Position Paper

Distributed Software Development using Jini

Jörg Niere, Matthias Gehrke
Department of Mathematics and Computer Science
University of Paderborn
Warburger Str. 100
D-33098 Paderborn
Germany
[nierej|mgehrke]@uni-paderborn.de

Albert Zündorf
Institute for Software
Technical University of Braunschweig
Gaußstr. 11
D-38023 Braunschweig
Germany
zuendorf@ips.cs.tu-bs.de

Keywords
software development process, team software development, distributed software engineering

1 INTRODUCTION

Any team of software developers has to coordinate itself with respect to task scheduling and access to common documents, e.g. to the source code. Groupware and workflow management systems offer solutions for task scheduling and process enactment [Dev]. Concurrent access to common system documents, like the source code may be coordinated by using version and configuration management systems (VCMS) like SCCS, RCS, or ClearCase. In co-located teams, these VCMSs frequently employ pessimistic locking techniques to coordinate the access to common documents. This means, that only one team member obtains a writeable copy of a source file. Other team members have to wait until the document owner releases his/her lock or they have to communicate their change requests to the document owner.

With the worldwide growing business of the software-industry comes the need of more flexibility and adaptability for single developers, respectively for whole developer-teams. A software-developer who works for a big company and develops a software-system with other team members in several other countries all over the world, has to coordinate his/her results with the members of his/her group.

Existing approaches do not work for such distributed software development over the internet. First, most groupware and workflow systems do not work well over the internet. In addition, since the team is distributed and cannot meet on a regular basis for coordination purposes, an even more sophisticated groupware and workflow system is required, cf. [JSZ97]. Second, strict locking techniques do not work for distributed software development teams. In a distributed team, it might become difficult to contact the owner of some documents because he/she may have different working hours or he/she is just not on-line at the moment.

This paper proposes a distributed software development process based on (1) a VCM system with optimistic locking concepts (i.e. CVS) and on (2) a tight integration of VCM and workflow system and on (3) special project access software that allows to work off-line most of the time and that coordinates document access and workflow update during on-line update sessions.

2 MAIL BASED WORKFLOW ENGINE

Integrating workflow in the development process and combining it with VCM access allows a better support for software engineers participating in several projects. With such a coupling, information given to the developer can be more specific. As an example Figure 1 shows the software developing process that has been used in our department in several projects.

Since our development teams are not co-located, we use an optimistic locking concept provided by the CVS-System [CVS]. The upper part of Figure 1 shows the original CVS-System. CVS supports highly distributed software development with version control and automatic merge algorithms. Each developer in the team has a local writeable copy of the software and is able to work off-line when there is no internet access, temporarily. CVS manages a central repository and local copies can be synchronized whenever needed. Synchronizing a local copy with the repository is done by calculating deltas and storing them in the repository.

If one team member wants to upload his/her changes on the software (author), he/she sends a so called ‘checkin’ command to the CVS-System together with some workflow data. First, a pretty printer, initiated by CVS, runs over all source code files in the local workspace. This ensures a standardized formatting and preserves indentation changes in the source code caused by different editor tools. For example, one developer uses tabs to indent the source and another one uses spaces. Without pretty printing, opening a file in an editor using spaces whereas the file has last been edited using tabs, means that all lines will be marked as changed without really changing a line. After pretty printing all source code files, the CVS-System calculates the deltas and sends a notification mail to each team member. Those members can now download the changes sending an ‘update’ command. In addition, the CVS-System can be configured, e.g. to send notification mails to certain team members, subscribed for the modified application parts.

In order to provide a better workflow process, we have extended the original CVS-System by an email based review workflow engine called ‘distributed software development clerk’ (DSD), shown in the lower part of Figure 1. The review process is initiated by collecting the additional workflow data provided by the author. The additional workflow data and the delta produced by the CVS is sent to the DSD clerk via email. The delta includes all changed, added, or deleted lines of code including a context range of

40 lines. Workflow data might be a reference to the fixed bug, a todo item number, status information, or the time working on the project to finish a task.

According to extreme-programming and the 4-eye principle, all code uploaded into the repository has to be reviewed in order to detect certain defects, early, to achieve common understanding, and to guarantee common coding and documentation standards. Therefore the DSD clerk sends an email to another developer of the team. For fairness reasons, the reviewers are managed as a round-robin list. Since the team is working distributed by not working at the same time, the reviewer can make the review during the next week. (The deadline stems from our practical experiences.) Has a developer reviews outstanding over one week (excepting e.g. vacations), he/she is not able to upload his/her changes into the repository until the review(s) are done. This forces a permanent review process and avoids a little bit the situation that a code to be reviewed is changed before the review is finished.

Reviewing itself means that the reviewer has to look over the changed lines of code and checks if all points in the provided style guide are fulfilled. This checklist contains items like english as project language, documenting each class and method, using understandable variable names, and implementation guides for design elements, e.g. associations between classes.

When a team member has done a review it is send back to the DSD clerk via email and the clerk forwards the mail to the original author. The mail is not directly send to the author, in order to support an anonymous review process. This means that all identification possibilities are left out of the review mail in order to avoid personal conflicts between team members. Finally, the original author is responsible to integrate the desired changes of the reviewer. The developer does not have to justify his reaction to the reviewer. This avoids an endless cycle of reviews and discussions of different solutions preferred by developers, and leads to a better and more productive work climate.

Whether the author decides to integrate the notes of a reviewer or not, the review process including all review mails are stored in a separate database. This allows to track hindsight, whether the reviewer has not seen a problematic part in the review or the author has not reacted on the notes. When an error occurs in a certain part of the software, the author as well as the reviewer gets an email, automatically. This creates a self improving process.

We also store statistic data for each checkin, e.g. author, lines changed, lines added, lines deleted, time used etc. The statistic data can be evaluated during the process and may lead to project management actions to hold for example the next release shipping deadline. Also the team leader (project manager) may get an email to see the progress of the project and to be able to intervene whenever needed.

PRACTICAL EXPERIENCES

As mentioned before, we are using the combined CVS/DSD-System in several projects. Students at our university have to perform a practical training which is a little software engineering project (10 kLOC) done in groups by 8-10

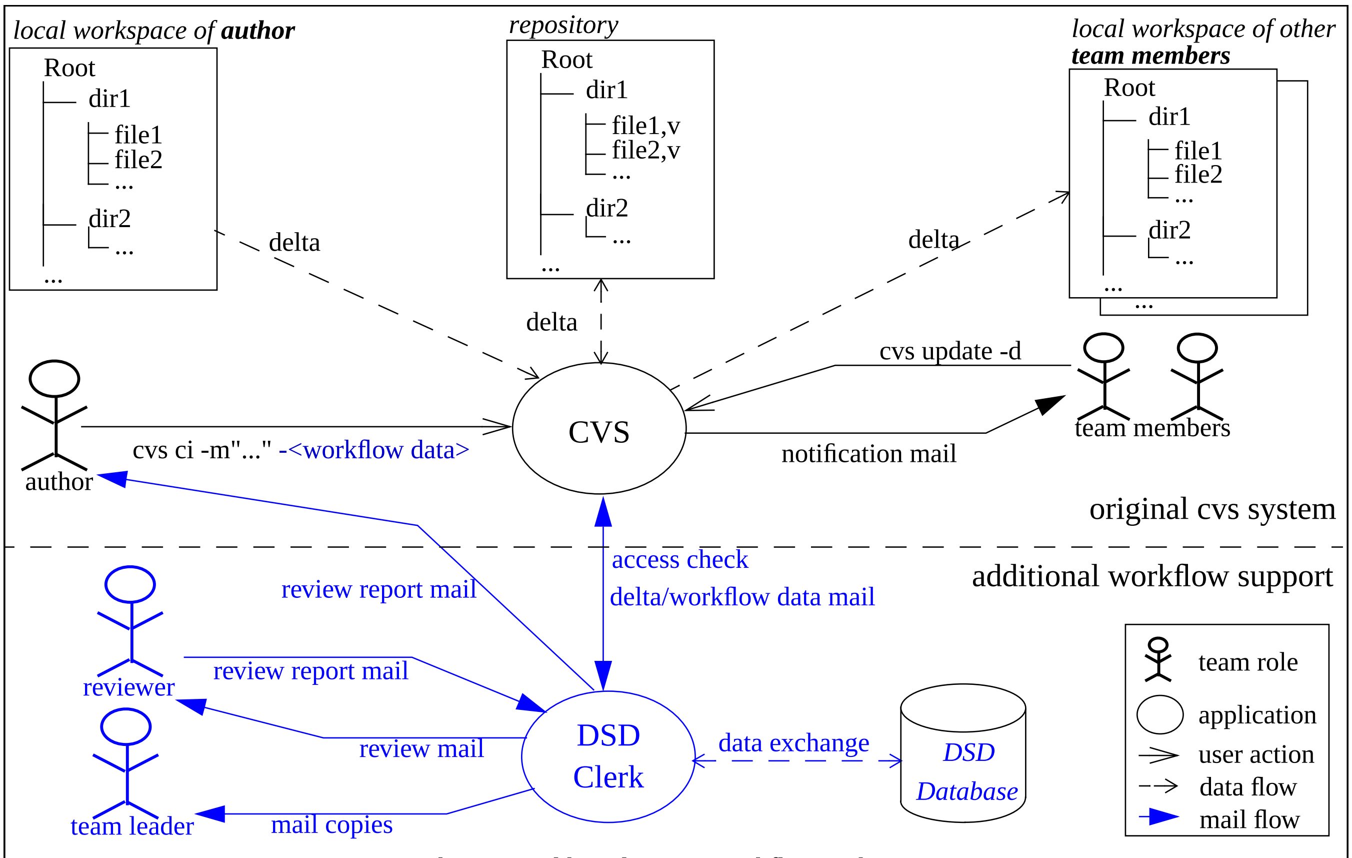

Figure 1 Combining mail based review workflow with repository access

students. Each group has its own repository starting with a legacy system in it and each student has equal rights as author and reviewer. A layer project where we are using the CVS/DSD-System is our Fujaba environment. Fujaba is an UML case tool written in 100% pure Java. The project has over 470.000 LOC developed by a team of 10 to 20 persons (PhD, graduated students) over the last three years.

Overall, the experiences using the CVS/DSD-System are very good in the beginning, novice developers have some objections resulting from the different kind of development and from the effort to make reviews. However, the positive results preponderate. The produced source code has a high quality, is well understandable, easy to maintain, and thereby easily reusable. The reviewing process leads also to a common coding standard, less code duplication in the project, and in addition, it trains the developers. The training leads to a high education of the developers because reviewing is also a kind of learning from other developer's experiences. Finally, the review process facilitates the integration of new developers.

In addition, we started the Fujaba project as a local development team and since we are publishing the environment and invite users all over the world to join our development team, the CVS/DSD-System has shown good experiences. Especially the coupling of the repository and the workflow to integrate novices is very effective although the developers have completely different knowledge and experiences in programming.

Although we have made very good experiences with the current process there are some problems concerning the mail based workflow. Typically, mails are unstructured and mails can also be very different, e.g. plain text mails and html format mails. Parsing them in an adequate structured format is very hard. In addition, mails get lost, mails are not recognized as workflow mails, mail servers have down times, etc. All these problems can disturb or destroy (parts of) the workflow and can not be handled in an appropriate manner.

3 COMMON PROJECT ENVIRONMENT WITH JINI

As described in the previous chapter, our approach with the mail-based CVS/DSD-System is a good solution and helps the developers to cooperate, coordinate and integrate their source code very easily. However, with the beginning of distributed development over the internet it becomes more and more inoperative. As our developers started to develop at home, used the internet to connect to the server, and used different tools (e.g. other mail-tools) for developing, they got several new problems and we had to find a better solution.

One possible way is a server-based internet application, accessible with a standard internet-browser. In this case it is possible to store and retrieve important data via the internet. For every user the server creates a personalized task list which e.g. contains the reviews he/she has to do. The information is presented in a normalized manner and the developer is supported with standard browser-based interfaces. With this approach most problems using the email-based solution can be solved.

Unfortunately, a web-based solution inherits other difficulties. It is necessary to be connected to the internet to do the work, e.g. to make the reviews or read new tasks. Furthermore the network infrastructure (internet) itself contains problems concerning reliability and availability. Therefore it is not possible to connect to the internet at every time (connect to an internet-provider) and make adjustments. Another problem is the use of different kinds of locally installed applications or different kinds of browsers. These browsers support different kinds of HTML-, jscript-, vbscript or Java-Versions, which sometimes leads to completely different results or even makes it impossible to work with the system. Finally, this makes it very difficult to implement the needed application with internet technologies, because the possibilities for web-based user interfaces are not sufficient enough to realize all needed functionality. These problems lead us to the deliberation that it is mandatory, that every developer uses the same applications to communicate and work with the project management system.

To overcome the problems mentioned above we have decided to implement a stand-alone application, which on the one hand supports all CVS-functionality and on the other hand supports additional functionality like workflow for reviews, bugtracking or statistic analysis. We call this the "Distributed Software Development Client" (DSD-Client). At this point we have to observe another difficult situation, because in this solution it becomes necessary to install, update and maintain the DSD-Client software on every local machine of every developer. This is a very expensive and time spending job including all update problems. These problems reach from the distribution of different lingos of the DSD-Client to different versions on different operating systems. To solve these problems we use the Jini-technology.

Jini© [EDW00] is a relatively new architecture from SUN Microsystems® and is completely implemented in Java©. It allows different kinds of programs to connect spontaneously and use services from other programs. Therefore, a program *asks* the net for a special kind of service and will get a receipt if that service is available, without the knowledge where the service is hosted. This architecture reveals a very good solution for reliability and scalability, because every Jini-service can be started several times on many different machines.

In our approach (Figure 2) we split the original CVS into many small modules to make the application easier to maintain and easier to extend (extended-CVS-Client). All CVS-commands, like add, diff, checkin, etc. are separated into single modules where each module is implemented in the Java Intelligent Network Infrastructure (Jini) and becomes a single Jini-service.

In our solution we have implemented a very thin client (DSD-Client) which seeks and executes Jini-services. This locally stored client connects to the system and downloads the corresponding Jini-service (e.g. CVS-"checkin"-service). This situation is illustrated in Figure 2. The Client seeks for an available Jini-service, downloads and executes it. In addition, we have added other Jini-services to the CVS-Jini-Services to support further functionality. One extension is the implementation of a Jini-service, that connects to a relational database and stores additional data for statistical or reviewing reasons. This Jini-service can be automatically

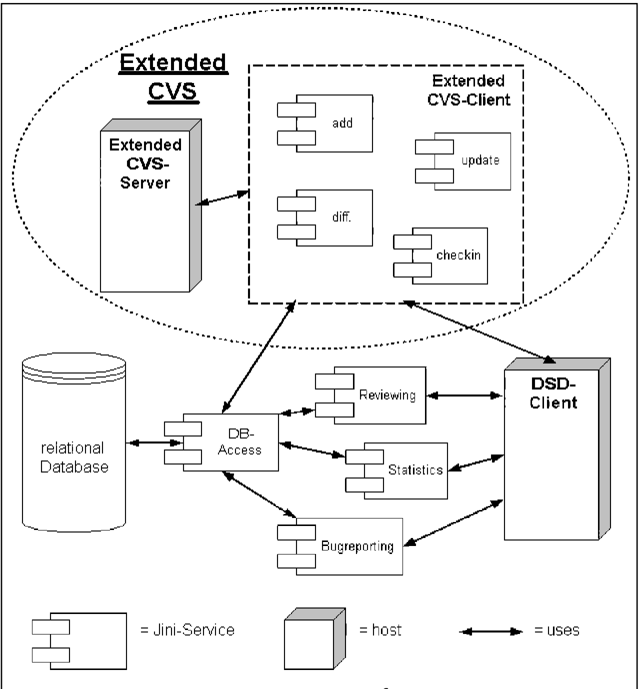

Figure 2 Jini service infrastructure

used by other Jini-services (e.g. "CVS-checkin"-service) to store information. The stored information can now be extracted, separated and used to initiate an automatic reviewing process, which is used e.g. to control the correctness of new or modified source code. One great benefit for the use of the Jini-technology is, that the distribution of a new software is not necessary, because the needed classes are downloaded on demand and the user gets the current version at every time. When a developer wants to submit his/her changes he/she connects to the internet, activates his/her DSD-Client and transfers the changes, where the DSD-Client ensures that the current version of the corresponding Jini-Service will be used.

In the last paragraph, we described a solution to handle the update problem. In this solution the developer has to be connected to the internet (on-line) to do his job. This is not satisfying, because the developer must be able to do his job even when no connection exists (off-line). At least, he/she should be able to view and update his/her task lists. In this case it must be possible to store the needed programs (Jini-Services downloaded by DSD-Client) and needed data on the local machine. After the developer finished his/her work, he/she establishes a connection to the internet and transfers the results back to the server.

To reduce network traffic, it is possible to implement an automatic caching/storing mechanism to allow off-line-work and reduce on-line-costs. Therefore, the *ClassLoading mechanism* of Java can be extended, that the user gets the possibility to store a version of the Jini-services (in DSD-Client) on his/her local machine preventing to load the whole Jini-Service every time he works with the system. Once a connection is established, the system verifies the versions of the Jini-Services on the DSD-Client with the actual version available on the net and transfers differences from the net to the DSD-Client. After verifying the versions of the DSD-Client, the locally stored data will be verified, too, and the differences will be exchanged.

This data can be the results of the reviewing process or other data like to-do items or bug reports. For example if an application crashes the thrown exception is analysed and the corresponding line of code is determined. Then the responsible author of that line is retrieved and the bug report is automatically forwarded to him. With this solution it is possible to route a bug report directly to the developer and initiate a workflow to solve this bug and inform the bug reporter when the bug was fixed.

In addition, it is possible to store data for statistical purposes. This may be the amount of modified source code during a given week, the amount of received/solved bug reports during the last year, or the date the last time a given developer has checked-in his new source code.

4 CONCLUSIONS AND FUTURE WORK

Team coordination based on CVS is common practice in many distributed and open source software development projects. Examples for a workflow management system that works well for large distributed development teams are bug tracking systems like BugZilla [Pro], also used in many open source projects. However, an integration of CVS and a workflow management system allows a better support for the developers in the team but is not yet state-of-the art.

This work proposes such an integration based on light-weight project access systems exploiting Jini technology. The proposed CVS/DSD-System based on Jini services facilitates project monitoring and keeps all team members up-to-date. Due to the established reviewing process, we have achieved high quality coding standards. Common design issues are easily enforced by adding them to the review check lists.

The development of the Jini based project access system to support a common working environment for each developer is current work. However, Jini access to the CVS/DSD-System is done. Integrating a caching mechanism for Jini services to enable off-line work and reduce network traffic needs to be completed. Jini services for reviewing, agenda control and workflow support are still under development.

For the Fujaba environment see http://www.fujaba.de/

REFERENCES

[CVS] CVS. *Concurrent Versions System - The open standard for version control*. http://www.cvshome.org/.

[Dev] Lotus Development. *Domino Workflow - Buisiness Processes*. http://www.lotus.com/world/germany.nsf/_/Produkte.

[Edw00] W.K. Edwards. *Core-Jini*. Sun Microsystems Press - Java Series. Prentice Hall, 2000.

[JSZ97] J.H. Jahnke, W. Schäfer, and A. Zündorf. *An experiment in building a light-weight process-centered environment supporting team software processes*. Software Process Improvement and Practice, 3(3), John Wiley & Sons, 1997.

[Pro] Mozilla.org Project. *BugZilla "A bug tracking system"*. http://bugzilla.mozilla.org/.