Supporting an Explicit Organizational Model in Global Software Engineering Projects

Oliver Creighton, Allen H. Dutoit, Bernd Brügge
Technische Universität München
{creighto,dutoit,bruegge}@cs.tum.edu

February 20, 2003

Abstract

In this paper, we propose the integration of relevant support tools for a global software development project based on a shared organizational model. By providing a single directory service where consistency and accuracy of this model can be better controlled, we intend to achieve several benefits. In particular, the ability to automate some of the tasks associated with initiating a distributed software project, thereby reducing the latency between the setup phase and the development phase.

1 Introduction

A software engineering project requires several different tools at the same time: communication tools, like email clients or web-based bulletin boards, workflow applications for process enactment, and traditional CASE tools such as integrated development environments. Many depend on a model of the project organization. The simplest form has been traditionally to store user names and passwords, a more complicated form, which also supports workflows, includes notions of teams, roles, and resources. The tools have their own scheme to store the models, as a consequence much of the organizational information is duplicated across tools and development sites. This leads to problems of inaccuracy, redundancy, and incompleteness. In the past these were accepted by developers in local environments, as they could be resolved through informal communication.

The problems, however, have become harder in a distributed environment: If the organization is best-effort driven, as e.g. many open source projects are, change occurs frequently and spontaneously, making one person the key contact for an entire system, where most subsystems might still state a previous person as the contact. Updating this information in all associated tools across potentially several development sites quickly becomes too much of an effort for any organization, the frequent solution is therefore to simply not store this information explicitly in the first place. In general, distributed organizations suffer from reduced informal communication, which in turn results in the need to make implicit organizational knowledge explicit [2, 5]. Ideally, the information about the organization (e.g., "Who is programmer, tester, or maintainer for what?") is made explicit for every artifact in every tool, especially in a globally distributed project.

The key issue that we address in this paper is how to minimize redundancy in the organizational knowledge stored across tools in order to minimize inconsistencies. If more knowledge about the organization is made explicit and shared among many

tools, the value of this knowledge increases dramatically. Moreover, if all project participants can update the part of this knowledge that is relevant to them, the chance that this knowledge is up-to-date increases accordingly. Hence, we propose a central project directory service supporting the authentication of users, the storage of user-specific attributes (e.g., email address, web home page), team compositions, roles, and access control lists. This project directory can be conveniently accessed by any tool in the project for storing and retrieving organizational knowledge, across sites, tools, and users.

We identify four main properties that such integration efforts should expose: Solutions should be

open: Several entry levels of abstraction (from basic access protocols such as LDAP or HTTP to higher levels of abstraction such as a Java API) should be provided.

encapsulated: All integrated components (the tools and infrastructure extensions) should be exchangable by using standardized interfaces and protocols.

secure: Distribution of personal data should be definable by users, following the informational self-determination principle (making it necessary to use encryption for network communication and possibly storage).

scalable: A single directory service should be distributed across server clusters or even globally. The unity needs only to exist for data integrity, but response times and firewall issues need to be considered. Adding new tools and new sites should be easy, so a single directory hosted on a single server is not an acceptable solution.

Such a solution has several benefits. Take for example the startup phase of new projects in a project-based organization. When a large number of new hires are assigned to a newly initiated project, the project setup also faces social barriers as informal communication requires team-building and a relaxed atmosphere for getting to know each other. We compare such highly condensed project start-ups to a big-bang, where suddenly a large mass of active elements are introduced in a formerly empty environment. The formation of a clear structure on these elements is key to good collaboration. It is common practice to begin projects with some ice-breaking activities that will help participants memorize their names or get a feel for background and interests of each other. But in the case of global software development, this is sometimes impossible to organize, as sending all developers to meet everyone else is too expensive.

As the various support tools need to be installed and configured, projects usually begin with a preparation phase where a small group of project initiators try to generate as much of the anticipated required structure as possible. This task is typically on-going during the entire project duration, as changes in organization frequently also lead to changed requirements for the support infrastructure.

As mentioned, almost all tools that aid in software engineering contain some form of user management for access control, contact information, or representing the roles that various project members have in the project. By unifying some of the required installation steps through tool integration, we expect to shorten the setup phase and begin the development phase of projects earlier.

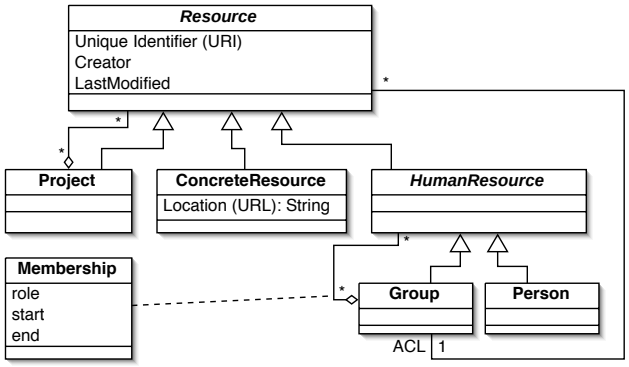
2 Single Directory Service based on an Organizational Model

The typical use of a centralized directory service in organizations is to provide a phonebook (the "White Pages" of the organization) containing basic contact information for every employee. While this is a useful and important service in itself, when it comes to supporting CASE tools in a broad sense, the standard idea of an alphabetical list of people is not sufficient. Providing information about the properties of each employee, in an organized fashion that makes it straightforward to look for related work areas, would be another use (the "Yellow Pages" of the organization).

But even if a central directory service for these purposes is installed, and is provided by standard access mechanisms such as LDAP, the benefit is not yet reaching the CASE tools directly. To provide integrated user and resource management, it is necessary to first make the organizational model explicit

and then to share it across tools and sites for the consistency reasons explained before.

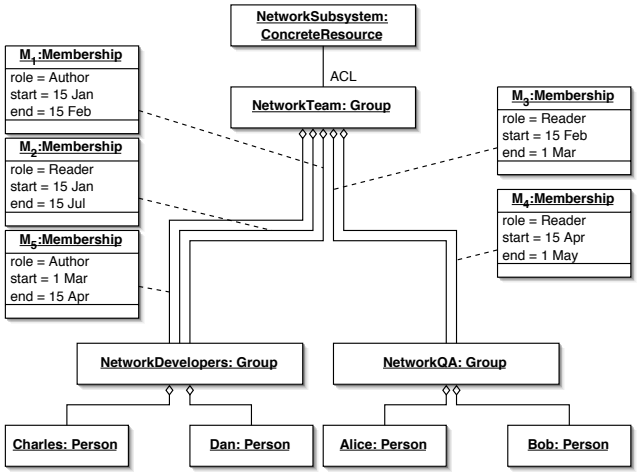
The basis of our directory is the following object model, which we believe can express all organizational models we encounter in the various project courses that we test our tools in (cf. section 3):



In short, human resources can be grouped in arbitrary depth, and the associations have attributes to support changes in the organizational structure over time. Projects are represented as the sum of all resources they contain, and on the abstract level of *Resource*, the basic functionality for authorization is integrated by associating it with one group as its Access Control List (ACL). The exact access role is represented by the role attribute in the Membership class, which states, for example in the directory itself, roles like "Administrator", "Author", or "Reader". But since this attribute is interpreted by the individual applications, it can be extended with other roles as they require.

With this model, the directory service is predestined to provide a service for authentication, to verify the identity of users, as it then can be assured that if applications want to control access to specific entity objects they control, and they are concrete subclasses of *Resource* in our model, the directory service can provide verification of access (authorization) for users.

Consider the following example, illustrated by an instance diagram of the directory object model:



The NetworkSubsystem, a ConcreteResource controlled and modified by different tools (such as a version control system for the source code and a workflow support tool for quality assurance scheduling) is associated with the NetworkTeam. This Group is interpreted as the ACL for the resource, which causes that the role attribute in the Membership association objects specifies the access rights of the contained *HumanResources*, in our case the two groups NetworkDevelopers and NetworkQA. Our example shows how one would model a source code promotion strategy, where after the code freeze deadline of February 15th, developers can no longer modify the subsystem, as it has been handed over to the quality assurance (QA) group for testing. The QA group's deadline for testing is in our example March 1st, after which developers are again allowed to modify the subsystem, but the QA people have no access until the next cycle, starting April 15th.

For the entire duration of the project, developers are allowed to read source code; note that Membership objects can exist multiple times for the same time span and *HumanResource* combination, only varying in role. The Membership objects between the Groups and Persons that we omitted in the instance diagram could be used for temporarily assigning more people to the testing group, but the role attribute will be insignificant for the NetworkSubsystem as only the NetworkTeam group is interpreted as its ACL. In other words, the containment relation is transitive, whereas the *is-ACL-for* relation is not.

By storing this information in a single directory and providing mechanisms for modifying it, while retaining a consistent access restriction as to who is allowed to, it becomes possible to integrate functionality in the workflow tool for project management to delay the date of code promotion to QA stage, and automatically assure that developers will continue being able to modify the source code without changing anything in the version control system.

3 Evaluation Environment

In researching distributed software engineering, we have taken the approach of "learning-by-doing." We have taught several global software engineering (Global$^{S}_{E}$) project courses in which teams of students located in Pittsburgh, PA and in Munich, Germany collaborated on developing a system for a single industrial client [1, 3]. We follow a sawtooth process in which developers present a sequence of incrementally more refined prototypes to the client and to project management, so that the scope and the direction of the project can be refined at regular intervals.

Distributed reviews are done using a combination of video conferencing, phone, application sharing environments, shared slides, and the web. The infrastructure also included asynchronous tools such as Lotus Notes bulletin boards, CVS for version control, a UML modeling tool, and an integrated development environment. More recently, we have also introduced a requirements management tool [4], an awareness infrastructure [6], and a workflow tool for process enactment. As most tools do not share the same user information, developers need a user name and password for each tool. Moreover, while the infrastructure available to students includes the same set of tools in both sites, these tools are usually administered locally, resulting in duplication of user and project organization information, as described in the introduction. Worse, not all knowledge for administrating this information is available in any one site, resulting in inconsistencies and information that is out of date (e.g., users who left the project, teams whose name and purpose changed).

To address these issues and to evaluate the value of a single project directory approach, we have been adapting the project infrastructure so that it uses the directory described in the previous section. In particular, the following aspects of the environment are being revised:

User authentication. The most immediate benefit of the single directory is that users have a single account across all tools in the infrastructure. If they change their password, the new password will take into effect immediately. As the directory is shared across sites, users also have a single account across all sites. Similarly, all attributes associated with users, such as email addresses used for notifying developers, are stored only once.

Directory user interface. We implemented the directory in our Lotus Notes infrastructure. As a consequence, users can update their directory entries as before, by editing their record in the Lotus Notes address book. As every user has write access to their own data, the attributes associated with each users are kept up to date more often.

Team structure. The team membership of each user is represented in the directory as groups, which are also resources associated with each project. As developers change teams and often take part in more than one projects, this enables us to track team membership over time, when assessing a developer's skills. From the infrastructure stand point, this also enables tools to leverage off the team organization. For example, the workflow tool we use allows tasks to be assigned to a team or to an individual. When the task responsibility changes, an email notice is sent both to the participants who are responsible and to those who were responsible for the task. The workflow tool retrieves the list of team members from the directory. As other tools which provide group notifications (e.g., Lotus Notes) also retrieve the same information, the actions of these different tools are consistent. Moreover, this encourages participants to keep this information up-to-date.

Role and access control. The Group and Membership classes in the directory enable us to represent role information. By using a Group as an access control list, each tool can offer a differ-

ent behavior depending on the role of the user. For example, the workflow tool only allows users with a manager role to change the process model. Users with a coach role can change the responsibility of each task. We are currently modifying other tools to take advantage of this information. For example, in our requirements tool, only analysts would be able to modify the requirements while both analysts and reviewers would be able to annotate the requirements with questions.

We plan to complete these changes and evaluate the single directory concept by our next distributed project course, which will take place during the summer between Munich, Germany and Otago, New Zealand.

4 Conclusion

In this paper we propose a technology-driven approach, as opposed to a business administration-driven approach, for enabling software engineers to build better software through rationale capture and knowledge management.

Key challenges in project organization include identifying incentives for developers to accept the tools the organization wants to employ and addressing individuality and privacy concerns.

When identifying potential incentives, we note that lack of sharing of information among sites leads to adverserial relationships and lack of trust. We anticipate that sites can benefit from the system by offering a greater transparency into their activities. Such transparency can then lead, for example, to certification frameworks for supplier sites and reinforce long term relationships among sites.

When addressing privacy concerns, we propose that providing a unified storage and access control model that is powerful enough to individually set the data distribution scope while at the same time, through unification, being simple enough to allow every individual to determine the scope themselves, we can offer enough flexibility for people to trust a centralized storage of their personal information.

In general, the above issues are difficult to predict and anticipate, as they relate to complex organizational and human processes. Only an experimental approach will enable us to assess the impact of the system with respect of these issues and design solutions to address them.

We intend to continue our approach to first integrate the organizational model across tool boundaries with integrating entity objects that CASE tools work with by identifying common data structures that could then also be shared in a similar fashion. The description of our underlying meta-model encompassing not only the Global$^{\mathrm{S}}_{\mathrm{E}}$ organizational models, but also the system models and rationale models is out of scope for this paper and can be found in Kobylinski et al.[6].

References

[1] B. Bruegge, A. H. Dutoit, R. Kobylinski, and G. Teubner. Transatlantic project courses in a university environment. In *Asian Pacific Software Engineering Conference*, Dec. 2000.

[2] B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11), Nov. 1988.

[3] A. H. Dutoit, J. Johnstone, and B. Bruegge. Knowledge scouts: Reducing communication barriers in a distributed software development project. In *Asian Pacific Software Engineering Conference*, Dec. 2001.

[4] A. H. Dutoit and B. Paech. Rationale-based use case specification. *Requirements Engineering Journal*, 2002.

[5] R. E. Grinter, J. D. Herbsleb, and D. E. Perry. The geography of coordination: Dealing with distance in R&D work. *ACM*, 1999.

[6] R. Kobylinski, O. Creighton, A. H. Dutoit, and B. Bruegge. Building awareness in distributed software enginering: Using issues as context. In *International Workshop on Distributed Software Development, International Conference on Software Engineering*, May 2002.