Increasing Awareness in Distributed Software Development Workspaces

Marco A.S. Mangan[1,2], Marcos R.S. Borges[3], and Claudia M.L. Werner[1]

[1] Programa de Engenharia de Sistemas e Computação – COPPE/UFRJ, Brazil
{mangan,werner}@cos.ufrj.br
http://www.cos.ufrj.br/~odyssey
[2] Faculdade de Informática/PUCRS, Brazil
mangan@inf.pucrs.br
[3] Núcleo de Computação Eletrônica and Instituto de Matemática/UFRJ, Brazil
mborges@nce.ufrj.br

Abstract. This work presents a middleware for collaborative applications that increase product and workspace awareness information available to users of computer-aided software engineering tools. This middleware-based approach helps application developers to construct enhanced tools, adapted to specific needs, reusing software components and existing applications. These enhanced tools must be designed to overcome some of the technical difficulties of collaboration in distributed software development scenarios, like the need of monitoring changes in remote workspaces. This paper describes the middleware architecture and intended usage, presents examples of enhanced tools, and proposes future case studies.

1 Introduction

In the last decades, many organizations have adopted remotely located facilities and outsourcing in software production. Global Software Development (GSD) must deal with the strategic, technical and cultural issues of participants and teams dispersed over time and physically distant [5]. Distributed Software Development Environments (DSDEs) try to provide software developers with facilities that help to overcome some of the difficulties imposed by the separation over time and distance. For instance, communication and awareness breakdown inside virtual development teams are difficulties that are dealt with such environments. Collaboration facilities range from shared repositories, on-line and off-line communication channels to coordination and awareness mechanisms.

This work deals with the difficulties of constructing such environments, and, in particular, on obtaining and managing awareness information. The most challenging problems are that the enhanced environment should provide (a) adequate support for software development activities and, conversely, (b) useful awareness information for software developers. We propose a middleware-based approach that enables the creation of shared workspaces on top of pre-existent software

G.-J. de Vreede et al. (Eds.): CRIWG 2004, LNCS 3198, pp. 84–91, 2004.

tools and groupware. Team participants must assess tools and the correspondent application events they want to share before the collaboration activities take place.

Once configured, the middleware is able to provide services that are similar to those found in groupware applications. Depending on the degree of modification of the tool or of its execution environment, and the participant's objectives, it is possible to provide a set of collaboration services. These range from simple presence information to tracking of development history, product and workspace awareness information, and even collaborative editing.

Our assumption is that collaboration can be enhanced with small modifications in real working environments. Information gathered from the environment can be used to identify opportunities for collaboration that participants may not be aware of.

The remainder of this article presents an overview of the class of collaborative environments (Sec. 2) we plan to develop by using the proposed middleware architecture (Sec. 3). We summarize roles and a process to develop awareness enhancements in Computer-Aided Software Engineering (CASE) tools (Sec. 4) and also present some examples of enhancements (Sec. 5). Finally, we present the conclusion of this article and the next steps in this ongoing work (Sec. 6).

2 Distributed Software Development Environments

Distributed Software Development requires the coordination among teams and individuals dispersed over time or physical distance. Herbsleb et al. [5] believe that we have to understand distribution as a more ample term. Even colleagues that work a few rooms down the corridor or in different floors of the same building may suffer from some sort of breakdown in communication, coordination, and awareness that are characteristic of global and distributed software development scenarios.

Recently, some software development environments were proposed to support GSD [2–4, 6, 7, 1]. These environments aim to combine software development task support with some sort of teamwork support. Altmann and Pomberger's [2] ample survey of cooperative software development practices lead to a proposal of an environment aimed at large software projects organizational support. Distributed process coordination is present at Serendipity [4] component-based environment. Palantir [1] and Gossip [3] propose environment enhancements based on improved product awareness information over artifact changes. Real-time, collaborative editing support is applied on distributed extreme programming in both MILOS [6] and Tukan [7] environments.

This diversity of approaches in CSCW support and software development scenarios is an indicator that GSD support is a complex problem. Software development task support must be fitted to meet particularities in software development process, technology, and methodology. It is not economically viable, if not virtually impossible, to provide a different environment for every particular combination of these factors. In practice, these collaborative environments offer

software development tools that are not as effective as their single-user counterparts. As a result, these environments neither evolve to follow new trends in software development nor are commercially supported. Clearly, a successful DSDE must support high levels of customization and be built on top of existent, evolving, supported tools and applications.

DSDEs are usually developed using one of two approaches. In the first one, the starting point is a working software development environment on top of which some enhancements are produced to create a more collaborative environment. Typically a collaborative library or toolkit is adopted, e.g., Tukan is developed over a Smalltalk environment and COAST [7]. In the second approach, the starting point is a distributed programming platform (e.g message passing, distributed objects, or shared memory) or a collaborative library. The software engineering support is completely programmed from the scratch, e.g., MILOS. Both approaches require a large programming effort. We propose a third approach, based on the existence of a collaborative middleware.

3 Middleware Architecture

The proposed middleware architecture (Fig. 1) is composed of three types of elements connected by an event notification service. The first element is the CASE tool, or application, that supports a specific set of operations and deals with end-user input and output. The CASE tool provides a rich context for software engineering activities that is explored as a source of awareness information.

The second element, the collaborative extension, also provides end-users with input and output. Mainly, the extension provides some visualization about events in remote workspaces. We have successfully implemented collaborative extensions that provide diverse awareness information: telepointers, radar-views, group and task awareness widgets.

The third element, the collector, monitors operational events in the CASE tool and its run-time events. The collector is activated by some application operation, captures data from the application current state, and places new objects in the event notification system. The collector should be a simple program, so that it should be easy to maintain and develop. We have successfully implemented collectors using different approaches: monitoring the application windowing system information, input devices, and persistence mechanisms, and also using tool extension mechanisms and aspect-oriented techniques [8].

The event notification system offers storage, distribution, and query facilities. The current implementation is based on tuple spaces, a distributed programming paradigm [10]. A high-performance tuple space server [14] can hold a large number of independent objects (called tuples) and offers three basic primitives: write, read, and take. The write primitive places a new tuple in the tuple space. Read and take are query primitives that receive a query template as a parameter and return a tuple from the space that satisfies the template. Take is a destructive query, which means that the tuple is removed from the space.

We summarize the flow of information in the architecture through a description of four supported activities: capture, secure, analyze, and distribute

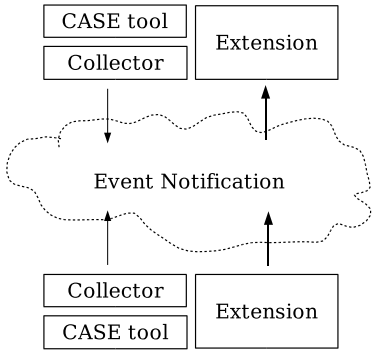


Fig. 1. Middleware architecture.

awareness information events. Since the CASE tools chosen by the team participants already define the work environment, the main problem is how to capture the necessary application events. Current DSDEs are programmed specifically to provide points of capture for the necessary events. In order to provide loose coupling of applications, the architecture use collectors, software programs that are programmed to collect significant application and environment events. Collectors are activated by subscription of one or more collaborative extensions. The selection of collaborative extensions requires the existence of appropriated collectors. On the other hand, the availability of collectors constrains the applicability of extensions to a given CASE tool.

An event is defined by the awareness information need of some extension element. Events can range from simple input and output activity in the end-user workstation to more elaborate information about operations over the objects present in the workspace. For instance, a real-time shared workspace extension may provide a telepointer implementation that relies on mouse move events collected from the CASE tool run-time environment.

The main concern in this architecture proposal is how to program collectors without interfering with the application perceived performance. A sensor may not be able to capture accurately every relevant event and its programming can be a challenging task, even if the application source code is available. On the other hand, we assume that risk because programming sensors is less complex than programming a new DSDE.

Event occurrences are expected to be very high. A database is assumed to secure all event occurrences for posterior analysis. On a distributed setting, more than one database may be necessary for different groups and organizations to control their own data. Event analysis is to be made based on causality of actions and common operation of artifacts. Some analysis results may be feed back into the work environment (distribute). A sideshow window may be adopted to notify team participants of relevant information, thus, reducing the need for another interference in the applications. Collaborative editing is the more challenging extension because we need to feed remote events back into the application, which is not always possible.

4 A Process to Increase Awareness in DSDEs

This process aims to organize guidelines to developers producing a DSDE. The process definition prescribes four different roles. Each role requires a set of skills and attributes some responsibilities. Process enactment requires the identification of which roles each participant will have. We expect these roles to be performed by different persons. This allows work specialization and reduces the complexity of a DSDE.

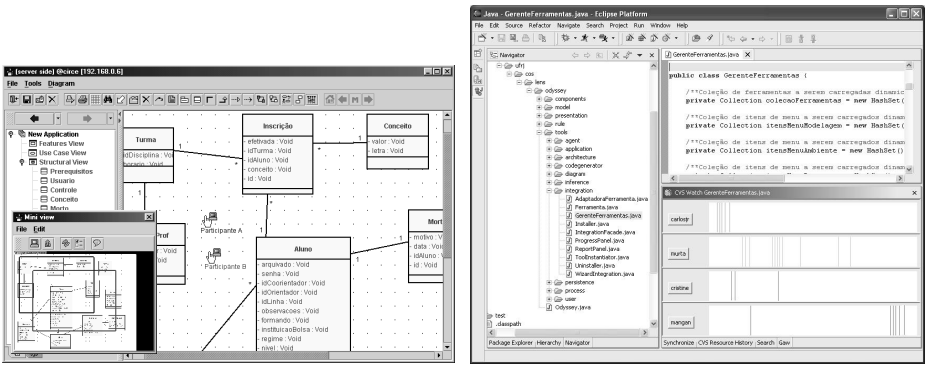
We summarize the role descriptions as follows. **CASE tool developer** is responsible for the development of a particular CASE tool development. This role may be responsible for providing generic tool extension mechanisms. **Enhancement developer** is responsible for a particular collaborative extension. **Integration developer** is responsible for develop adapters between a particular CASE tool and a particular extension implementation. This role requires a highly skilled programmer. In some cases, this role may need the assistance of the CASE tool developer. **Environment composer** is responsible for the selection and composition of a CASE tool and collaborative extensions in order to produce an enhanced tool, that is adequate for a particular scenario.

We suggest the following steps to enact the proposed approach: **People Assessment**, the environment composer role lists involved people, roles, and individual goals; **Tool Assessment**, for each person, obtain a list of the tools she uses; **Context Assessment**, for each tool, obtain a list of the actions types and object types each person wants to share and be aware of; **Extension Selection**, to select extensions that present context information for the tool set. Extension developers may be activated in this step to produce new extensions. **Collector Selection**, to select collectors that collect context information for the tool set. CASE tool developers and integration developers roles may be activated on this last step to produce new collectors. Currently, we are working on the production and adaption of documents and tools to support the proposed process.

5 Enhancement Examples

In order to evaluate the proposed approach, we are developing some prototypes that try to reproduce the collaboration support of DSDEs on top of pre-existing CASE tools. We have selected two CASE tools implementations: the Odyssey SDE's [9] Unified Modeling Language (UML) diagram editor (Fig. 2(a)) and the Eclipse IDE's text editor (Fig. 2(b)). All tools involved are Java applications.

For the first tool, we initially proposed the following scenario: two developers need to peer-review a software model. They agree to work together at the same time, one developer exposes the design rationale and the other places questions and tries to point out problems while both review the system requirements. One designer is a senior designer that will take notes during the session. They are not in the same place, but both have the same CASE tool and access to the same shared repository. The session will last for about one hour, at an agreed upon date and time, and communication will be held over a voice chat.



(a) (b)

Fig. 2. (a) Radar view (small window at left hand side) and telepointers (cursor hands at center) and (b) Group awareness widget extensions.

This setup could be enacted without further support, although the participants would need a communication overhead to give each other indication of which diagram and model element they are presently reviewing. The application of strict-WYSIWIS screen sharing (as present in MILOS and commercial applications) is not adequate to this scenario because the participants need to access other applications in their workstations (e.g. a text editor to see the requirements document and to take notes) and may need to cross-check information of one diagram in another diagram. To overcome such problems, DSDEs such as Tukan and Serendipity provide support for relaxed-WYSIWIS application sharing. Unfortunately, both Tukan and Serendipity do not support UML diagrams and navigation support that are present in the Odyssey SDE.

The scenario indicates that one participant needs to be aware of the current positioning of the other participant. Relevant application events are mouse, window, and viewport resizes and moves. Two classic widgets for real-time collaborative authoring were implemented as collaborative extensions: a telepointer and a radar view software components. A collector was designed to capture the relevant events directly from the application platform's windowing system. The extension implementation is a refactoring of support available in groupware frameworks and kits [12] [11].

For the second tool, we propose the enhancement of awareness information on the following scenario: a developer is currently making changes to software code. The changes will take some days to be completed and she may need to ask some questions to other developers about the code. She is concerned with being aware of: (a) who is making changes in the same file that she is working on and (b) someone who had made contributions in the same files in the past.

The information she needs is available from the configuration management repository's log file. She would have to pull information and interpret it. Log entries are not user friendly and she would need to browse the file to organize the data. We come out with the refactoring of an awareness widget that provides

group awareness information [13]. The widget represents a timeline for each developer reported in the log file and uses colored bars to represent activity. The developer is able to quickly find out that three other developers have worked with the file, and that one of them is working in parallel. The collector monitors changes in real-time, therefore there is no need to poll the repository to see recent activity logs. The extension is integrated as a view in the Eclipse environment, promoting greater availability and organization of awareness information. The view can easily be hidden and displayed with a simple mouse click.

Available extensions can be applied in both CASE tools, i.e., the telepointers can be applied in the Eclipse tool and the group awareness can be applied to the Odyssey SDE. An appropriate collector implementation is the main requirement. In the examples, the first collector is generic to Java GUI applications, the second collector must be redefined to the Odyssey SDE, changing the monitoring object from file changes to model changes.

6 Conclusion

This article presents a middleware-based approach to the enhancement of awareness information available in current software development workspaces. The approach is a guideline to DSDE developers that want to build collaborative environments on top of existing CASE tools. This work proposes a new process to develop DSDEs because current environments are not being applied in real software development scenarios. We argue that DSDEs will not be adopted because a software team has the need for specific tools that must be enhanced and not replaced by collaborative counterparts, which are usually not adequate for software development. At some extent, the example enhancements demonstrate the feasibility of this new development approach.

There are some limitations on our approach. Example enhancements are limited to side-show windows and graphical elements superimposed to the application window. The collection of events is easier when the tool already has a native extensibility mechanism. In general, these extensibility mechanisms offer quite good flexibility because they must support demanding tools, such as, compilers, reverse compilers, profilers, and model analyzers. Besides these technical limitations, we have found that the collaborative work process in the enhanced tool is constrained by the individual work process in the original application. Therefore, the changes in the application and work processes are conservative.

Currently, we have new collaborative extensions under development that explore the structure of software engineering models and related metrics. In this way, we can provide a richer set of awareness information. We expect to provide a catalog of extensions, tools and collectors along with guidelines about the applicability of extensions.

Further work will propose two evaluations: a developer evaluation and an end-user evaluation. The first evaluation will have the process enacted by a team of developers trying to propose enhancements to a specific scenario. They will be oriented to adapt current extensions or suggest new extensions. The second evaluation will observe the reaction of developers actually using the enhancements in specific activities.

Acknowledgments

This work is partially supported by CNPq and CAPES grants.

References

1. Sarma, A., Noroozi, Z., van der Hoek, A.: Palantir: Raising Awareness among Configuration Management Workspaces. Proc. of Twenty-Fifth Int. Conf. on Software Engineering, May, Portland, Oregon (2003) 444–454
2. Altmann, J., Pomberger, G.: Cooperative Software Development: Concepts, Models, and Tools. Proc. Tech. of Object Oriented Languages and Systems, Santa Barbara, Aug. (1999) 194–277
3. Farshchian, B. A.: Integrating Geographically Distributed Development Teams Through Increased Product Awareness. Information Systems Journal, 26(3), May (2001) 123–141
4. Grundy, J.C., Hosking, J.G.: Serendipity: Integrated Environment Support for Process Modeling, Enactment and Work Coordination. Automated Soft. Eng., Jan., Kluwer Academic Publishers (1998) 27–60
5. Herbsleb, J.D., Moitra, D. (eds.): Global Software Development. IEEE Software, March/April (2001)
6. Maurer, F., Martel, S.: Process Support for Distributed Extreme Programming Teams. Proc. Int. Conf. on Soft. Eng., Int. Workshop on Global Software Development, Orlando, Florida (2002)
7. Schümmer, T., Schümmer, J.: Support for Distributed Teams in Extreme Programming. Succi G., Marchesi M.(eds.), Boston, MA: Addison Wesley (2001) 355–377
8. Kiczales, G.: Aspect-Oriented Programming. ACM Comp. Surveys (1996) 28(4es): 154
9. Werner, C.M.L. et al.: OdysseyShare: an Environment for Collaborative Component-Based Development. Proc. Information Reuse and Integration Conference, Las Vegas, Nevada, Oct. (2003)
10. Gelernter, D.: Generative Communication in Linda. ACM Trans. Program. Lang. Systems (1985) 7(1): 80–112
11. Gutwin, C., Greenberg, S.: Effects of Awareness Support on Groupware Usability. ACM Trans. on CHI (1999) 6(3): 243–281
12. Begole, J., Rosson, R., Shaffer, C.: Flexible Collaboration Transparency. ACM Trans. on CHI (1999) 6(2): 95–132
13. Kreijns, K., Kirshner, P. A.: The Social Affordances of Computer-Supported Collaborative Learning Environments. Proc. 31th ASEE/IEEE Frontiers in Education Conference, Oct., Reno (2001)
14. GigaSpaces Inc.: GigaSpaces Server. http://www.gigaspaces.com (2004)