SOFTWARE PROCESS IMPROVEMENT AND PRACTICE
Softw. Process Improve. Pract. 2005; **10**: 255–272
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spip.228

Process Modeling Across the Web Information Infrastructure

Research Section

Chris Jensen*,† and Walt Scacchi
Institute for Software Research, University of California-Irvine, Irvine, CA, USA

Web-based open source software development (OSSD) project communities provide interesting and unique opportunities for software process modeling and simulation. While most studies focus on analyzing processes in a single organization, we focus on modeling software development processes both within and across three distinct but related OSSD project communities: Mozilla, a Web artifact consumer; the Apache HTTP server that handles the transactions of Web artifacts to consumers such as the Mozilla browser; and NetBeans, a Java-based integrated development environment (IDE) for creating Web artifacts and application systems. In this article, we look at the process relationships within and between these communities as components of a Web information infrastructure. We employ expressive and comparative techniques for modeling such processes that facilitate and enhance understanding of the software development techniques utilized by their respective communities and the collective infrastructure in creating them. Copyright © 2005 John Wiley & Sons, Ltd.

KEY WORDS: interorganizational process modeling; process collaboration; process conflict; interorganizational interaction; process integration; open source software development; Apache; Mozilla; NetBeans

1. INTRODUCTION

Previous studies of interorganizational processes focus on devising languages to represent workflows across organizational boundaries (Rasch and Hansen 1997, Lenz and Oberweis 2001, Shen and Liu 2001) and verification of workflow nets (van der Aalst 2002b). Little work (van der Aalst 2002a) demonstrates real-life interorganizational processes. In contrast, this article contributes a framework for discovering, analyzing, and modeling interorganizational software processes within a multiproject software ecosystem. The ecosystem selected is a network of organizations responsible for core of the Web information infrastructure. Although nonopen source organizations are also members of this ecosystem, our focus is on open source software development (OSSD) organizations. Large-scale geographically distributed software development projects, such as OSSD project communities, present challenging process problems. The Apache, Mozilla, and NetBeans[1] OSSD communities collectively have millions of estimated users, and tens of thousands of community participants contributing in one fashion or another. Such magnitudes would be difficult for most closed

* Correspondence to: Chris Jensen, Institute for Software Research, University of California-Irvine, Irvine, CA, USA 92697-3425
† E-mail: cjensen@ics.uci.edu

Contract/grant sponsor: US National Science Foundation; contract/grant number: 0083075; 0205679; 0205724; 0350754

[1] The Eclipse project affiliated with IBM is similar in many ways to the NetBeans project, in that both efforts are very large OSS projects developing Java-based IDEs. But for our study, we selected the NetBeans project.

source organizations to manage. Yet, these three communities have proven extremely successful at it. Further, they have done so in a delicate ecosystem that includes evolving Web standards, data and software repositories, and tools. These serve as a framework for integrating each community's tools together. Therefore, as the components of this framework coevolve, each community must synchronize, (re)integrate, and stabilize its position within the process space.

In this article, we look at processes within and across three related OSSD project communities (cf. Scacchi 2002). In our efforts to model software development processes on both community and infrastructure levels, we have used a variety of techniques. These include detailed narrative models of processes, semistructured hyperlinked models, formal computational process models, and a reenactment simulator (Scacchi *et al*. 2004), all of which serve as input for other process engineering activities (Scacchi and Mi 1997, Noll and Scacchi 2001). Further, all of our process models are hypermedia artifacts that, when submitted back to the communities, may be consumed by the processes they describe.

From here, we will set the stage for our investigation with a discussion of each process modeled independently before examining the infrastructure as a whole in order to examine intercommunity process modeling issues and outcomes as we found them. Finally, we look at the modeling techniques themselves, how they may be used to guide developers, and how they can serve as a basis for process simulation and other process activities.

2. MODELING PROCESSES WITHIN WEB INFORMATION INFRASTRUCTURE PROJECTS

The Apache Web server, Mozilla Web browser, and NetBeans integrated development environment (IDE) together form a Web information infrastructure for developing and deploying Web-based software applications, content, and services (Fielding *et al*. 1998, Mockus *et al*. 2002). However, as the projects that develop each of these three open source software systems operate as virtual enterprises (Noll and Scacchi 1999), we have no basis to assume that their development process activities, roles, or tools are identical or common, nor how or where they interact. Thus, in order for these projects, and other OSSD projects like them, to collectively produce and sustain a viable global Web information infrastructure, they must be able at some point to synchronize and stabilize their processes, their process activities, shared artifacts, and targeted software releases (cf. Cusumano and Yoffie 1999).

Before we can understand software development processes across each of these three Web information infrastructure components, we must understand them individually. As previously introduced (Jensen and Scacchi 2003b, Scacchi *et al*. 2004), we address three process modeling techniques here as a sampling of those we have applied in our study. These are the rich hypermedia, process flow graphs, and formal modeling. Formal modeling in turn supports tools for simulated reenactment of software processes, which is used to preview, interactively walkthrough, validate (Atkinson and Noll 2003), and support process training on demand (Scacchi and Mi 1997). Additional details on these techniques can be found elsewhere (Scacchi *et al*. 2004).

This section seeks to address both of these issues. We start by presenting a brief overview of the *quality assurance (QA) process* in the Mozilla Web browser release cycle, modeling it as a rich hypermedia, followed by the Apache *release process,* modeled as a flow graph, and lastly, the NetBeans *requirements and release* process, which we model formally and reenact. For brevity, we will skip presenting the rich hypermedia, flow graphs, and formal models for each process; however, these models are detailed elsewhere (Ata *et al*. 2002, Carder *et al*. 2002, Jensen and Scacchi 2003b, Oza *et al*. 2002).

2.1. Rich Hypermedia Modeling with the Mozilla Quality Assurance Process

Building on and extending the rich picture concept described by Monk and Howard (1998), we created a rich hypermedia variant as an informal model of software development processes in each of Mozilla, Apache, and NetBeans projects. These models show the relationships between tools, agents, their development concerns (i.e. nonfunctional requirements), and activities that compose the overall process, and its functional requirements. While Monk and Howard propose a flat, static model, our hypermedia is interactive, deep, and navigational (Noll and Scacchi 2001), including process enactment

scenarios described and hyperlinked as use cases. Use cases are a known technique compatible with the Unified Modeling Language (UML) for representing user-system process enactment scenarios (Fowler 2000). The hypermedia artifacts are also annotated with detailed descriptions of each tool, agent, and concern. Each of these process objects is hyperlinked to its description. Descriptions can, in turn, be linked to other data or hypermedia resources. In this way, the modeler can define the scope of the rich hypermedia to include as little or as much information as the need requires. The rich hypermedia provides a quickly discernable intuition of the process without the burden of formalization. A rich hypermedia model for the Mozilla quality assurance process is shown as an image map in Figure 1.

The daily Mozilla QA cycle (Carder *et al*. 2002) begins with the closing of the source tree to submissions. After this, the 'code sheriff' and system build engineer create a build of the source code tree using the Mozilla Tinderbox build tool. If build errors are present, the sheriff and build engineer contact the 'on the hook' developers, reviewers, and super reviewers responsible for the build source, then they are called on via e-mail notifications to correct the defects. When the defect is corrected or the problematic source code is removed, the source is rebuilt. This process iterates until all build errors are corrected, and the compiled and built source and executable image is ready for the next process step.

The build results (including the executable software image) are placed on the community FTP server and the 'smoke test' coordinator issues a call for developers and volunteer testers via the community Internet relay chat (IRC) channel to download and evaluate the build results (e.g. execute the image on a test platform). After this, developers participating as QA contacts, QA owners, and volunteer

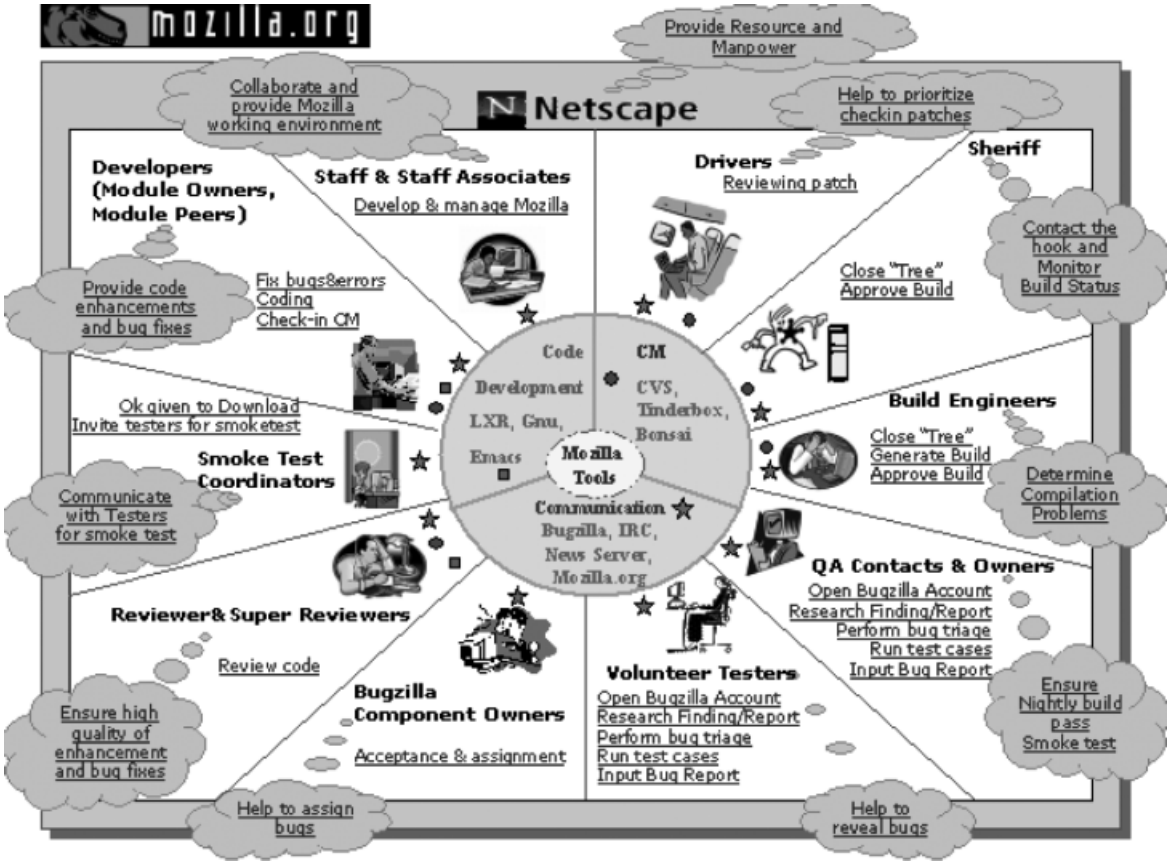


Figure 1. Mozilla quality assurance process rich hypermedia (cf. Carder *et al*. 2002)

testers will announce what they plan to test, download, and install the build and perform a series of smoke tests, security-specific (SSL) smoke tests, or less critical 'general tests' (periodic regression checkups), based on bug reports submitted to the bug repository. Testers note and discuss the results over the IRC channel. Critical bugs are identified and assigned to the on-the-hook developers to be patched, whereupon the source is retested. Noncritical bugs are set aside until another tester confirms them, uploaded to the Bugzilla defect repository, and further dealt with at a later time. Once all critical defects are corrected, the sheriff and build engineer reopen the source tree to further development and source submission.

When first detected, defects are entered into Bugzilla as unconfirmed, noting their severity, component, and platform where the defect was observed. A member of the quality assurance team (either a QA contact or owner) must then research the defect and certify it as a new defect or marking it as a duplicate of another known defect. Patches are then created by developers during the course of development or by drivers as the release date approaches to ensure the overall quality of the product, and the status revised to reflect the changes.

2.2. Process Flow Graph Modeling with the Apache HTTP Server Release Process

The process flow graph illustrates the flow of resources (development artifacts) through a path of interaction. The interaction accounts for process agents using tools that manipulate the resources through performance of tool-based activities. This semistructured workflow representation provides a partial ordering of the process fragments and allows us to tease out dependencies between artifacts and activities seen in the rich hypermedia. It also offers an idea of which artifacts and activities are most vital to development, by measuring the fan-in and fan-out of each.

These artifacts are likely to be the cause of bottlenecks in the development process when they are found to be inadequate, incomplete, or faulty results of prior development activities. Borrowing from Web modeling terminology, an artifact that is a hub or nexus for several activities will hold up development until it is completed or found satisfactory. Likewise, an artifact that is a product of several inputs inhibits activities that require it until it is ready for further processing. Additionally, we can also detect cycles of development (re)work, such as in the stabilization process and refining the software build release plan.

While these insights can be captured in other representational forms, this diagram, like the rich hypermedia, provides an overall representation of the context for process activities without the weight of the details of more formal models. Process entities shown in the flow graph may also be hyperlinked to resources in the community Web to provide interactive richness, as well as to enable process inspection activities. Figure 2 shows a process flow graph for the Apache HTTPD Server project's release process, where the boxes denote process activities and ellipses denote the resources or artifacts flowing through the process. Further, software developer roles are associated with each process activity.

As shown by this flow graph, in the Apache release process (Ata *et al*. 2002, Erenkrantz 2003), anyone can submit features or bug fixes in the form of patches to the server project. To be included in the project source repository, a 'committer' who has write access to the repository must upload the code. These patches are uploaded to the development branch of the repository. It may then be promoted to the stable release branch by a vote of the committers. While any developer can vote on an item, only the votes of primary author, committers, and members of the project management committee are binding (Fielding 1999). To be promoted, there must be at least three binding positive votes, and no vetoes. Further, for a release, there must be a majority approval (that is, at least three binding positive votes, and more positive than negative votes). Granting a developer committer status requires a complete consensus of the project management committee.

When a committer decides to create a release, she/he typically sends an announcement to fellow developers outlining the release plan, and stating who the release manager will be. Acceptance of the release plan requires a 'lazy consensus' approval of the committers[2]. As development moves towards completion, the release manager determines which features are fit for inclusion in the release and

[2] http://httpd.apache.org/dev/guidelines.html

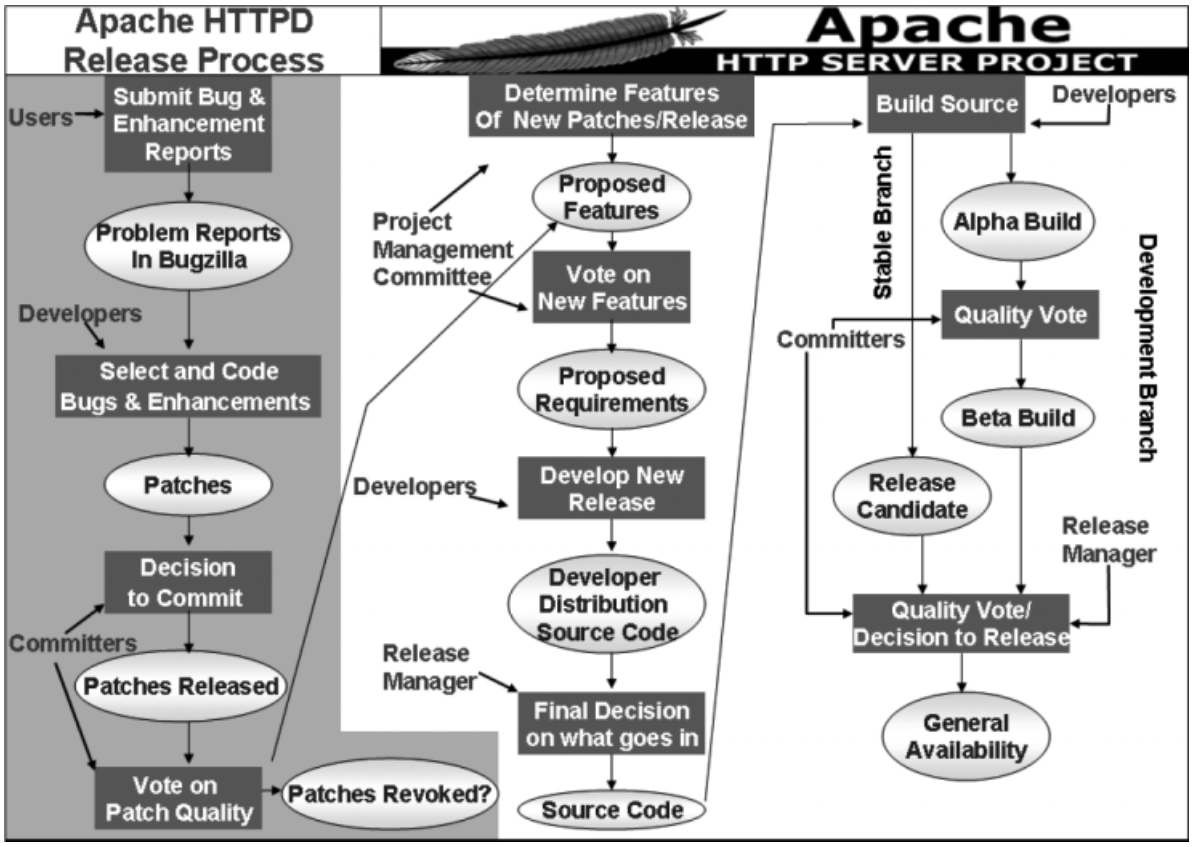


Figure 2. Apache HTTP server release process flow graph (cf. Ata *et al*. 2002)

which are not. Those that pass are compiled into an alpha build, which is made available on the community Web site and announced on the developer mailing lists. Developers and committers are then called upon to test the build on their own servers manually or through use of the automated Apache server test suite. Discovered defects are sent to the community development e-mail list (and potentially submitted to Bugzilla) and patched by developers and subsequently subjected to the patch review process. In the stable source branch, releases that pass the committer vote usually skip the beta and release candidate phase and proceed to general availability.

When the release manager is adequately satisfied with quality of the source in the development branch, she/he will declare the release suitable for beta or final release candidacy. When she/he announces this, the builds are made available on the main page of the community Web and adopted by a wider audience, for continued testing and patching. At some point, the release manager deems the source fit for general public use and creates a general availability build release, announcing it on the development, committer, and tester mailing lists. This build is then voted on by the committers and tested on the Apache community Web site. If there is a simple majority of approval and at least three positive votes, the release is declared final. The result is announced via the community Web and mailing lists, and then released for distribution via a system of mirrored Web sites.

2.3. Formal Modeling and Reenactment Simulation with the NetBeans Requirements and Release Process

We developed formal models of software processes following our preexisting process meta-model (Mi and Scacchi 1996) using Protege-2000 knowledge editing environment (Georgas 2002, Noy *et al*. 2001). The resulting models have the form of a semantic

web/hypertext (Noll and Scacchi 2001). The work done here is identifying instances for all the process meta-model components: agents, resources, tools, actions, and activity control flows, which we represent using the Protege-2000 tool. Once a process instance is input, it may be exported to an XML format, a graphical representation using the OntoViz tool, or to our process modeling language, PML (Noll and Scacchi 2001).

Protege-2000's editing and visualization facilities provide for a multitude of alternative views and visual rendering of the modeled process components, as well as their interrelationships and dependencies. As a result, the graphical rendering of a process model or process object-relation class views can at times be more intuitive than a coded textual format. Figure 3 shows a graphic representation of an underlying PML model of the NetBeans Requirements and Release process that has been interpreted for visual rendering and layout of its relational interdependencies. However, the textual PML representation can be used as input to other process engineering tools such as those supporting process enactment or process improvement.

The first step in the NetBeans requirements and release process (Oza *et al*. 2002, Jensen and Scacchi 2003b) is to establish a release manager, a set of development milestones (with estimated completion dates), and a central theme for the release. The theme is selected by the community members who have taken charge of the release, with the goal of overcoming serious deficiencies in the

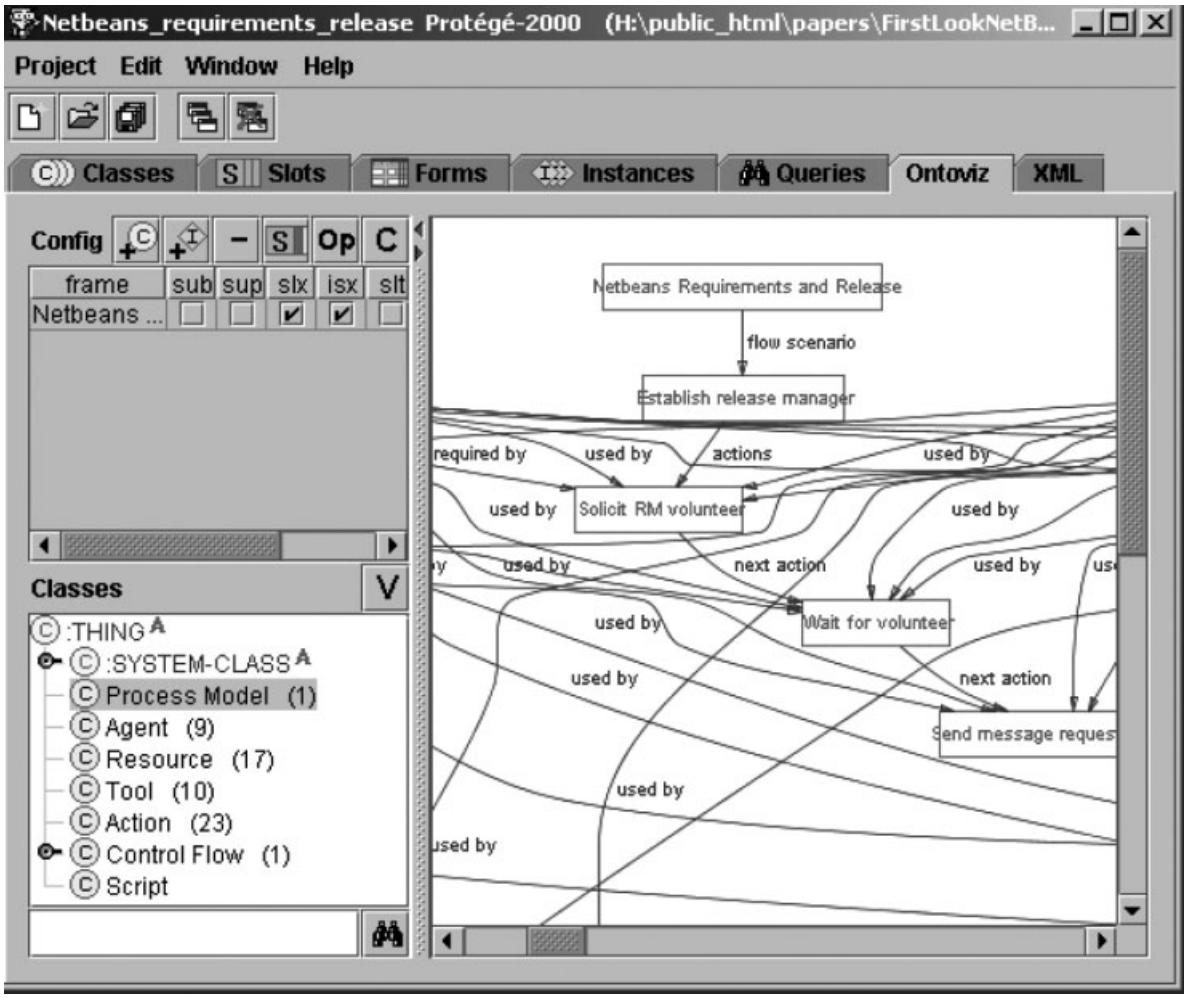


Figure 3. Visual rendering of the NetBeans requirements and release process formal model using Protégé-2000

product (e.g. quality, performance, and usability), in addition to new features and corrective maintenance planned by module teams. Historically, most releases have been led by members employed by Sun Microsystems, which provides development and financial support for the community, though volunteer releases also occur. On the basis of this, and in conjunction with input from the feature request reports, lead developers will draft a release plan, providing the milestones, target dates, and features to be implemented in the upcoming release. After review and revision by the community, the plan is accepted and developers are asked to volunteer to complete the tasks outlined therein and a volunteer is sought to act as release manager and coordinate efforts of community. Usually, a developer will either volunteer or be volunteered for the role via the mailing list by and accepts the nomination or is accepted through community consensus.

All creative development must be completed by the feature freeze milestone date specified in the release proposal, which signals the end of the requirements subprocess and the beginning of the stabilization phase, the release subprocess. At this point, only bug fixes may be submitted to the source tree. The stabilization phase consists of a build-test-debug cycle. Nightly builds are generated by a series of automated build scripts and subsequently subjected to a series of automated test scripts, the results of which are posted to the community Web site. Additionally, the quality assurance team performs a series of automated and manual testing every few weeks, as part of the Q-Build program with the aim of ensuring that source code submitted regularly meets reasonable quality standards. Defects discovered during testing are then recorded in the IssueZilla issue repository and subsequently corrected. When the release branch is believed to be devoid of critical 'show-stopping' defects, it is labeled a release candidate. If a week passes without any further showstoppers, the release candidate is declared final; else the defect is corrected and another release candidate is put forth. In addition to the formal depiction given in Figure 3, the NetBeans requirements and release process has also been reenacted. The motivation for this is as follows.

Process analysis seeks to identify potential pitfalls that can be discovered before or after their deployment or adoption in a project. Process model verifiers check static semantic properties such as whether a resource required downstream is provided by some upstream process activity (Atkinson and Noll 2003). Process simulators provide dynamic analysis capabilities through enactment or reenactment of processes, which are especially useful when validating, modifying, or redesigning a process, as well as for providing on-demand training (Scacchi and Mi 1997, Scacchi 2000, Atkinson and Noll 2003).

Our process enactment simulator (Choi and Scacchi 2001, Noll and Scacchi 2001) interactively serves a series of Web pages, or links to multiple alternative pages, according to the control flow expressed in the PML model of the process flow graph. This simulator allows process performers and other community members to simulate enacting the process through a step-by-step interactive walkthrough. With such a reenactment simulator, developers within a project may be able to exercise, critique, and identify improvement opportunities within processes that can be observed at a distance. It also provides the potential for a more easy transition from the simulator to live process enactment transactions on the community Web site.[3] In doing so, we have been able to detect processes whose workflows may be 'hidden', unseen, or unfamiliar to project participants with little/no involvement with the process. Similar analyses can also detect process flow segments that are unduly lengthy or otherwise suboptimal, which may serve as good candidates for process improvement or redesign. It allows for viewing and detection of the effects of duplicated work by participants that may be unaware of such duplication. Figure 4 thus displays a screen shot of one step during the reenactment of the NetBeans requirements and release process (Jensen and Scacchi 2003b). This display shows the process decomposition on the left, the enactment step, and corresponding link traversal options next, and then the enacted step's invocation on the right.

With the above insight into the development processes executing within Mozilla, Apache, and NetBeans, we can now explore development processes between them.

[3] For example, the NetBeans.org project posted a link to our ProSim'03 Workshop article (Jensen and Scacchi 2003) where some of these ideas were initially proposed and evaluated. See http://www.netbeans.org/community/articles/UCI_papers.html.

```
sequence Set Release Date {
  ...
 iteration Update IssueZilla {
  action Report Issues To IssueZilla {
    requires { Test results }
    provides { IssueZilla entry }
    tool { Web browser }
    agent { users, developers, Sun ONE Studio QA team, Sun ONE Studio developers }
    script {
     <br><a href="http://www.netbeans.org/issues/">Navigate to IssueZilla </a>
     <br><a href="http://www.netbeans.org/issues/query.cgi">Query IssueZilla </a>
     <br><a href="http://www.netbeans.org/issues/enter_bug.cgi">Enter issue </a> } }
 ...
```

The PML fragment (excerpt) that specifies a process step for the action, “Report Issues to IssueZilla”, corresponding to its reenactment below.

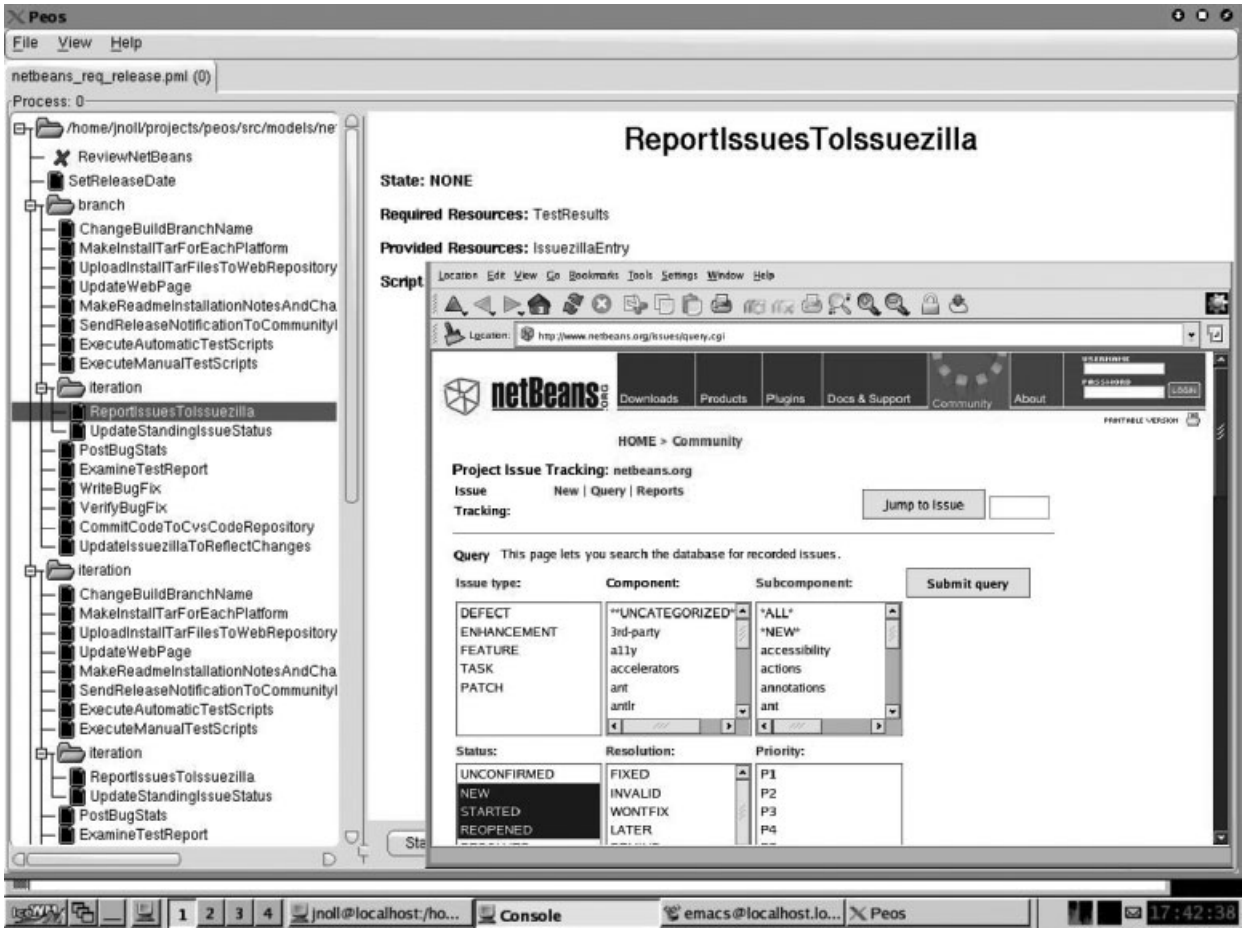


Figure 4. A step in the simulated process reenactment of the NetBeans requirements and release process (cf. Noll and Scacchi 2001)

3. MODELING PROCESSES ACROSS WEB INFORMATION INFRASTRUCTURE PROJECTS

We would like to be able to model interorganizational processes using the same techniques we use to model intraorganizational processes: top–down modeling followed by bottom–up discovery. In intraorganizational process modeling, we often know *a priori* the types of activities, artifacts, tools, and roles we are likely to see (Jensen and Scacchi 2003b). Subsequently, we can create a reference model describing how to classify their instances (Jensen and Scacchi 2003a). This guides us in our search for process data within the software development artifacts available for study. In modeling interorganizational processes used across projects in the Web information infrastructure, we have no such reference model. Instead, we first step back and examine the types of relationships that exist between organizations. Management and information systems research give us tools to characterize the types and extent of interorganizational processes. We show how to identify stakeholders (boundary agents) and objects of interaction (boundary objects), and their concerns, modeling them as a rich hypermedia. Next, we examine communication processes between organizations, as are modeled in flow diagrams, as described previously. These can be then modeled formally, and reenacted via simulation. Finally, we apply this strategy in a detailed example.

3.1. Characterizing Interoperation Among Web Information Infrastructure Projects

Successful interoperation between components of a Web information infrastructure depends on the relationships between the organizations of which it is comprised. Traditional management literature (Bluedorn *et al*. 1994) identifies six mechanisms of interorganizational interoperation or integration, which entail loose or tight coupling. These mechanisms include joint ventures, network structures, federation, cooperative agreements, trade associations, and interlocking directorates (see Table 1). Though originally devised to describe corporate and government interorganizational relationships, our choice is to apply them to provide an overall characterization of a software ecosystem for, in this case, the Web infrastructure. Unsurprisingly, the degree of coupling carries implications for the degree of process integration between these organizations. Alter (1999) defines degrees of process integration (ranging from loose to tight coupling) to include sharing a common culture, utilizing common standards, information sharing, coordination, and finally collaboration, as described in Table 2. Together, these give us a basic framework for examining the types of processes we can

Table 1. Interorganizational synchronization and stabilization mechanisms (after Bluedorn *et al*. 1994)

Interorganizational form	Example	Tightness of coupling
Joint venture	Apache, GNU foundation members	*Tightly coupled*. Two or more firms form a separate entity for a variety of strategic purposes (e.g. market power, efficiency, transfer of learning).
Network structure	System plug-in developers	*Tightly coupled*. A hub and wheel configuration with a focal firm at the hub organizing interdependencies of a complex array of firms.
Federation	Mozilla 'on-the-hook' developers	*Tightly coupled*. Established to manage and coordinate the activities of affiliated members (common in hospitals). The federation controls all or part of the management activities of the members.
Cooperative agreements	Meta-communities (e.g. JTC)	*Loosely coupled*. Arrangements between two or more firms that have strategic purposes, but do not have shared ownership.
Trade associations	Tool integration	*Loosely coupled*. Distribute trade statistics, analyze industry trends, offer legal and technical advice, and provide a platform for collective lobbying.
Interlocking directorates	NetBeans governance/community management	*Loosely coupled*. Information sharing, expertise and enhanced organizational reputation.

Table 2. Levels of business process integration (cf. Alter 1999)

Level	Example	Description
Common culture	OSSD motivations, development methods	Shared understandings and beliefs
Common standards	Data formats, communication protocols	Using consistent terminology and procedures to make business processes easier to maintain and interface
Information sharing	OSSD Web repositories	Access to each other's data by business processes that operate independently
Coordination	Meta-communities, tool integration, plug-in development	Negotiation and exchange of messages permitting separate, but interdependent, processes to respond to each other's needs and limitations
Collaboration	NetBeans, Mozilla spell-checking module development	Such strong interdependence that the unique identity of separate processes begins to disappear

expect to find in the Web infrastructure. Further, they provide the constructs of the rich hypermedia: interacting members of the Web infrastructure (stakeholders), their relationships, and the motivations of these relationships (concerns). Identification of these stakeholders, relationships, and concerns requires analysis of the interprocess communication among infrastructure projects. We address this next.

3.2. Interprocess Communication Among Web Information Infrastructure Projects

Communication between project communities provides opportunities both for integration and sources of conflict between them (Elliott and Scacchi 2003, Jensen and Scacchi 2004). We will say communication is *integrative* if it identifies compatibilities or potential compatibilities between development projects. From a process perspective, integrative communication enables external stakeholders to continue following their internal process as normal, perhaps with a small degree of accommodation. They also reinforce infrastructural processes since they do not require changes in the interoperations between communities. If the degree of accommodation or adaptation becomes too great, this can precipitate *conflictive* communication between project communities. Conflict may occur due to changes in tools or technologies shared between them, or in contentious views/beliefs for how best to structure or implement new functionality or data representations across projects. These conflicts may require extensive process articulation to adapt (cf. Scacchi and Mi 1997). Sections 4.2.1 and 4.2.2 take a closer look at process integration and conflict, followed by a discussion of how these processes are discovered and modeled.

3.2.1. Process Integration

Process integration can be direct and explicit, as in the case where NetBeans and Mozilla community members collaborated on a spell-checking module. But, it can also be indirect and implicit. For Mozilla's browser to correctly present Web artifacts, it must implement both protocols for processing Web transactions to the Apache server, and also standards for displaying content of the document or object types generated by NetBeans IDE. Similarly, the NetBeans IDE must produce artifacts and applications forms that artifact consumers, including Mozilla's browser and Apache's server, expect. The Apache server, for its part, must comply with the standard transaction protocol the Mozilla browser anticipates (e.g. HTTP/1.1) and provide Web application module support required by applications produced by the NetBeans IDE. Although these communities may not have explicitly negotiated and agreed on common data standards to be used between them, they have individually implemented standards provided and maintained by outside parties – particularly, the W3C, the World Wide Web Consortium.

These standards can be viewed as objects of interaction or *boundary objects*[4] (Star 1989, Pawlowski *et al*. 2000). Following Alter's (1999) classification, shared standards connote a low degree of process interaction between organizations in the Web infrastructure. However, other boundary objects exist, as shown in Table 3. Within OSSD project communities of the Web infrastructure, boundary objects include (a) shared beliefs and culture (Elliott and Scacchi 2003), (b) community infrastructure tools,

[4] Boundary objects are those that inhabit and span several communities of practice, as well as satisfy the informational requirements of each community (Star 1989).

Table 3. Boundary objects of the Web information infrastructure

Object type	Example
Community infrastructures	
Community culture/bylaws	Source licenses, governance style, community organizational composition
Community infrastructure tools	Defect repositories (e.g. Bugzilla, IssueZilla), collaborative development tools (e.g. WIKI, CVS, mail list managers)
Development processes	Defect discovery/submission procedures, source check-in procedures
Product infrastructure	
Product infrastructure tools	Plug-ins, modules, libraries
Development artifacts/software informalisms	Software documentation, how-to guides, design styles (e.g. P2P, client-server)
Protocols	HTTP, RPCs
Shared data formats	HTML, CGI, XML

such as defect repositories produced by other affiliated organizations (Halloran and Scherlis 2002), and (c) development processes. Additional boundary objects are found in the product infrastructure (e.g. applications program interfaces and remote procedure calls that enable data sharing and remote invocation of software modules across systems). These may take the form of software application plug-ins or modules. The Java-based Tomcat Web Server, created by the Apache community and integrated into the NetBeans IDE, is one example. They may share or coordinate development artifacts. And, as discussed, they may implement or utilize common data communication protocols and data representation formats that enable reliable communication between their tools. Although no structure is implied by the modeling paradigm, our rich hypermedia has traditionally featured development tools at the center of intraorganizational process models. In modeling the Web information infrastructure, we have expanded our view to include other types of boundary objects, as shown in Figure 5a.

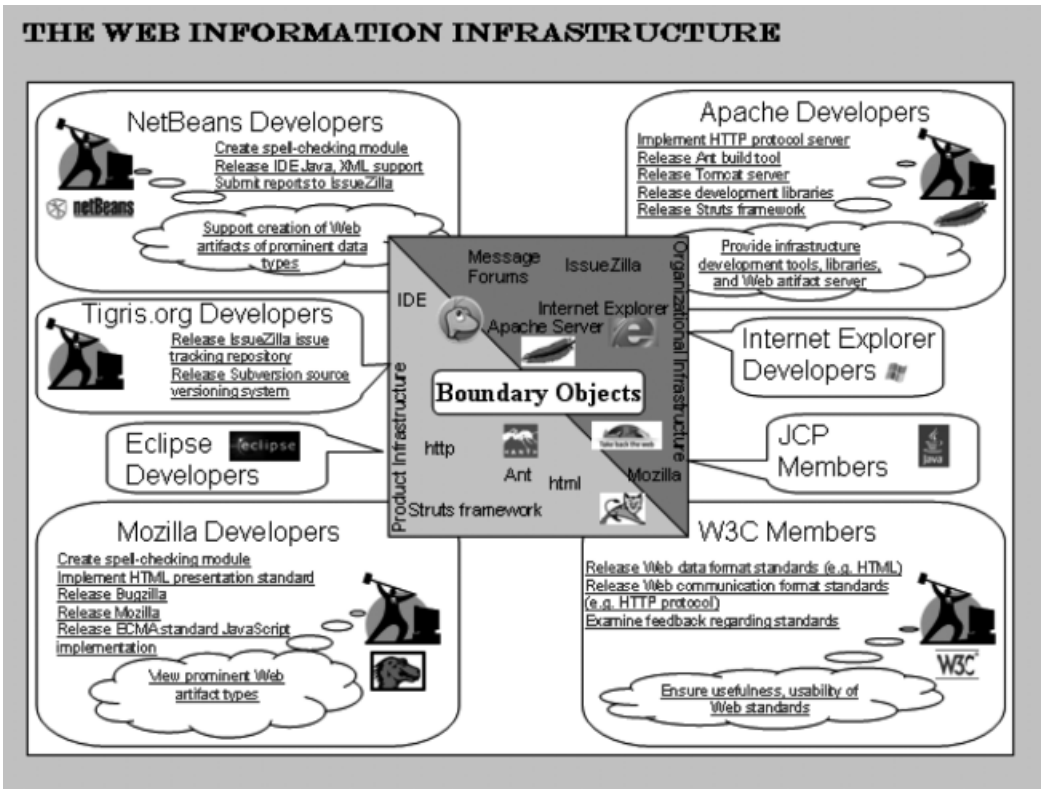
While certain boundary objects indicate a degree of interaction between processes in the Web infrastructure, it is yet unclear how this interaction plays out. As long as each member of the infrastructure adheres to these standards, they may choose to operate independently, following their individual processes as usual. However, the Web infrastructure is not a static network of interacting objects or a single coherent virtual enterprise. Commonly held standards change to meet evolving needs. Relationships between interacting software system developed by otherwise independent OSSD projects help adapt to infrastructure changes. Such relationships may require tighter coupling at the level of integration or explicit collaboration between organizational processes. By synchronizing their communication protocols and common data representations with one another through the process integration mechanisms of their choice, they stabilize the network. When an individual community varies from a standard or implements an update/revision to an existing standard, the other communities act to support it or choose to reject it. Likewise, defects in data representations or operations of one Web infrastructure software system can cause breakdowns or necessitate workarounds by the others. We look at the causes and negotiations of these conflicts next.

3.2.2. *Process Conflict*

Process conflict can precipitate or follow from process breakdown, disarticulation, or disintegration. Conflictive activities often arise from organizations competing for market share and control of the technical direction of infrastructure and shared technologies. It also arises from common and less belligerent activities, such as introducing a new version of a tool or database that other organizations depend on, requiring massive effort to incorporate. In these cases, the organization placed into conflict may simply choose to reject adopting the new tool or technology alterations, possibly selecting a suitable replacement tool/technology if the current one is no longer viable. This path was chosen by the shareware/open source image editing community infrastructure due to patent conflicts with the GIF image format in the 1990s, leading to the creation of the portable network graphics (PNG) image format standard[5].

[5] http://cloanto.com/users/mcb/19950127giflzw.html

(a)



(b)

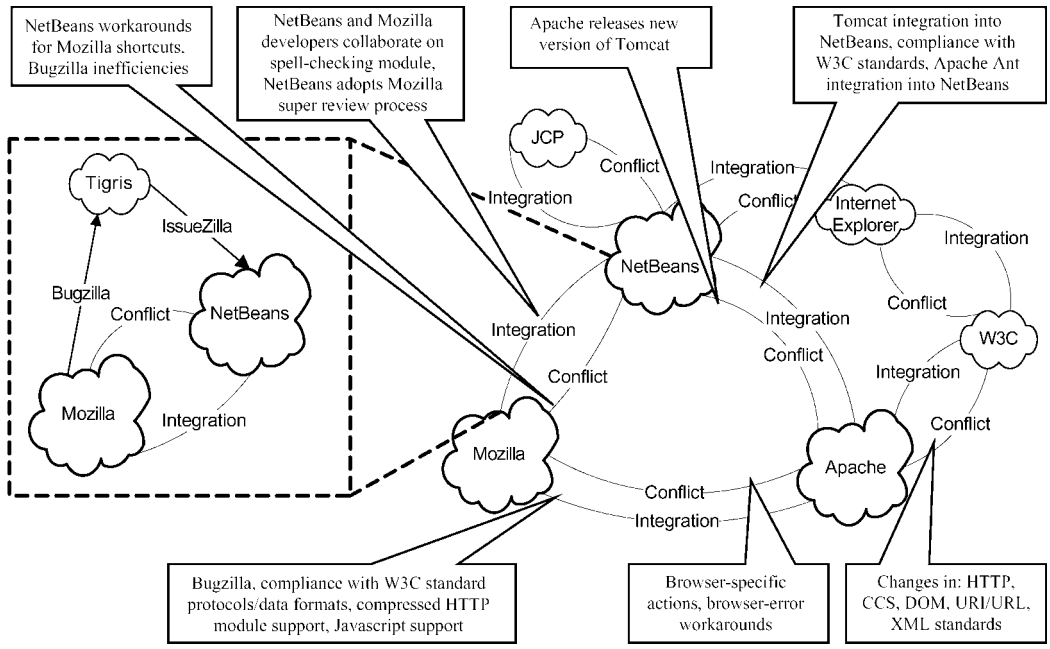


Figure 5. (a) Rich hypermedia modeling of processes spanning web information infrastructure projects. (b) Intercommunity interprocesses communication flow spanning Web information infrastructure projects. (c) Formal PML, (d) and reenactment models of processes spanning Web information infrastructure projects

(c)

```
Process Web Information Infrastructure Evolution (excerpt)
...
Sequence Mozilla Processes{   Action Release Bugzilla Defect Repository{ ...} ... }

Sequence Tigris.org Processes{   Sequence Create IssueZilla{
                        Action Download Bugzilla Sources{ ... }
                        Iteration Modify Bugzilla/IssueZilla Sources{ ... }
                        Action Release IssueZilla Issue Repository{...}
            }...
}...
Sequence NetBeans Processes{ ...
            Sequence Deploy IssueZilla issue repository{ ...          } ...
             Sequence NetBeans Development Process{
                        Action Submit Issues to IssueZilla{ ... }
                        Action Detect Inefficiency/Problems with IssueZilla{
                                    Requires { Issue reports }
                                    Provides { Problem description }
                                    Agents { NetBeans developers }
                                    Tools { HTTP Web browser implementation, IssueZilla deployment }
          Script {     <br><a href=http://qa.netbeans.org/processes/bug-handling-guidelines.html>Developers notice that IssueZilla
                      cannot track bugs across versions</a>
                      <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=333146>Developers
                      bring up the matter on community mailing lists/message forum</a> }
                        }
                        Action Determine Possible Workarounds/Solutions{
                                    Requires { Problem description }
                                    Provides { List of workarounds/solutions }
                                    Agents { NetBeans developers }
                                    Tools { HTTP Web browser implementation, IssueZilla deployment, developer, community discussion
message forums }
                    Script {     <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=389086>Discuss
                                workaround viability</a>
                                <br><a href=http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=331896>Discuss
                                workaround viability</a> }              }
```

(d)

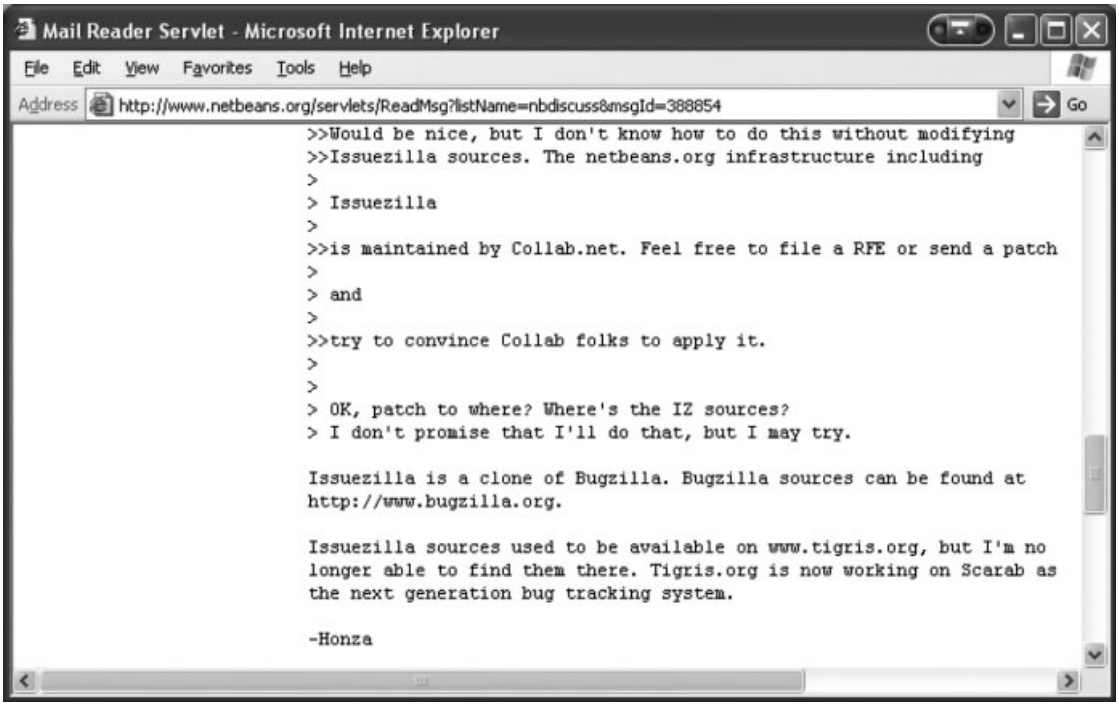


Figure 5. (*Continued*)

Conflicts across OSSD projects that get resolved are done so through collaborative means. Most typically, this occurs through the exchange of messages between participants (message threads) communicated on project discussion forums or other computer-mediated communication systems (e-mail, chat, instant messaging, etc.). Alternatively, an organization causing or resisting a tool or technology may succumb to pressure exerted by support from the rest of the infrastructure. Irreconcilable differences, if they persist and are strongly supported, can lead to unresolved conflicts (e.g. software updates that do not get implemented), incompatibilities in the interoperating software systems, or possibly to divisions in the infrastructure.

3.2.3. Discovering Interprocess Communication Across Web Information Infrastructure Projects

Though instances of direct communication between organizations within the infrastructure are visible (e.g. NetBeans and Mozilla developers collaborating on spell-checking module), indirect communication appears the more prevalent method. We see it in the form of version changelogs announcing support (and changes in support) for tools and technologies integrated into development. It may also appear in defect/feature request repositories, e-mail discourse, and community newsletters within the respective community Web sites, in addition to external news sources (e.g. slashdot.org and freshmeat.org). Communities must monitor these information sources to assess their degree of impact and whether the impact is directly or indirectly integrative or conflictive. NetBeans, for example, uses the IssueZilla bug/feature request repository developed by the Tigris community, which is, in turn, an extension of Mozilla's Bugzilla tool (see Figure 5b). Detecting indirect communications and their relationships to development process activities can be as simple as entering community specific terms, such as 'Mozilla', in the Search function input field on the NetBeans main Web page.[6] This action will search the NetBeans community Web site for instances of artifacts (e.g. project Web pages or forum postings) that contain the term Mozilla. The returned search results provide links to development artifacts or boundary objects associated with different development releases, bug reports, software module (in)compatibilities, or external (user) functionality assessments. In other cases, process communication can be nearly impossible to detect if no evidence is publicly available or is too circumstantial to validate.

3.2.4. Modeling Interprocess Communication Across Web Information Infrastructure Projects

Synchronization and stabilization of shared artifacts, data representations, and operations or transactions on them are required for a common information infrastructure to be sustained. This process is not 'owned' (Larsen and Klischewski 2004), located within, or managed by a single organization or virtual enterprise. Instead, it represents a collectively shared set of activities, artifacts, and patterns of communication that are enacted across the participating communities. Thus, it might better be characterized as an ill-defined, *ad hoc*, or one-off boundary-spanning process that differs in form with each enactment. Consequently, the form of these processes is dynamic and emergent, rather than static and recurring. Modeling such one-off processes thus must be justified, since they occur infrequently and do not reoccur. As such, our approach to modeling trades off representational detail of individual process forms, and instead uses a more abstract, low-fidelity representation (Atkinson *et al*. 2004). This is done so as to only model (or suggest) an abstract set of relationships of interaction, whose individual elements would be composed anew for each enactment.

Community communication channels (i.e. recurring patterns of communication of shared artifacts, data representations, or protocols) can be used to connect the interprocess resources flows between interoperating communities within the Web infrastructure. Each channel between communities connotes *ad hoc* processes that articulate the interoperability or interdependence of tools and technologies between them, as well as the boundary objects shared between them. The Web information infrastructure development process can therefore be characterized by the communication flow that enables integration or conflict process activities between constituent projects. Figure 5b illustrates some of the interorganizational relationships we have uncovered across the NetBeans, Mozilla, and Apache organizations, spanning many of the mechanisms of interorganizational interoperation described by Bluedorn and associates (1994), as

[6] http://www.netbeans.org/

well as the degrees of process integration outlined by Alter (1999) as a low-fidelity resource flow graph. Subsequently, these communication channels that enable interoperation and interdependence can be represented as a rich hypermedia, a low-fidelity resource flow graph, or as a low-fidelity formal process model, as shown in the examples of Figure 5. A narrative of the NetBeans IssueZilla issue tracking tool integration process depicted is these models follows.

3.3. Example: NetBeans IssueZilla Integration

The NetBeans community Web site is hosted by Sun Microsystems utilizing the commercially available collaborative development environment and Web portal software from Collab.net. Collab.net in turn utilizes the IssueZilla open source issue tracking system (sometimes also called 'issuetrack'). Thus, the NetBeans project community has a reciprocal trading relationship (see Table 1) – in this case a producer–consumer relationship – with the Tigris.org community. Tigris.org has a similar relationship with Mozilla. As such, any changes made, defects discovered, and documentation provided by Tigris.org and Mozilla may provide occasion for integration and conflict processes between NetBeans and the Tigris.org and Mozilla communities. One such occasion occurred in April of 2001. NetBeans developers noticed an inability to track issues across multiple versions and branches of the source tree[7]. This caused interorganizational coordination problems as they needed to know what technical problems existed in the previous version. In response to this and other less serious imperfections of Bug/IssueZilla system, NetBeans developers were forced to come up with workarounds until a new issue tracking system could be adopted. These workarounds involved storing the versioning information in heretofore-unused meta-data fields in each bug report, and implementing a transition plan for this new bug submission process. Unlike integration and conflict situations that arise from changes to the organizational network, the circumstances that created the need for a workaround led to a process adaptation within NetBeans that arose from an inability for existing network state to innately handle the changing needs of the NetBeans community. The adaptation process proceeded as follows.

Upon realizing the inadequacy of the issue repository, and following discussion thereof on the mailing lists[8], core developers posted an update to the bug-handling guidelines community Web page, announcing the nature of the problem and possible solutions. This was followed by discussion on the mailing lists of the desirability of the solutions presented[9]. Being an open source tool itself, some developers considered trying to modify the IssueZilla tool; however, development on it had ceased[10], the source from Bugzilla had long been forked[11], and the only available source versions were very complex[12]. In the end, the decision was made to alter the issue submission policy to use a previously unused field in the issue report to store the version-specific data as a stopgap solution, until a new defect repository was adopted. Though the inadequacy of the issue repository was reported in April 2001, the community still awaits the deployment of a new issue repository. Figure 5 provides rich hypermedia, process flow graph, and formal PML, and reenactment representations of this process.

4. DISCUSSION

The Apache, Mozilla, and NetBeans project communities are three prominent members of a larger organizational ecosystem that constitutes the Web infrastructure. In the space of software development, this ecosystem forms a development domain: the Web information infrastructure. Other prominent members of this ecosystem include OpenOffice.org, Tigris.org, Microsoft, IBM, the JCT, the W3C, and the Java Community Process (JCP). The three communities are the focus of our process modeling effort because they are developing large-scale software systems and related products through

[7] See: http://qa.netbeans.org/processes/bug-handling-guidelines.html

[8] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=333146

[9] http://www.netbeans.org/servlets/ReadMsg?listName=nbdev&msgId=79215

[10] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=388854

[11] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=389086

[12] http://www.netbeans.org/servlets/ReadMsg?listName=nbdiscuss&msgId=331896

complex processes that integrate efforts of tens of thousands of developers with millions of users. At the same time, the ecosystem is not static. The communities, and different OSSD projects within them, rise and fade from prominence. As they increase in mass (membership) and interconnectivity, they create a sense of both gravity and inertia around them, and other organizations may seek integrative relationships. While closed source projects tend to enjoy tightly coupled integration with relatively few counterparts, OSSD communities tend towards loosely coupled interoperability with many counterparts. The effect of this is that there are likely many more organizations impinging on the ecosystem with more complex, but weaker, bindings than those of proprietary system relationship networks, which may well be sparser and rely on more stable ties enforce through contractual arrangements or partnerships.

5. CONCLUSION

In this article, we described techniques and issues in modeling software processes used within three large and interdependent open source software development communities. The software developed in these communities form an information infrastructure for creating, serving, and consuming Web artifacts. We described an approach to modeling software development processes within and across these communities, as well as issues and trade-offs that arise along the way. Our approach draws attention to the need to model such processes using informal, semistructured, and formal process modeling techniques and representations, as well as to reenact them using a process simulator. We demonstrated how development processes within these communities interact in terms of ad hoc or fragmentary processes across communities through the direct or indirect flow of development artifacts found on each community's Web sites. This helps show the potential for the use of Web-based artifacts like rich hypermedia, resource flow graphs, and semantic web/hypertext models to capture and specify the processes of OSSD projects in the Web information infrastructure. We believe this promotes a more comprehensive, multimodel understanding of the processes rendered, as well as offering insights for the application of additional process improvement tools and techniques. The results presented here suggest a need for additional work in discovering, modeling, simulating, and analyzing interorganizational software processes. Our classification framework is rooted in a more traditional, closed source software development paradigm. But, are interorganizational relationships and their associated processes changing with the involvement of nontraditional software development organizations? Do these factors differ across software ecosystems? Finally, how can these lessons be applied to devising and improving interorganizational processes? Questions such as these denote issues to be addressed in follow-on studies that seek to model software processes that span independent projects in different organizations. After processes?

ACKNOWLEDGEMENTS

The research described in this report is supported by grants #0083075, #0205679, #0205724, and #0350754 from the US National Science Foundation. No endorsement implied. Contributors to work described in this article include Mark Ackerman at the University of Michigan Ann Arbor; Les Gasser at the University of Illinois, Urbana-Champaign; John Noll at Santa Clara University; John Georgas, Maulik Oza, Eugen Nistor, Susan Hu, Bryce Carder, Baolinh Le, Zhaoqi Chen, Veronica Gasca, Chad Ata, Michele Rousseau, and Margaret Elliott at the UCI Institute for Software Research.

REFERENCES

Alter S. 1999. *Information Systems, A Management Perspective*, 3rd edn. Addison-Wesley: Reading, MA.

Ata C, Gasca V, Georgas J, Lam K, Rousseau M. 2002. The Release Process of the Apache Software Foundation, http://www.ics.uci.edu/~michele/SP/index.html [10 January 2005]

Atkinson D, Noll J. 2003. Automated validation and verification of process models. *Proceedings of the 7th International IASTED Conference on Software Engineering and Applications*, Marina del Ray, CA.

Atkinson DC, Weeks DC, Noll J. 2004. The design of evolutionary process modeling languages. *Proc. 11th Asia-Pacific Software Engineering Conference*, Busan, Korea, 587–592.

Bluedorn A, Johnson R, Cartwright D, Barringer B. 1994. The interface and convergence of the strategic

management and organizational environment domains. *Journal of Management* **20**(2): 201–262.

Carder B, Le B, Chen Z. 2002. Mozilla SQA and Release Process, http://www.ics.uci.edu/~acarder/225/ [10 January 2005]

Choi JS, Scacchi W. 2001. Modeling and simulating software acquisition process architectures. *Journal of Systems and Software* **59**(3): 343–354.

Cusumano M, Yoffie D. 1999. Software development on internet time. *Computer* **32**(10): 60–69.

Elliott M, Scacchi W. 2003. Free software developers as an occupational community: resolving conflicts and fostering collaboration. *Proceedings of ACM International Conference on Supporting Group Work*, Sanibel Island, FL, 21–30.

Erenkrantz J. 2003. Release management within open source projects. *Proceedings of the 3rd Workshop on Open Source Software Engineering*, Portland, Oregon, 51–55.

Fielding RT. 1999. Shared leadership in the Apache project. *Communications of the ACM* **42**(4): 44–45.

Fielding RT, Whitehead EJ, Anderson KM, Bolcher GF, Oriezy P, Taylor RN. 1998. Web based development of complex information products. *Communications of the ACM* **41**(8): 84–92.

Fowler M, Scott K. 2000. *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, 2nd edn. Addison Wesley: Reading, MA.

Georgas J. 2002. *Software Process Modeling with Protégé*. University of California: Irvine, CA. 9 June, 2002. http://www.ics.uci.edu/~jgeorgas/ics225/index.htm [10 January 2005]

Halloran T, Scherlis W. 2002. High quality and open source software practices. *Proceedings of the 2nd Workshop on Open Source Software Engineering*, Orlando, FL.

Jensen C, Scacchi W. 2003a. Applying a reference framework to open source software process discovery. *Proceedings of the First Workshop on Open Source in an Industrial Context*, Anaheim, CA, 39–42.

Jensen C, Scacchi W. 2003b. Simulating an automated approach to discovery and modeling of open source software development processes. *Proceedings of ProSim'03 Workshop on Software Process Simulation and Modeling*, Portland, OR.

Jensen C, Scacchi W. 2004. Collaboration, leadership, control, and conflict negotiation in the NetBeans.org community. *Proceedings of the Fourth Workshop on Open Source Software Engineering ICSE04-OSSE04*, Edinburgh, Scotland, 48–52.

Larsen MH, Klischewski R. 2004. Process ownership challenges in IT-enabled transformation of interorganizational business processes. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, Big Island, Hawaii.

Lenz K, Oberweis A. 2001. Modeling interorganizational workflows with XML nets. *Proceedings of the 34th Annual Hawaii International Conference on System Sciences*, Maui, Hawaii, on CD.

Mi P, Scacchi W. 1996. A meta-model for formulating knowledge-based models of software development. *Decision Support Systems* **17**(4): 313–330.

Mockus A, Fielding R, Herbsleb J. 2002. Two case studies of open source software development: Apache and Mozilla, *ACM Transactions on Software Engineering and Methodology* **11**(3): 1–38.

Monk A, Howard S. 1998. The rich picture: a tool for reasoning about work context. *Interactions* **5**(2): 21–30.

Noll J, Scacchi W. 1999. Supporting software development in virtual enterprises. *Journal of Digital Information* **1**(4): http://jodi.ecs.soton.ac.uk/Articles/v01/i04/Noll/[10 January 2005].

Noll J, Scacchi W. 2001. Specifying process-oriented hypertext for organizational computing. *Journal of Network and Computer Applications* **24**(1): 39–61.

Noy NF, Sintek M, Decker S, Crubezy M, Fergerson RW, Musen MA. 2001. Creating semantic web contents with protégé-2000. *IEEE Intelligent Systems* **16**(2): 60–71.

Oza M, Nistor E, Hu S, Jensen C, Scacchi W. 2002. A First Look at the NetBeans Requirements and Release Process, http://www.ics.uci.edu/cjensen/papers/FirstLook-NetBeans/ [10 January 2005]

Pawlowski S, Robey D, Raven A. 2000. Supporting shared information systems: boundary objects, communities, and brokering. *Proceedings of the Twenty First International Conference on Information Systems*, Brisbane, Queensland, Australia.

Rasch RH, Hansen JV. 1997. A design approach for analyzing interorganizational information systems. *Annals of Operations Research* **71**(1): 95–113.

Scacchi W. 2000. Understanding software process redesign using modeling, analysis and simulation. *Software Process–Improvement and Practice* **5**(2/3): 183–195.

Scacchi W. 2002. Understanding the requirements for developing open source software systems. *IEE Proceedings – Software* **149**(1): 24–39.

Scacchi W, Jensen C, Noll J, Elliott M. 2005. *Multi-Modal Modeling of Open Source Software Requirements Processes*,

Proceedings of the 1st International Conference on Open Source Systems, Genova, Italy.

Scacchi W, Mi P. 1997. Process life cycle engineering: a knowledge-based approach and environment. *International Journal of Intelligent Systems in Accounting, Finance, and Management* **6**(1): 83–107.

Shen M, Liu D. 2001. Coordinating interorganizational workflows based on process-views. *Lecture Notes in Computer Science* **2113**: 274–283.

Star SL. 1989. The structure of Ill-structured solutions: boundary objects and heterogeneous distributed problem solving. In *Distributed Artificial Intelligence*, Vol. 2., Gasser L, Huhns MN (eds.). Pitman: London, 37–54.

van der Aalst WMP. 2002a. Inheritance of interorganizational workflows to enable business-to-business E-commerce. *Electronic Commerce Research* **2**(3): 195–231.

van der Aalst WMP. 2002b. Inheritance of interorganizational workflows: how to agree to disagree without loosing control? *Information Technology and Management Journal* **2**(3): 195–231.