCVS.
- *AutoUpdater.java*
The `AutoUpdater` class is the core implementation of the auto-updates of the modules. It uses a thread to update all modules listed in the SubCVS database, after which the thread is put to sleep for a certain amount of time (2 hours by default). After the sleep, the thread awakens, and restarts this routine.

LoadModule and AJAX

As shortly described in the previous section, the LoadModule-servlet is a servlet that works on the background of SubCVS. The servlet cannot be directly accessed by users.

The LoadModule is used by Modules-servlet. The Modules-servlet is used to display the contents of a module. Users can browse through the files in the module and can add and remove files in the module. The Module-servlet provides the interface to these actions. Since the Module-servlet is such an interactive servlet, a lot of small actions and page refreshes, AJAX was used.

AJAX (Asynchronous JavaScript and XML), or Ajax, is a group of inter-related web development techniques used for creating interactive web applications. A primary characteristic is the increased responsiveness and interactivity of web pages achieved by exchanging small amounts of data with the server "behind the scenes" so that the entire web page does not have to be reloaded each time the user performs an action. This is intended to increase the web page's interactivity, speed, functionality, and usability.

The role of the LoadModule-servlet in this context is to process the request that the Module-servlet receives and return the data to the Module-servlet. For example, if a user requestst an update of the files of a module, the LoadModule-servlet is called. LoadModule then processes the request, gathers the data and sends this data back to the Module-servlet. The Module-servlet then displays this information on the screen. This way the Module-servlet does not need to refresh the whole page, but just a small section, containing the new data.

An screenshot of this action can be seen in Figure 11 and Figure 12. Figure 11 show the state before an update. Figure 12 show the screen after the update. The update does not cause a full page reload, instead, only a small section of the screen is reloaded. This sections displays the update log.

The screenshot shows the SubCVS web interface. At the top, there is a navigation bar with links for Home, Modules, Settings, About, and Contact. Below this is a submenu with options: edit module, add module, and remove module. The main content area is titled 'Contents of module: example2' and contains a table of files and folders. Below the table is an 'Update module' section with 'Update' and 'Refresh' buttons. At the bottom, there is an 'Add a file to this directory' section with a file selection button and a log message field.

| Name | Size | Rev. | Modified | Author |
|--|------|-------|---------------------|--------|
| _cvsignore | 23 | 1.1 | 2006-01-19 14:10:13 | |
| _temp.log | 3684 | 1.1 | 2007-04-10 15:37:28 | |
| Demo.java | 2164 | 1.23 | 2006-12-13 14:33:19 | |
| kernel_lpsolve_LpSolve.h | 2717 | 1.1 | 2006-01-19 14:10:13 | |
| lpsolve.exp | 2856 | 1.1 | 2006-01-19 14:10:13 | |
| LpSolve.java | 4873 | 1.2 | 2006-12-13 14:36:19 | |
| lpsolve_interface.c | 6557 | 1.1 | 2006-01-19 14:10:13 | |
| makefile | 737 | 1.1 | 2006-01-19 14:10:13 | |
| test1.txt | 3439 | 1.106 | 2008-01-19 11:51:20 | |
| test2.txt | 98 | 1.27 | 2007-05-16 16:49:03 | |
| test3.txt | 43 | 1.15 | 2007-05-16 16:49:03 | |

Figure 11 – Before an update

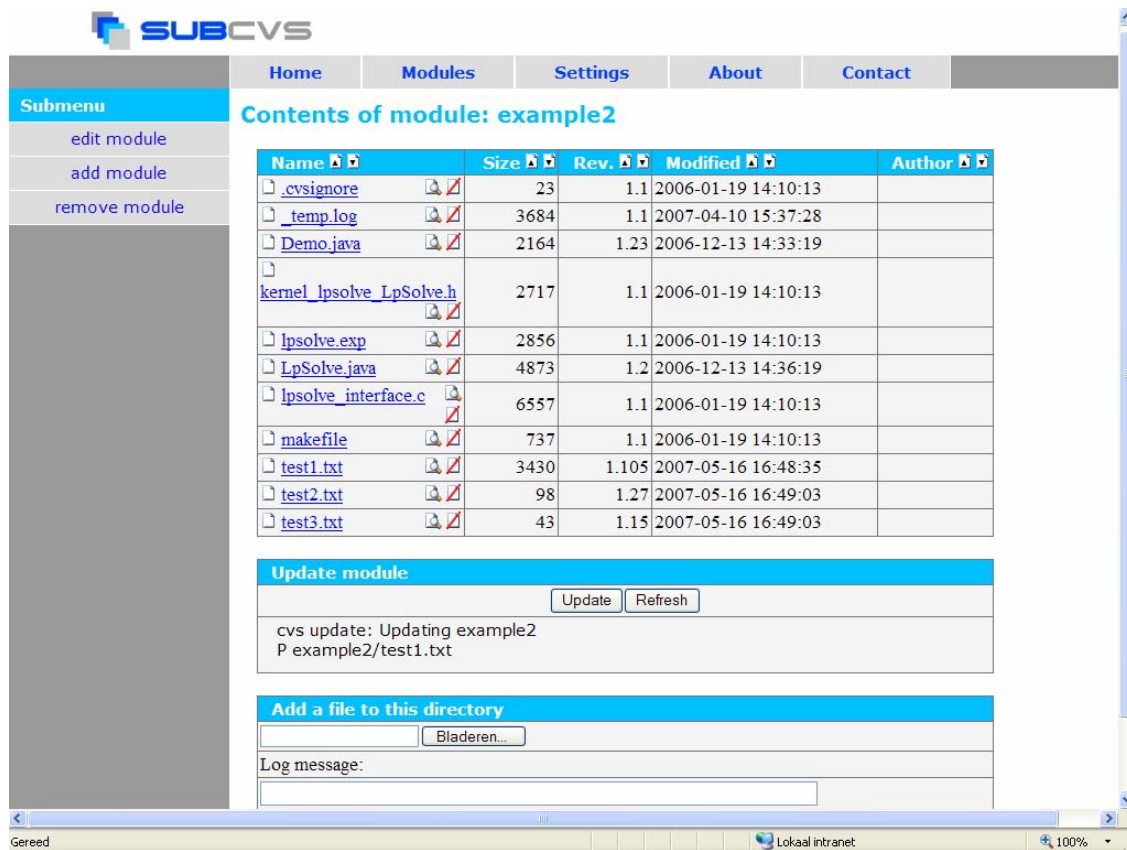


Figure 12: SubCVS after an update. Using AJAX only a small section of the screen is reloaded.

New features

Automatic checkout

Originally SubCVS only did a checkout on a module when the user explicitly requested to browse a module. When the user chooses to view the contents of a module, SubCVS would check if the module was already checked out, and if not, a checkout would take place. If the module was already checked out, an update operation would take place, to make sure the latest version of the files on the repository are displayed.

This setup is easy in use, but has as a disadvantage that it is slow. When the user requests to view a module, he has to wait for the module to checkout first. It would be easier if the modules were already checked out, and only need to be updated before it is displayed. In general an update operation is much faster than a checkout.

In the current implementation of SubCVS, the application automatically does a checkout on all modules listed in it's database when the auto-updates thread are initiated. Also, when a new module is added, it is immediately checkout. When a user requests to view the contents of a module, SubCVS now only has to do an update, so that the information can be faster presented to the user.

Though this system works, it still has situations in which it isn't faster. When the user initiates SubCVS and immediately wishes to view the contents of a module (the auto-updates thread is not running), it might be the case that that module has not been checked out yet. In this case the user still has to wait for the module to be checked out first.

Automatic updates

Next to an automatic checkout of modules, another automatization has been implemented. One of the advantages of a version control system is that a user can see when a file has been updated. Originally, SubCVS allowed the user to update modules that are managed with SubCVS, but the user had to do an update manually. What we would like SubCVS to do is to automatically do updates of the projects stored in its database, and also present these updates to the user in the form of a notification. Although this idea is very simple, its realization is a little more complicated.

When a module has been changed, it does nothing more than store these changes and keep a log. It does not send a signal to clients that are watching the repository that files have been updated. The client explicitly needs to do an update before he can see what changes are made to the repository. As it is impossible to do an update whenever an user commits changes to a repository, a different approach is to do an explicit update to all modules on a regular basis.

This approach was implemented in SubCVS. Every 2 hours (default time set) the application does an update on all of the modules listed in its database. This is done by starting a thread that does an update of all the modules, and putting the thread to sleep for 2 hours. When the thread is wakened, it does updates again, after which is goes to sleep again, and this loop restarts. The updates are done by the same thread that does the initial checkout of all the modules, it executes on the background. The thread can be described in the following steps:

- *Step 1: Checkout modules listed*
On initialization of the update thread (in the background) all modules listed in the database are checked out (see previous section).

- *Step 2: Sleep*
The thread sleeps for a certain amount of time (sleepTime, default is at 2 hours)
- *Step 3: Update*
The thread is awakened, and does an update of all the modules. If an update is found to a module, the update is stored in the database.
The thread now goes back to Step 2, ready for the next update round.

Recent updates

SubCVS now automatically does a checkout of all modules that it is managing and also updates these modules, so that the user can view the latest files when browsing a repositories. When a module is updated, and changes are found (files are edited, added, removed, etc), a current version system displays these changes. This way the user can see exactly what files are changed. What we would like is that a user that opens SubCVS would see a list of all the changes made to a the modules, and also view these changes.

To implement this, the database was extended to also temporarily store module changes. As SubCVS is constantly updating the modules that it manages, it now also catches all update reports. When a module has been changed (a file has been edited/added/removed), SubCVS catches this report during its automatic update and stores this report in the database. The changes are stored per module, so that its easier to see what changes are made to which file and in which module.

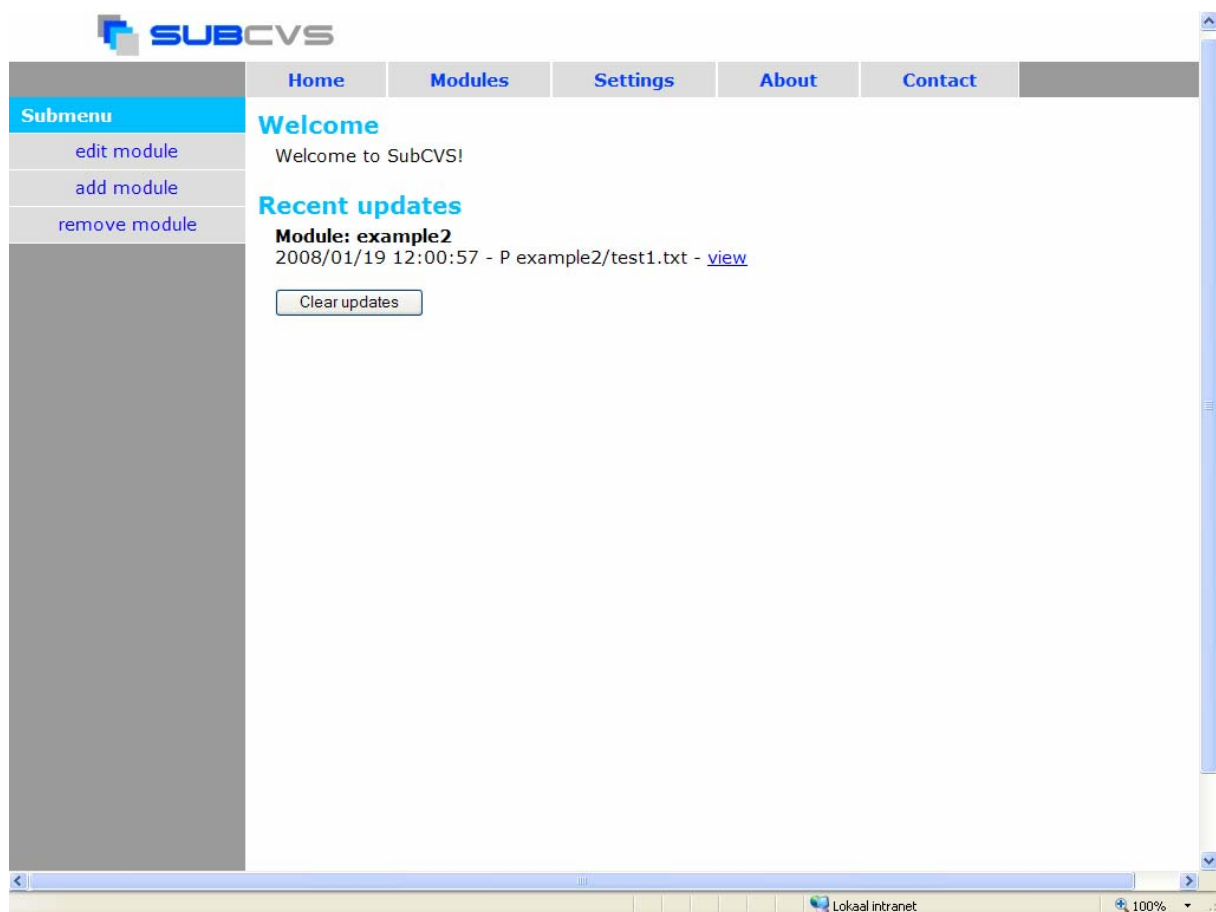


Figure 13

The automatic updater has found an update in a module. When the user opens SubCVS he can find these new updates in the Recent Updates section of the homepage.

When the user now opens SubCVS, he gets a listing of the updates that were made to the modules. Next to this notification of the updates, the user also gets a link for each update, that shows a diff between the new file and its previous version (a diff is a command that shows the differences between 2 files). Users of SubCVS can now easily see what modules were updated, and also what updates were made. After viewing the updates, the user can clear updates with a button. When the button is pressed, the database removes all of the update reports that it has stored. This makes sure that next time the user opens SubCVS, he only gets reports of newer updates.

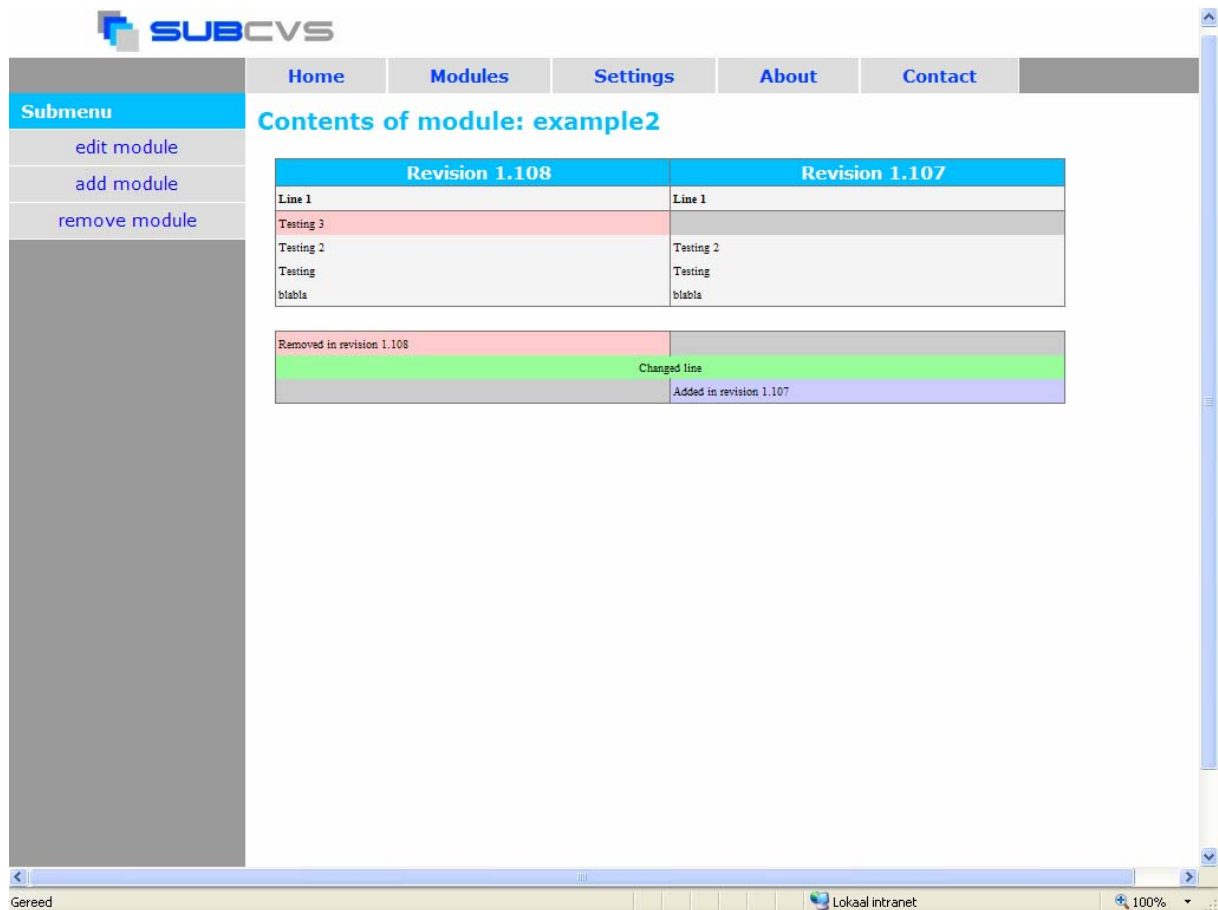


Figure 14

When the users clicks on 'view' on a recent update, the changes made to the file compared to the previous version of the file are shown.

Module Management

As stated before, SubCVS now offers the user an easy interface to add, edit and remove modules that are managed under SubCVS. The user does not need to change a text file anymore. When adding a module, the user simple has to fill in a form with the required information, and the new module is immediately added to SubCVS. This is also to case for editing and removing modules.

As an extra feature, SubCVS also checks if a newly added module exists. When the user adds a module (by filling in the form with the appropriate data), SubCVS immediately does a checkout on that module. If an error is received from the server during the checkout, the module is not added to the SubCVS database and the user is informed about the problem. If the module was correctly updated, the user receives a confirmation. A secondary advantage of this method is that the new module is now already checked out, making it faster to update & browse.

SubCVS Manager

The automatic updates are done through a thread that is constantly updating (or sleeping). One can imagine the situation that the user needs to stop this thread. In another situation it might be the case that the user wishes to change the sleep time between updates. Or, maybe the user does not wish to wait for the next update round of the automatic updates thread and wishes to see the updates immediately.

For these cases, the user can make use of the SubCVS Manager. The SubCVS Manager is a servlet that can actually start and stop the automatic updates thread. When SubCVS is initiated, the automatic updates thread is not running. The user has to start it from within this manager servlet. If the user wishes to stop the thread, he can also achieve this from this manager servlet.

The SubCVS Manager servlet also provides the user the means to alter the sleep time of the automatic updates thread. As default a sleep time of 2 hours is used. Using this servlet the user can change this value to his own wish.

As a third feature, the user can also request for a immediate update of all the modules under SubCVS. This way the user can see all the changes made to the modules from the last update round of the automatic updates. By doing this immediate update, the automatic updates thread is also reset. This means that after this immediate update, the automatic updates thread is set to sleep again for the given sleep time. Its routine has been reset. A screenshot of the SubCVS Manager can be foun in Figure 15.

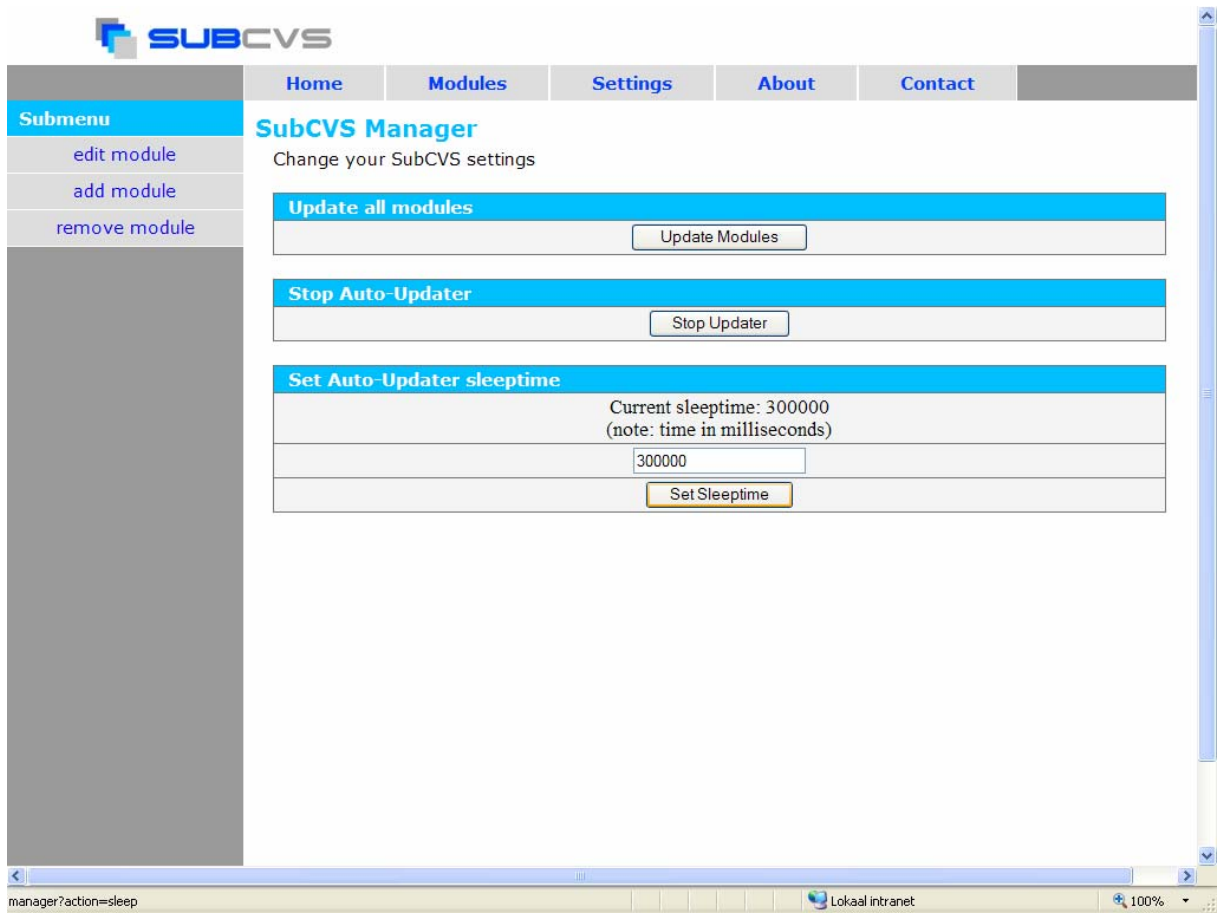


Figure 15
The SubCVS manager

Casestudy

Like the original SubCVS application, this version was also tested. Using different CVS and Subversion repositories from projects within LIACS, the original features and new features of SubCVS were tested.

To make use of SubCVS, the user needs the following applications:

- SubCVS deployed
- CVS
- SVN
- HSQLDB server

The database can be empty when SubCVS is initialized, but if it is already filled by default with some modules, SubCVS will automatically load these.

When SubCVS is initialized, the user can start the automatic updates thread from the SubCVS manager. When the automatic updates start, all modules that are found (from the database) are checked out. This happens in a thread on the background. The user does not see this. This process also does not cause any pause or stall as it is executed in the background. The user can continue without even noticing the checkouts.

After the checkout of the modules is complete, the application continues to update all the modules looking for updates. This happens every 2 hours, but this 'sleepTime' can be changed in the SubcvsManager. If the user is to quit SubCVS, the servlet will continue checking for updates. When SubCVS is opened again, the user will get a notice of the updates that were made to projects. The user can also click on the updates to see what differences there are with the previous version of the file that has been updated.

Modules can be added, deleted and edited. When a change is made to a module, the update-thread (responsible for automatic updates) will be reset, thus taking the changes made to the modules into account. Next to these new features, the user can still browse and update repositories, add and delete files and view files.

With every request that the user makes:

- An appropriate command for that action/repository is build
Classes in action: CVSModule, SVNModule and Module
- That command is executed on the background and the output is captured
Classes in action: CmdExecutor and OutputStreamGobbler
- The output is presented to the user
Classes in action: HtmlTemplate and the appropriate servlet

SubCVS was tested with multiple repositories and over length of time. It handled its original functions & request correctly, and didn't have problems handling the new features. It's become an ever simpler, practical and easy to use management tool for CVS and SVN repositories, all through a web-browser.

Conclusion

As the goal of this project was to improve and extend the functionality of SubCVS, we can now safely say that these goals have been accomplished. SubCVS already was a first in its kind, a servlet-based web application to browse repositories with support for multiple version control systems, its now even more than that.

As the original SubCVS was a welcome addition to the set of tools of an software engineer, SubCVS now provides even more quality and features, making it even more practical and intuitive.

Although SVN and CVS are two different versioning systems, SubCVS has proved that is quite possible to create an application that can manage modules from both versioning systems. Users of SubCVS can easily do the basic operations on their modules, without having to think whether they're working on a CVS or SVN repository. Users can simply issue commands to SubCVS, which in the background executes the specific commands for each versioning system.

SubCVS is a good tool for managing CVS and SVN repositories. And with its new features might even come in handy when using only CVS or only SVN, as it now also saves users time by automatically showing project updates.

Discussion

During this project several enhancements and new features were added to SubCVS. However, its quite clear that SubCVS has a lot more potential. Additional features could be added to make SubCVS more complete. Some features one might consider:

- Support for other version control systems
- Give user ability to edit and commit files
- Possibility to add, delete and change directories
- A project history viewer
- Introduce user-accounts, making it possible to having more control over who viewer of a project
- Adding a command-line interface, thus giving a user to also execute commands manually (in case of special operations)
- More use of AJAX interaction

During the development of SubCVS several issue and problems arose. These were mostly small and we're easily fixed. However, there is still one problem which has not been resolved. When executing an operation on a SVN repository, the AJAX interaction seems to fail and does not correctly return the data. Although CVS and SVN operations are handled exactly the same way, and the correct data is send by the LoadModule-servlet to the Module-servlet, the Module-servlet does not receive any data from the LoadModule-servlet in the case of a SVN request.

What causes this problem and how it can be resolved is not yet known. It is most probably issue that is caused by the platform on which SubCVS is used (platform used at Liacs is Linux). On different platforms (non-Linux environment), the problem does not occur. Note that this issue only concerns the display of the log data. This means that if a user for example adds a file to a module, the file is correctly added to the module, but the display of the log message which states that the file has successfully been added fails to display.

References

- [1] CVS, Concurrent Versions System - <http://www.nongnu.org/cvs/>
- [2] SVN, Subversion - <http://subversion.tigris.org>
- [3] LIACS, Leiden Institute of Advanced Computer Science - <http://www.liacs.nl>
- [4] M. Vijfhuize - *SubCVS: merging version control systems in a servlet-based repository browser (2007)*
- [5] HSQLDB - www.hsqldb.org
- [6] Apache Tomcat - <http://tomcat.apache.org>