Supporting collaborative development using process models: a tooled integration-focused approach[‡]

Komlan Akpédjé Kedji[1,*,†], Redouane Lbath[1], Bernard Coulette[1], Mahmoud Nassar[2], Laurent Baresse[3] and Florin Racaru[3]

[1]*IRIT, University of Toulouse, Toulouse, France*
[2]*Université Mohammed V - Souissi, Rabat, Morocco*
[3]*AKKA Technologies, Toulouse, France*

ABSTRACT

Collaboration in software engineering projects is usually intensive and requires adequate support by well-integrated tools. However, process-centered software engineering environments (PSEE) have traditionally been designed to exploit integration facilities in other tools, while offering themselves little to no such facilities. This is in line with the vision of the PSEE as the central orchestrator of project support tools. We argue that this view has hindered the widespread adoption of process-based collaboration support tools by incurring too much adoption and switching costs. We propose a new process-based collaboration support architecture, backed by a process metamodel, that can easily be integrated with existing tools. The proposed architecture revolves around the central concepts of 'deep links' and 'hooks'. Our approach is validated by analyzing a collection of open-source projects, and integration utilities based on the implemented process model server have been developed.

KEY WORDS: process-support; collaboration; tool integration

1. INTRODUCTION

Software engineering is usually a highly collaborative activity [1]. Moreover, the collaboration involved is often complex and has different aspects: task coordination, artifact change management, defect tracking, communication, and so on. The various efforts to support some specific aspect of collaboration have produced different formalizations of collaboration. Every collaboration support tool embodies some idea, some formalization, some knowledge, and some 'model'[§] of a software project. The harmonious integration of all such models is essential for supporting collaborative work [2, 3].

However, process-centered software engineering environments (PSEEs), which manipulate process models, have traditionally been developed with design assumptions that hurt their integration capabilities and therefore their adoption.

On the one hand, PSEEs, as evident in the name PSEE, assume that the whole working environment revolves around the process model [4]. It is therefore not surprising that most PSEEs expect that activities are launched and ended inside them by clicking buttons (that is, these actions need not be

*Correspondence to: Komlan Akpédjé Kedji, IRIT, University of Toulouse, Toulouse, France.
†E-mail: kedji@univ-tlse2.fr
‡This is an extension of a paper published in the proceedings of the ICSSP 2012 Conference [26].

automatically invokable from a third-party tool), that they take care of launching tools, and that almost all the support they provide is by native components of the PSEE, not by third-party tools through an extension mechanism.

Such process-centric view of the software project marginalizes process-based tools because it leads to high adoption costs, as well as high switching costs. Adopting a PSEE means giving it total control of the development setup, which is most of the time unrealistic, as other tools were probably in use before the introduction of the PSEE. The integration between PSEEs and external tools is also usually performed by PSEE-specific configuration procedures and not through generic mechanisms such as environment variables, composable textual interfaces, Hypertext Transfer Protocol (HTTP) APIs, and so on. This means existing tools cannot be easily retrofitted to communicate with the PSEE and makes switching between different PSEEs a major hurdle.

On the other hand, control and data integration between a PSEE and other tools is usually one-way [2, 5]. The PSEE can invoke other tools or detect that they have been invoked (control integration) and read the artifacts created using another tool (data integration). The flow of information goes from the external tool to the PSEE and is at the initiative of the PSEE, not the reverse. Unfortunately, unlike existing collaboration solutions, PSEEs make it difficult for other tools to integrate with them, as they do not offer a simple way to be invoked or queried.

For example, Abriola *et al.* [5] lament the fact that some external tools do not expose enough data and control so that the PSEE can control them but do not point out that the PSEE itself does not expose its data nor does it provide control integration mechanisms. Even the surveyed and recommended data integration approaches in [5] assume that the PSEE should know about editors but not the reverse. This is actually not surprising, as the PSEE usually does not offer any simple facility to read its data or control it. Things would have been different if the PSEE did not act as *the responsible* of control integration in the workspace but as a *participant* like any other tool.

We argue that the idea of the process model being the most important representation of a software project and therefore must be the foundation of tool support is misguided. It either leads to PSEEs trying to support every aspect of a project or considering every other aspect as secondary to the process followed. This has unfortunate design implications that hurt the adoption of PSEEs. A bug model, for example, is an abstraction of software defects and people responsible for them. Similarly, a process model can be considered as just another abstraction, which happens to focus on activities, the people carrying them out, and the artifacts they produce and consume. Process-support tools can thus be designed, such as bug and artifact tracking tools, to provide core services related to activities and *most importantly* to expose this model so that it can be integrated with other models of the software project. Such design can be used to easily provide context for developments activities by linking to their definition in the process model or automatically reflect process events on the development environment and vice versa.

Our contribution is, on the one hand, additional concepts and relationships for the representation of collaboration in process models. On the other hand, we propose a mechanism for the integration of process-support tools with other collaboration support tools, which follows the Unix principle of loosely coupled tools, each fit for a narrow purpose. Indeed, supporting a software project requires to have appropriate information on the ongoing project. While traditional process models can provide information on *what is planned*, additional details are needed to be able to use a process model as an information source on *what is happening*. We propose Collaborative Model-based Software and Systems Process Engineering Metamodel (CMSPEM), a metamodel that augments SPEM with the ability to describe project-related concerns and an engine that can participate in collaboration support using information extracted from CMSPEM models.

The rest of this paper is structured as follows. Section 2 describes and conceptualizes some integration capabilities in the collaboration support tools that we would like to integrate process-based tools with, and Section 3 investigates how information in process models can contribute to collaboration support. Section 4 describes how our metamodel captures that information, Section 5 presents the implementation of our process-support approach and its application to an example from

the industry, and Section 6 describes a statistical study conducted on a collection of open-source projects to validate the approach and discusses the implementation of some integration utilities. Section 7 summarizes related works, and Section 8 closes with a conclusion and perspectives of our contribution.

2. INTEGRATION CAPABILITIES IN EXISTING COLLABORATION SUPPORT TOOLS

If we want to integrate process-based tools with other collaboration support tools, we need to understand their integration capabilities and which design decisions enable such capabilities. Bug trackers, version control systems (VCSs), communication tools (chat, mailing list, etc.), build tools, and knowledge centers (websites, wikis, etc.) are some common collaboration support tools used in software engineering projects [6]. This section investigates the integration capabilities of two popular tools (bug trackers and VCSs), identifies which design decisions enable those capabilities, and proposes a conceptualization of such design decisions.

2.1. Examples of integration capabilities

Bug trackers are based on a model of the defects in software products, their resolution states, their severity, their interdependencies, and participant responsibilities with respect to them. This model is used by the bug tracker and third-party tools to offer a set of collaboration support tools such as email notifications on bug status changes, duplicate bugs identification, release engineering (which bugs should be resolved for the next release), quality control (which parts of the code exhibit the most bugs and which participants usually introduce bugs), and so on.

Version control systems are another example of collaboration tools based on a model of the software project. A VCS works with a model of the artifacts in a software project, their modification history, changes made to them and the participants who made those changes, identification and manipulation of product states (with tags and branches), and so on. The VCS implements the modification of such model and the resolution of potential problems during such modification such as artifact merging. However, much of its usefulness lies in the rich model that it allows third-party tools to query. The model can be directly queried, for example, to find out the last time a file was changed, by whom, why, and so on. Tools can also take actions on specific events such as after a commit, before a merge, after receiving code in a remote repository, and so on.

The previous examples show that existing tools concentrate on a core problem, solve it, and expose the model of the specific project aspect related to the problem they solve. They are therefore useful beyond their core competency and can easily participate in the ecosystem of collaboration support tools. These tools do not assume that they are the only ones used in a project, that they are the entry points of every action, or that the information they have on the project is only useful for them. The design of such tools recognizes that collaboration concerns are interlinked and automation can only deliver on its promises when different automation solutions can be automatically linked, so that, for example, a commit in a version control tool can automatically trigger a bug resolution state change in the bug tracker. Without such automatic connections between tools, practitioners waste time with manual bookkeeping tasks. They have to manually search one tool for information that gives context to data in another tool or must manually instruct one tool to react to events generated by another tool.

2.2. Conceptualization

The analysis of the integration capabilities of existing collaboration support tools highlights two important integration mechanisms: deep linking for data integration and hooks for control integration.

2.2.1. Deep linking. Popular software engineering tools for version control, bug tracking, continuous integration, documentation, communication, and so on usually make their data available over the network and exploit the resource linking facilities provided by the Web, that is, hypertext links. This makes it possible for a commit in a VCS to refer to the bug it fixes or for a discussion on Internet

Relay Chat to directly refer to a build failure in a continuous integration tool. This easy linking requires tools to support 'deep linking' [7], that is, the ability to address a single item of the data managed by a tool (a particular build, a particular commit, a particular comment on a bug, etc.). Deep linking only requires that a global identifier is assigned to each data item that can be deep linked to, by using a well-known addressing scheme.

2.2.2. Hooks. Integration problems can be approached by creating new interoperability standards. However, this results in an all-or-nothing situation, where tools that use the same standard communicate with no additional effort, and those that do not, cannot communicate. Another approach is to design tools so that simple glue scripts can be written to make them communicate with little effort. This requires the definition of 'hooks' [3, 8], which are specific events that can occur in a tool. Each time they occur, hooks trigger the execution of the glue scripts defined for them. For example, hooks are used in continuous integration systems to announce build failures in company chat rooms. They are also used in VCSs to run tests and trigger deployments after commits. Hooks are actually lightweight event notification mechanisms, where listeners can be defined by writing scripts that are stored in a conventional location,[¶] and event notifications are sent by executing the scripts defined as listeners.[‖]

3. PROCESS-RELATED COLLABORATION SUPPORT IN SOFTWARE ENGINEERING

3.1. Generalities

This section discusses a set of collaboration support facilities that can be provided using a process model. These facilities are illustrated on an industrial software development project case study. The understanding gained by analyzing and conceptualizing the integration capabilities of existing collaboration support tools in Section 2 suggests that such opportunities come in two groups:

- Core opportunities: support based only on pure process considerations. Such considerations involve roles, products, and activities and their relationships, regardless of actual role affectations, product contents, or domain-specific details of activities.
- Transversal opportunities: support based on the intersection of process aspects with other development aspects such as bug tracking, version control, communication, and so on. Such development aspects, while usually acknowledged in process models, are typically modeled in detail and stored outside of the process model (bug tracking database, version control metadata, communication tool settings and history, etc.).

Process models are, fundamentally, project abstractions that focus on the activities to be carried out, their properties and relationships, as well as their parameters such as the people assigned and the artifacts consumed and produced. Such abstractions enable core support scenarios such as enforcing preconditions and post-conditions on activities, or tracking the status of each task. These concerns have largely been addressed by existing PSEEs.

One should however note that the core support opportunities for process-based tools are mostly about managing work. They are helpful to the manager interested in the global progress of the project or the quality manager interested in how much the organization is sticking to recommended work practices. However, for developers working on specific tasks, the part of the process they are working on is a guide and as such is more useful when it gives context to their daily activities such as design, coding, bug fixing, release engineering, browsing documentation, and so on. Arguably, the sometimes difficult adoption of software processes and associated tools has some roots in the poor integration between such tools and daily developer preoccupations.

[¶]For example, a hook that must be executed in the *git* version control system, before each commit, must be an executable script named .git/hooks/pre-commit.

[‖]Event parameters are usually transmitted with general purpose mechanisms such as environment variables or HTTP request parameters.

For developers, most support opportunities require a synergy between process-based tools and other collaboration support tools. One important consideration is the ease of access to contextual information [9]. Fixing a bug, for example, requires access to documentation (programming language and library tutorials, requirement documents), bug tracking information (bug status and comments), version control information (the date when the bug was introduced), test results (the tests that are broken by the bug), communication tools (relevant discussions on a chat or a mailing list), and release engineering information (systems running a version affected by the bug and the release that the bug fix is scheduled for). Productivity is improved when developers can have fast access to such context without needlessly interrupting each other, thus making collaboration frictionless [10, 11].

A process-based tool can contribute valuable contextual information such as the activities involved in the creation and resolution of the bug (with their description, duration, and preconditions and post-conditions), the roles and people who participate in such activities, the artifacts produced and consumed by such activities (which may also need to be modified or which offer additional insights on the bug), which deadlines one must comply with, and so on.

Another consideration is the automation of the reaction of one tool to an event that occurs in another tool. If a process engine makes its internal events available to other tools, some manual procedures can be eliminated. For example, deadline reminders can be automatically sent to bug discussions threads. Activity completion notifications can trigger test executions or builds. A work product state change, from 'document completed' to 'document validated', for example, can lead to the creation of a tag in a version control tool. The manual bookkeeping work that practitioners are usually required to do, to reflect process constraints on their work, will be reduced.

3.2. An industrial case study

Our case study is part of the French ANR Galaxy project, whose goal is to provide collaboration support infrastructure for the development of complex systems using the model-driven engineering approach.

3.2.1. High-level overview. The case study is based on the development practices of the 'Software and Systems' pole of AKKA Technologies, a European engineering and consulting firm, with offices in over 10 countries, a headcount of over 7000 engineers, and expertise in the aeronautics, railway, defense, space, and automobile sectors.

Each project follows a high-level division into core and transversal activities,** as shown in Figure 1.

Sub-activities of the 'Develop' activity include specification, conception, implementation, testing, delivery, and support. Depending on the development process chosen for a particular project, these activities may overlap in time or be repeated as much as necessary.

The description of these general activities includes their objective, input and output products, result control procedures, realization hints, references of supporting documents, and roles involved.

3.2.2. Integration opportunities. Let us consider the example of a project that, similar to the development work carried out at AKKA Technologies on the Galaxy project, uses Git for version control, Eclipse as editor, BugZilla as bug tracker, and Microsoft project for project management. Project participants are constantly switching between these applications, each supporting a particular aspect of collaboration.

If an explicit process model, backed by a PSEE, is to be used, much of its usefulness for developers lies in how tightly it is integrated in the existing setup.

For the needs of the case study, we focus on the 'Change management' activity, which consists in fixing the defects reported by the client after the product has already been deployed or enhancing the product with new functionalities. Tickets (enhancement requests or bugs reports) are created in BugZilla and assigned to a group of developers. Each developer is assigned a number of tasks (TaskUse in SPEM), each task being composed of a number of tickets. Each task is considered as completed when all the tickets it is made of are solved.

**This is not related to the core and transversal opportunities discussed in Section 3.1.

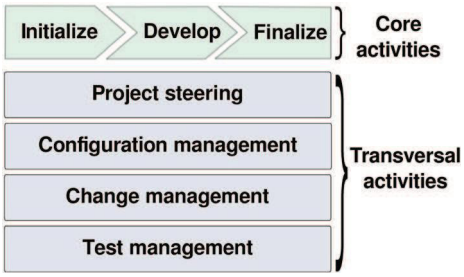

Figure 1. Main activities of the AKKA generic process.

When dependencies are discovered between tickets, they are added to the process model as precedences. For example, if solving a ticket requires the resolution of another ticket, a finish to finish precedence is used. Sometimes, ticket dependencies are discovered when investigating bugs and added to the process model as task dependencies. A bug can also be discovered to be the symptom of separate defects, in which case the bug is split, and newly created bugs are assigned. While investigating individual defects, developers regularly need context information on the task and version control revision in which it was introduced, as well as people that are most likely to provide valuable information about such bugs.

This sample situation exhibits a couple of integration opportunities between the process-support tool and other tools. For example, tasks can be represented in the process-support tool, and each task can link to the tickets in Bugzilla, which must be fixed in the task. Whenever a particular bug changes state, the process-support tool can be notified, and if appropriate, mark the task as completed. As such notification includes the identity of the developer who fixed the bug, a third-party tool, integrated in the process-support tool, can generate a weekly progress report for each developer. A utility can also monitor bug report creations and automatically provide process context information for participants and tasks referenced in the report, by posting comments for example. Section 6.2 discusses the implementation of such utilities.

4. MODELING COLLABORATIVE PROCESSES

A PSEE or any process-based collaboration support tool is based on a model of the software project that focuses on process aspects. To enable the creation of such model, we defined CMSPEM, a metamodel for the description of collaborative software processes.

4.1. Collaborative Model-based Software and Systems Process Engineering Metamodel: a metamodel for collaborative processes

The CMSPEM is an extension of SPEM2 [12].

Our extension is based on the insight that in an ongoing software engineering project, where issues such as collaboration are manifest, the concepts involved for the practitioners are the actual people doing the job, what each of them is doing, and which artifacts they are manipulating. However, process models are usually described using roles (a role may be played by different people, and someone may play different roles), products (a product such as a source file may have different physical representations in different workspaces), and tasks (a task may be carried out by different people, each focusing on a specific part of a product). We introduce concepts to account for these precisions, and describe how they relate to each other.

The CMSPEM introduces the concept of *Actor*, a specific human participant in a project. This addition not only allows a better description of collaboration but also can be easily linked to a commit author, a bug reporter, or a chat room participant.[††] This makes it easy to integrate process

[††]The promotion of concepts such as Actor to the metamodeling level is one way to address the problems with the Object Management Group (OMG) metamodeling framework, which arise when ontological and linguistic concepts need to be mixed. See [27] and [28] for a complete description and other solutions.

data with data manipulated by other tools. For example, a commit in a VCS can be linked to a task in the process-support tool using the author of the commit (an Actor in CMSPEM), which may be identified by his/her email address. CMSPEM also introduces the concept of *ActorSpecificWork*, which is a unit of work performed by a specific Actor in the context of a task (TaskUse in SPEM), and the concept of *ActorSpecificArtifact*, which is the personal copy of a WorkProductUse (from SPEM), in the workspace of a given Actor.

Figure 2 is a succinct representation of the metamodel that focuses on collaborative aspects. The three central concepts are linked to the SPEM concepts (blue/shaded) they add precision to. For example, ActorSpecificWorks detail what each project participant does in a task, and Actors detail which project participants play a specific role.

The three central concepts can be related to each other using relationships that capture a particular aspect of collaboration. For example, an *ActorSpecificWorkRelationship* can be used to specify the precedence relation between two ActorSpecificWorks, and *ActorRelationships* can be used to materialize the fact that a given Actor reports to another. Relationships are also used to denote assignment or possession. For example, a *TaskAssignment* is used to assign an ActorSpecificWork to an Actor, and an *ArtifactOwnership* precises the owner (an Actor) of an ActorSpecificArtifact.

In our approach, CMSPEM models can evolve at enactment time, in response to the availability of new information [13]. Such changes to a CMSPEM process model, such as the addition of a new actor, generate events (see [14] for a complete discussion). This allows third-party tools to listen to specific events on the process model and react to them. Events in CMSPEM enable control integration with other collaboration support tools. For example, when an activity changes state in a CMSPEM model, it can be automatically announced on the company's chat room or mailing list. A detailed description of CMSPEM is available in [15, 14], and the semantics of the CMSPEM concepts have been formalized in Object Constraint Language.

4.2. A domain-specific language for the description of collaborative processes

We developed CMSPEM DSL, a textual domain-specific language (DSL) based on Eclipse, by using the XTEXT framework. It offers an intuitive syntax for representing CMSPEM process models, code organizations facilities such as packages, and configurable visualizations that can be used to highlight particular relationships such as task dependencies. The DSL editor provides extensive, on-the-fly model checking, using consistency rules written in Object Constraint Language. Models created with the DSL editor can be exported in the XML Metadata Interchange format.

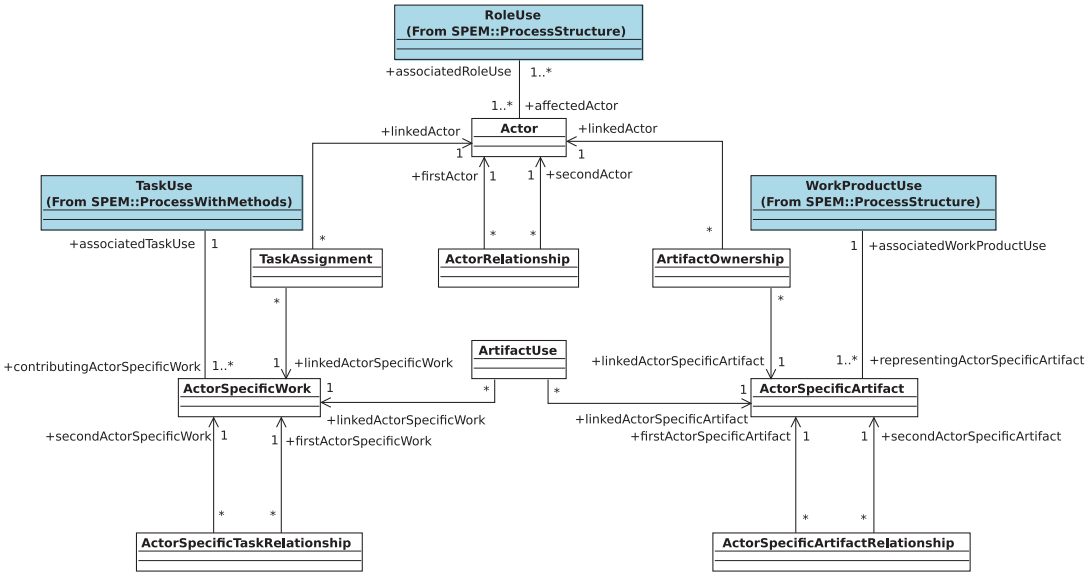

Figure 2. Main concepts of the CMSPEM metamodel.

A graphical editor, based on the TOPCASED [16] environment, is already available for editing CMSPEM models [14]. One of the motivations for using a textual DSL to represent CMSPEM models is that it makes it easy to generate part of a CMSPEM model from a third-party tool with simple text manipulations. For example, a CMSPEM model can be kick-started by generating Actor instances using the information in an organization's Lightweight Directory Access Protocol directory. Another motivation is the availability of the Eclipse platform's refactoring and code navigation facilities, which are very helpful when editing large models.

4.3. Modeling the case study

The case study easily lends itself to a representation in CMSPEM. Each developer is represented as an Actor and logical groups of defect tickets (for example, usability problems) as ActorSpecificWorks. Defect tickets are referenced by their Bugzilla identifiers in the definition of ActorSpecificWorks.

Figure 3 shows a part of the model, represented using CMSPEM DSL. Most concepts are defined using a similarly named keyword: for example, Actors are introduced with the actor keyword and ActorSpecificWorks by the asw keyword. Defect tickets are included, with the tickets keyword, as properties of the ActorSpecificWorks they must be fixed in, using their identifiers.

In this example, Laurent is defined as the project manager, Florin as a developer, and Jacques as a front-end designer. An ActorSpecificWork that consists in fixing a set of synchronization bugs is defined, with the related defect tickets listed by their numeric BugZilla identifier (31045, 31047, and 31050) and assigned to Florin the developer.

5. IMPLEMENTATION

5.1. A conceptual framework for process support

The Unix philosophy championed the use of pipes, filters, and short shell scripts as a lightweight integration solution between independently developed programs that operate on plain text data. However, such approach is only appropriate for integrating local tools or remote tools for which a local client is available. For the integration of possibly distributed tools, a solution that can work over a network is needed.

We propose a distributed conceptual framework for editing and exploiting process models, which enables deep linking and hooks (cf. Section 2.2) and is depicted in Figure 4. At the center of this

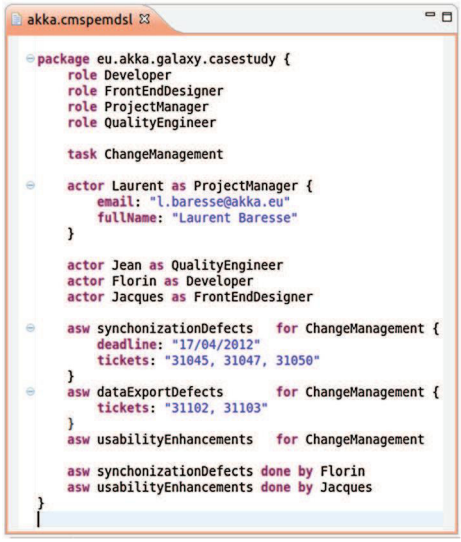

Figure 3. Eclipse-based domain-specific language for editing Collaborative Model-based Software and Systems Process Engineering Metamodel models.

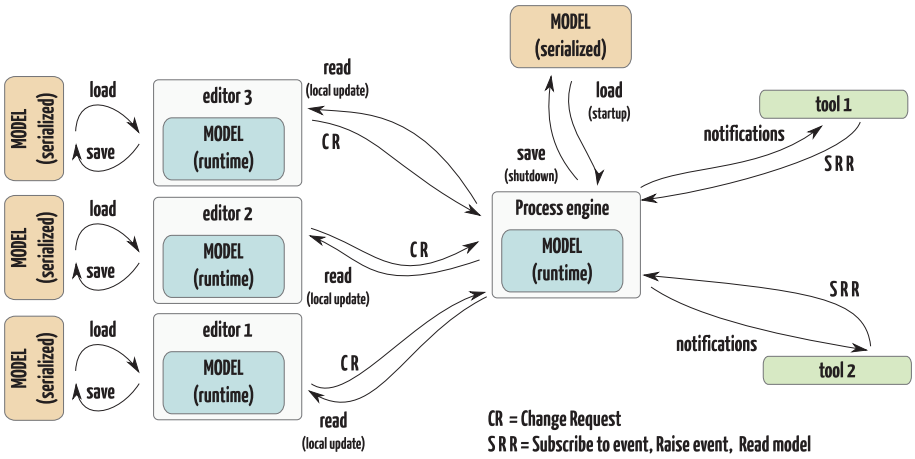

Figure 4. Process-based collaboration support conceptual framework.

framework is the CMSPEM process engine, a server that stores the authoritative version of the process model and manages process modifications and event notifications. Process model editors and external collaboration tools can update, query, or receive notification from the process model, through the process engine.

The CMSPEM engine assigns a globally unique identifier to each process model element, which allows deep linking to particular activities, roles, and so on. One should therefore be able to refer to a particular activity, by simply including a link to it in a commit message, or a mailing-list discussion. This use of deep linking addresses data integration needs.

Process editors can connect to the engine and send model change requests. This decouples model editing from updating the model in the process engine. The engine receives change requests and runs consistency checks before accepting them and updating the reference version of the model. Editors update the model by making API calls to the CMSPEM engine. The process model can also be serialized[‡‡] by model editors for local persistence or to guard against power failures on the server. In the architecture depicted in Figure 5, the process modeling tool can be a process editor, such as the visual CMSPEM editor that has been developed with TOPCASED.

To address control integration needs, the CMSPEM engine allows external tools to subscribe to process events. Each update to the process model on the server generates events, which external tools can be notified of, provided a matching subscription exists. Such events, such as a state change in an activity or the addition of a new actor to the process model, are hooks for which integration scripts can be written. The CMSPEM engine is responsible for invoking these scripts when the relevant events occur. An external tool can also raise events on the CMSPEM model, which other tools or the CMSPEM engine itself can react to. For example, a bug tracker can be configured to raise a ‘bug marked as duplicate’ or a ‘bug split’ event. Such events can be used to automatically update the model (by marking some ActorSpecificWorks as completed) or delivered as notifications to the project manager who decides how to update the process model. When a dependency is introduced between two bugs in the bug tracker, it could trigger the addition of a relationship between the corresponding ActorSpecificWorks in the process model.[§§] Similarly, an editor or a VCS could notify the CMSPEM engine of the availability of a new artifact. The CMSPEM engine can then react by introducing a previously configured relationship between the new artifact and an existing one (composition, refinement, etc.).

[‡‡]Serialization is the translation of data structures or object states into a storable format, which can be later retrieved and translated back.

[§§]Most external tools such as bug trackers already support hooks. For example, BugZilla refers to them as extensions, and they are Perl scripts that can raise events by making requests to the CMSPEM server. By using HTTP, for example, a POST request to https://www.example.com/cmspem/events, with a form parameter named event set to bug split (and other form parameters such as the original bug’s id etc.) can be issued by a BugZilla extension to notify that a bug has been split.

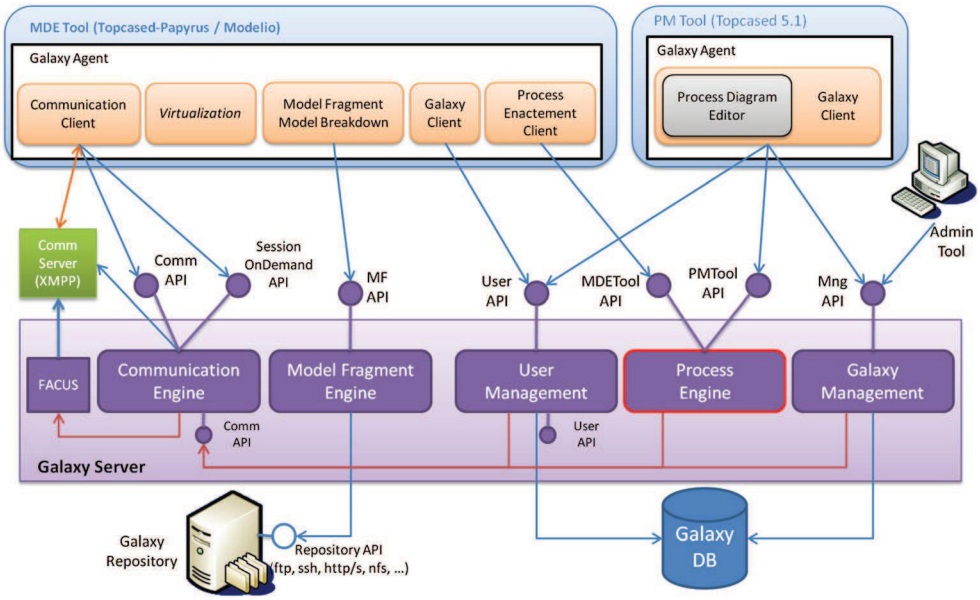

Figure 5. The Galaxy framework.

5.2. *Supporting the case study needs with tools*

The abstract framework described in the previous subsection can be realized using existing communication protocols. For example, HTTP has proved itself a robust interoperability solution for distributed systems or even between desktop tools (such as the Eclipse editor in Jazz [11], or process modeling tools) and web-based tools. This is largely due to the ubiquity of the Web and its firewall friendliness [17, 2]. HTTP already has extensive support for defining and linking to Uniform Resource Locators (URLs), which can be mapped to process elements, thus making such elements easy to address and link to. Notifications can be implemented with Web hooks, which allow systems with HTTP endpoints to notify each other of specific events. For example, Web hooks are used in continuous integration systems to announce build failures in company chat rooms. They are also used in VCSs to trigger deployments after commits.

With this setup, simple solutions to the integration needs of the case study are possible. Linking to BugZilla tickets is simply a matter of including BugZilla ticket URLs or numerical identifiers in the description of the tasks (SPEM TaskUse) or individual ActorSpecificWorks (cf. Figures 2 and 3).

Each ActorSpecificWork can be addressed by its own URL in the CMSPEM engine. Such URLs are actually HTTP endpoints, which can be used as Web hook URLs by external tools. Thus, BugZilla can, for example, be configured to send an HTTP request to such URL whenever the status of a bug changes. Upon receiving such notification, the CMSPEM engine could execute its own logic to decide if the enclosing task can be marked as completed or not. This discussion voluntarily leaves security considerations out, but incoming requests to the CMSPEM engine can be encrypted and authorization and authentication schemes included as needed.

Any tool using information that could be found in or derived from process data and events can benefit from integration with the CMSPEM engine. Such information could be task status (dashboards), task assignments (performance reports), activity starts and stops, or major changes to artifacts (broadcasts in various awareness and communication tools), actor availability (resource management), and so on.

The CMSPEM process engine is part of a broader framework, the Galaxy framework, which has been developed in the Galaxy project. The Galaxy framework (cf. Figure 5) supports model fragmentation and virtualization, user management, communication, and process enactment. The framework is composed of the Galaxy server, which offers collaboration facilities for a set of software development projects, and clients such as model-driven engineering tools (Modelio [18] or Papyrus [19] in the Galaxy project) and process management tools.

6. VALIDATION STUDY

The goal of this section is the description of the validation study carried out for our proposal. The study addresses the conceptual proposal (deep links and hooks) and its practical implementation in a process server.

On the one hand, we demonstrate how deep linking and hooks are used to structure existing software projects, by analyzing their mailing list messages. This demonstrates the suitability of those concepts to the integration of software engineering tools.

On the other hand, we show how the introduction of process data can make process support more useful and efficient, by discussing the implementation and benefits of some utilities enabled by such introduction. Those utilities have been identified in the case study of Section 3.2.

6.1. Usage of deep linking and hooks in software projects

To validate our conceptual proposal for the integration of process-support tools in software engineering environment with deep links and hooks, a study was carried out on a collection of open-source projects. The project list was compiled by searching the internet for mailing list archives available for download. Of this initial set, we restrict ourselves to the 219 most active ones[¶¶] as needed for the study. The resulting set contains major open-source projects such as PostgreSQL, QEMU, Emacs, Python, The Linux Kernel for ARM, Mythtv, FreeBSD, and LLVM. These projects each have hundreds of contributors and are thus appropriate for an analysis of collaboration.

6.1.1. Deep linking. Most of the discussion about open-source projects happens on project mailing lists. Theses discussions not only reveal how participants collaborate on the project but also show how they exploit software engineering tools to do so. More specifically, one can analyze references made by project participants to other tools. In this study, we restrict ourselves to hypertext links to precise pieces of data in software engineering tools, that is, deep links.

Linking to other resources is pervasive in mailing list discussions. Over the 219 studied projects, there are on average 0.93 link per message, and it can get as high as 15 links per message. As much as 70% of all links are deep links; that is, they refer to a precise piece of information in an external system.

Deep links point to a variety of project resources. PostgreSQL is the biggest project studied with 247,489 messages over 14 years (between January 1997 and May 2012), containing 120,325 links of which 17,873 are unique (0.48 link per message). The content of some link targets can be inferred from the structure and the keywords of the link's URL. On a subset of 22,623 links for which such inference can be done, the link distribution in the PostgreSQL project is as shown in Table I.

The data in Table I clearly show that deep links to documentation, mailing lists, and version-control data are the most frequent. In particular, it highlights how the availability of links to specific version control information items (commit, file at a specific version, pull requests, etc.) allows collaboration discussions to be contextualized. One can also note the high number of link repetition ratio for references to third-party bug information. Those are generally bugs in software (such as an operating system or a library) that PostgreSQL depends on, which requires the PostgreSQL developers to follow the evolution of such bugs, so they can modify PostgreSQL accordingly. This would not have been possible if the third-party bug trackers did not expose deep links to bug information.

6.1.2. Hooks. Some special project mailing lists are simple examples of using hooks for announcements. Typical examples are new commits, new bug reports or changes to bug status, and automatic build or deployment reports.

The PostgreSQL project, for example, has a special mailing list where CVS and Git commit notifications are sent (http://archives.postgresql.org/pgsql-committers/). Each notification message contains a short description, an expanded version, the branch name, a summary of changes (modified files), and most importantly, a deep link to more details about the commit, made available by the version control software (Figure 6).

[¶¶]Project activity is estimated by the number of mailing list messages and the age of the project.

Table I. Link distribution on the PostgreSQL main mailing list.

Link type	Link count	Unique link count	Link repetition ratio	
Industry standards	25	14	1	78
PostgreSQL sub-projects	64	36	1	77
Third-party bug trackers	73	5	14	60
Shared code snippets	102	25	4	08
Third-party version-control data	106	70	1	51
Internet forum posts	194	128	1	51
Developer home pages	292	93	3	13
Research papers	339	93	3	64
Third-party project mailing lists	407	254	1	60
PostgreSQL bugs	515	267	1	92
PostgreSQL builds	729	292	2	49
Encyclopedia, dictionaries, references	733	328	2	23
PostgreSQL software downloads	734	270	2	71
Third-party documentation	876	441	1	98
Third-party software	1306	616	2	12
PostgreSQL version-control data	1651	586	2	81
PostgreSQL's other mailing lists	3002	1396	2	15
PostgreSQL's main mailing list	5728	2398	2	38
PostgreSQL documentation	7074	1870	3	78

```
Fix syslogger so that log_truncate_on_rotation works in the first rotation.

In the original coding of the log rotation stuff, we did not bother to
[snipped]

Branch
------
REL9_2_STABLE

Details
-------
http://git.postgresql.org/pg/commitdiff/63aba79c7f1f06422b22e2b44fdcb563bbc3f7a5

Modified Files
--------------
src/backend/postmaster/postmaster.c |    6 ++++-
src/backend/postmaster/syslogger.c  |   43 ++++++++++++++++++++++++++----------
2 files changed, 35 insertions(+), 14 deletions(-)
```

Figure 6. Example of a commit notification on a mailing list (PostgreSQL).

6.2. *Improving tool support with process information*

The CMSPEM server is an implementation of the CMSPEM engine described in Section 5. It makes a simple process model manipulation API available over HTTP. The server has been developed in Java, using the Play framework (http://www.playframework.org/), Eclipse EMF model manipulation libraries (http://www.eclipse.org/modeling/emf/), and EMFJSON (https://github.com/ghillairet/emfjson) for EMF to Javascript Object Notation (JSON) conversion. For the purpose of the validation study, the server is deployed at http://cmspem.herokuapp.com and exchanges data with clients using JSON. JSON is a data representation format commonly preferred for the integration of Web-based systems for its simplicity and easy interpretation by Web applications (as it can be natively decoded in Javascript).

The CMSPEM server enables three main use cases:

- Querying the process model for process information. For example, a request to /projects/XXX/asworks/YYY returns data (attibutes and their values, as well as references to other model elements) about the ActorSpecificWork identified by YYY, in the project identified by XXX (Figure 7).
- Subscribing to process events and receiving notifications when they occur. A request to /projects/XXX/tasks/YYY/subscriptions can be used to request a notification when the task YYY ends, for

```
$ http GET                                                                  \
  http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA    \
  X-API-KEY:$CMSPEM_KEY
{
  "item": {
    "deadline": "25/11/2012",
    "name": "Create Wireframe",
    "task": {
      "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA"
    },
    "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
  },
  "url": "http://cmspem.herokuapp.com/projects/1/asworks/_yneHEQfMEeKsbtXB8p_uiA"
}
```

Figure 7. Querying the process model for process information.

example (Figure 8). When the event occurs, the CMSPEM server sends a request to the handler, the URL specified as a parameter in the subscription request.

- Raising process events. Some process events do not occur as direct modifications to the process model but simply consist in taking into account what happened in another development tool. For example, when a developer is on a sick leave, the actor unavailability event can be raised on the process model, by making a request to /projects/XXX/actors/YYY (Figure 9).

Integrating process-support tools in the development environment provides contextual information for some development activities and allows other tools to take actions in reaction to process events, as identified in Section 3.2. The following section presents three example scenarios.

```
$ http -f POST                                                                \
  http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions  \
  handler="http://cmspem-helper.herokuapp.com/task/new"                       \
  event=end                                                                   \
  X-API-KEY:$CMSPEM_KEY

{
  "item": {
    "correlationId": null,
    "created": 1348660352188,
    "event": "end",
    "eventsource": "/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions",
    "eventsourceId": "_EmwHoAc_EeK1qtFkCWkJBA",
    "handler": "http://cmspem-helper.herokuapp.com/task/new",
    "id": "21",
    "updated": 1348660352279,
    "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions/21"
  },
  "url": "http://cmspem.herokuapp.com/projects/1/tasks/_EmwHoAc_EeK1qtFkCWkJBA/subscriptions/21"
}
```

Figure 8. Subscribing to process events.

```
$ http POST                                                                 \
  http://cmspem.herokuapp.com/projects/1/actors/_RR3gMAe_EeKeLOtlvqDWnw     \
  event=unavailable                                                         \
  X-API-KEY:$CMSPEM_KEY

{
  "email": "eric.kedji@gmail.com",
  "fullName": "Komlan KEDJI",
  "name": "Eric",
  "roles": [
    {
      "url": "http://cmspem.herokuapp.com/projects/1/roles/_mXInsJ0tEeG5Y-K8FiquMQ"
    }
  ],
  "url": "http://cmspem.herokuapp.com/projects/1/actors/_RR3gMAe_EeKeLOtlvqDWnw"
}
```

Figure 9. Raising process events.

6.2.1. Contextual information in notifications. Whenever a bug is reported or a build failure happens, notifications are typically sent to some project participants. Upon receiving such notifications, one must gather contextual information about people relevant to the event and their availability, relevant tasks and their status, related work products and their states, and so on. This contextual information is available in the process model and can be easily queried on the CMSPEM process server.

The steps in the execution of such utility for bug reports are as follows:

- Subscribe to the bug creation event on a third-party tool using its API (one-time setup).
- Upon receiving a notification of bug report creation, query the bug tracker for the bug description, parse it, and extract references to tasks and actors.
- Query the CMSPEM server for status information about the aforementioned tasks and actors.
- Use the bug tracker API to add a comment on the bug report, with the contextual information extracted from the process model.

An implementation of such utility, using the Github (a code hosting and collaboration service) API generates the automatic comment depicted in Figure 10. An issue ticket has been opened, which references process model concepts such as participants and actor-specific tasks. Contextual information about two participants and actor-specific tasks (including its deadline) are automatically made available as an issue comment.

6.2.2. Automated reporting. Participants in software projects, especially in industrial settings, usually need to produce weekly progress reports. Such reports contain planned tasks carried out by the individual, exceptional tasks, and remaining work to do on assigned tasks.

Information needed to produce such reports is available in bug trackers, version control tools, and so on. However, it needs to be placed in the context of planned tasks. The concepts introduced in CMSPEM are exactly those needed to realize such mappings. For example, code contributions in a version control system are tied to participants by their email addresses. Actor instances in a

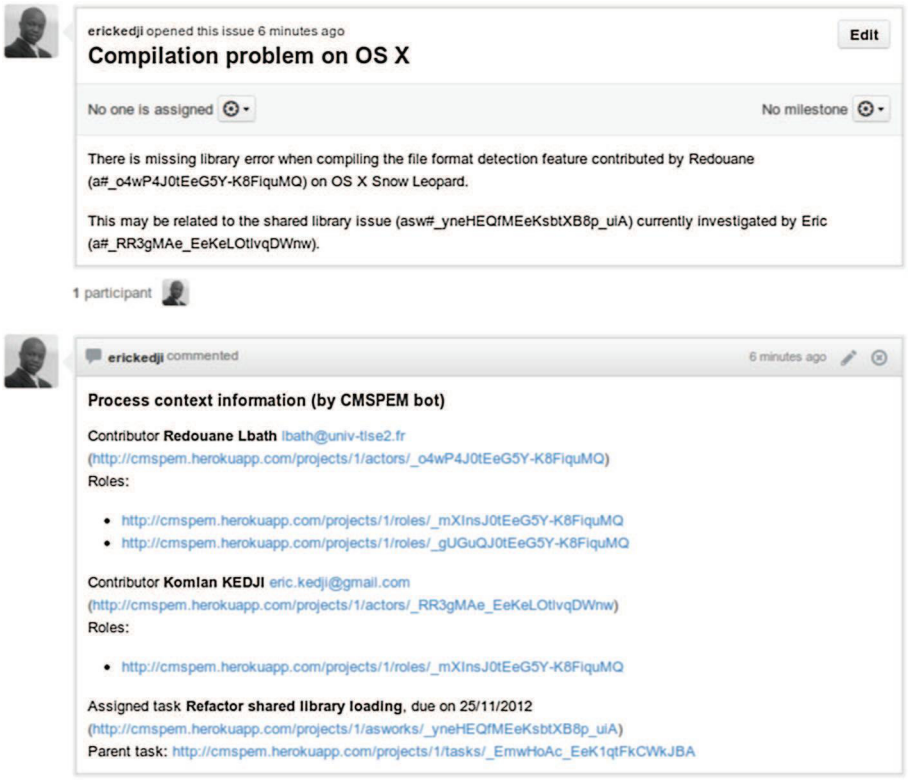

Figure 10. A comment on a Github defect ticket offering automatic context information.

CMSPEM model also have their email addresses specified, which allows to map these code contributions to an Actor.

Examples of mappings are as follows:

- commit author ↦ Actor (using the commit author's email address and the email property of the Actor concept);
- commit ↦ ActorSpecificWork (using a reference to the actor-specific work in the commit message);
- commit ↦ ActorSpecificArtifact (using the names of the files modified in the commit);
- bug report author ↦ Actor (using the bug report author's email address and the email property of the Actor concept); and
- bug report ↦ ActorSpecificWork (using a reference to the actor-specific work in the bug report description).

6.2.3. Cleanup actions. Generally speaking, information found in tools such as VCSs and bug trackers is finer grained than process information. The immediate consequence is that changes on a single process element affect a whole group of entities in third-party tools. This presents an automation opportunity, which can be implemented by listening to process events and automatically carrying out the needed actions.

A typical case is related to cleanup actions. For example, when a team member is temporarily unavailable or replaced, defect tickets assigned to him can be automatically updated accordingly, comments added to them and so on. The CMSPEM server enables the implementation of such utilities by exposing process events such as the availibity or unavailability of an actor and the end of a task.

6.3. Lessons learned

In this validation study, we showed how existing open-source projects use deep linking and hooks to integrate software development tools. We also demonstrated an integration script that integrates the CMSPEM server with a third-party service, Github. The development of such utility highlighted the usefulness of adopting existing tool integration conventions. The use of Web hooks between Github and the CMSPEM server and the fact that the two systems used JSON as a data exchange format minimized the amount of glue code needed. The ability to reference specific model elements such as actor-specific tasks by URL, combined with the automatic handling of links in comments by the Github service, means that participants can click on links in context-information messages and access further details on the CMSPEM server.

Initial applications of CMSPEM to practical cases yielded some insights. For example, it is not clear how the concept of ActorSpecificArtifact must be mapped to file system objects. In a version control system, for example, files are natural information units. However, at the level of process models, the concept of WorkProductUse refers to deliverables, which could correspond to some large group of files such as 'User Interface Code'. This may lead to a too sharp drop in granularity, which makes the link between ActorSpecificArtifacts and WorkProductUses less meaningful. A possible solution is to map ActorSpecificArtifacts to intermediary concepts such as programming language modules or packages, or file system directories.

New concepts introduced in CMSPEM are finer grained than the role, product, and activity concepts from SPEM. This results in a much higher number of model elements, which makes direct manipulation of the model in a graphical representation unpractical. It is therefore necessary to extract visualizations that highlight particular features of the process model. Fortunately, the model manipulation API made available by the CMSPEM server allows the development of third-party utilities that can be used to manipulate part of the model. For example, a visualization can be extracted that shows how 'close' project participants are by analyzing the tasks they collaborate on. A third-party tool can also implement user management and expose a view of the process model that contains only Actor elements.

While analyzing existing development tools, we found some creative uses of semi-structured text for integration. For example, Github, a code hosting service, parses commit messages, extracts defect tickets references, generates automatic comments as needed, or closes the tickets. Generalizing such existing conventions to other concepts such as participants, activities, and so on considerably reduces the work needed to implement integration utilities and improves their usability for software developers.

The implementation of the CMSPEM server is based on a share-nothing, stateless, HTTP framework. Such design avoids complexity induced by server-side state (session invalidation, object life cycle issues, etc.). However, one practical consequence is that each read and write request on the server requires loading the model from disk. Fortunately, loading even big models from disk does not produce any noticeable delay.[‖‖] Model size is also not an issue, as a 1000-element model only uses approximately 170 Ko of disk space. Thanks to the negligible overhead added by the JSON data format, bandwidth usage is also negligible, as server responses are dominated by model element data. One potential scalability issue is related to event handling. Currently, handlers are called sequentially before an update operation is executed on the server. As each handler call corresponds to an HTTP request, this can generate important delays when a lot of handlers are defined. However, the mechanism used in the prototype is not inherent to the server architecture and can be improved if needed. For example, a task queue can be used, so that the server can send the update response right away, and then (asynchronously) call defined handlers.

7. RELATED WORKS

The main research axes related to the work presented in this paper are requirements for collaboration support in software engineering, the design of collaborative development environments, the need to support incomplete process models, which may be enhanced at runtime, and the integration of process-support tools in software development environments.

Whitehead, in [2], laid down a roadmap for collaboration in software engineering. Proposed directions include collaboration infrastructures that support data as well as control integration. Our contribution builds on this insight and is a proposal to enhance the data and control integration abilities of process-based collaboration support tools. Whitehead also made the case for a deeper integration between desktop tools and Web-based tools, which are more and more used in software engineering. This proposal supports such integration using hooks, between a desktop tool, the editor, and a web-based tool, the CMSPEM server.

IBM Jazz is one of the current major efforts that take a holistic approach to tool-based collaboration support and is based on the Eclipse environment [11]. The Jazz project is well integrated with other development tools such as version control, bug tracking, and instant discussion tools. The environment uses the Eclipse editor as the central component, which aggregates information from other collaborative tools. This is understandable as development teams interested in this solution arguably already spend much of their time in Eclipse. However, this solution is not always applicable, as a team may be using another editor. Our approach does not make any assumption about the editor used and does not seek to make any tool the single responsible for the integration of collaboration tools.

Kobialka [20] makes the case for incomplete process models, which may be enhanced at runtime, by directly modifying process instances. The proposal is based on the observation that it is not practical to claim that all activities in software projects have to be completely defined. Incremental process support is therefore needed. While the contribution in [20] is focused on how incomplete process models can be supported by a PSEE using triggers and constraints, our proposal focuses on notifying external tools of changes that are made to a process model, so as to enable control integration.

Barghouti [21] described how Provence, a PSEE, tries to minimize its intrusiveness, by listening to information it needs at the file system level, rather than requiring high-level tool integration. However, this only solves the need to adapt existing tools so that they can be controlled by a PSEE. Intrusiveness can be further lowered by making the PSEE participate like any other tool in collaboration support, buy exposing process data and control integration points, as advocated in our approach. This makes sure that little glue scripts, which connect the functionalities offered by different collaboration tools, can be written without modifying any one of them.

In [5], Ambriola *et al*. made an extensive survey of PSEEs. Main features are listed and different implementations are discussed. This survey touches two points that are relevant to this work: control

[‖‖] Model element access and whole model access requests complete under 0.1 s on a model with 1000 elements, and element listing by kind for a list size of 1000 completes under 1 s.

and data integration. However, all surveyed approaches consider these forms of integration only when the information flows from third-party tools to PSEEs. In other words, surveyed approaches only invoke external tools or read their data. This is a clear indication of the one-way integration between PSEEs and external tools that this contribution argues against. A similar survey, carried out recently [22], studied seven modern PSEEs. However, all three PSEEs that scored well on the criterion of providing extension points for tool integration are restricted to model-driven engineering. As such, their integration capabilities are a natural result of the control they have on the kind of tools used in such style of development (model editors, transformation engines, etc., all using the same metamodel). Therefore, such integration style does not apply to generic software engineering tools.

Reviewing the state of the art in tool integration in 2004 [23], Wicks surveys several topics related to tool integration, among which process-based tool integration. The author notes how the lack of flexibility and adaptability prevents such solutions from being offered commercially in the marketplace. In a later study [24], the authors, while proposing a research agenda for tool integration, observed that defect tracking, change management, and configuration management are usually far better integrated than project management, requirements management, analysis, design, and implementation. The authors suggest that this is due to business decisions (e.g., because a team previously failed to deliver to customers the required software components for a specific software release). This study puts forth an additional hypothesis that links how much a concern (such as process modeling) is integrated to how easily it allows other tools to consume its data and events.

8. CONCLUSION

This work defined CMSPEM, a metamodel that extends SPEM with concepts and relationships needed to capture collaboration, and proposed a new approach for the integration of process-based tools with other collaboration support tools. We showed how supporting data and control integration in process-based tools can enhance their usefulness for practitioners. We extracted two important integration enablers in existing tools, namely, deep linking and hooks. We then proposed a framework for process support, based on the CMSPEM metamodel, that offers deep linking and hooks by design. A case study, based on the industrial practices of AKKA Technologies, a partner of the Galaxy project this work is part of, showed how the proposed framework can be used to enhance collaboration support, by combining the information in process models with the services offered by other collaboration support tools.

A graphical editor based on TOPCASED and a DSL editor based on Xtext are available for the edition of CMSPEM models and can communicate with the CMSPEM engine. The CMSPEM engine can host process models for multiple software projects and exposes a model element manipulation and a model event subscription and notification API over HTTP. Sample integration scripts between the CMSPEM server and existing software development tools have been developed. A validation study has been conducted on 219 open-source projects, which demonstrates the use of deep links and hooks as integration strategies, and we have shown how this strategy can be applied to process-support tools with sample integration utilities. This study showed practical implementations of the integration opportunities that were identified on the AKKA Technologies case study.

We plan to develop additional sample integration scripts (task deadline notification on mailing lists, mapping bug state changes to activity state changes, etc.), and apply the framework to other industrial case studies, especially in the context of model-driven development [25]. This will help us evaluate the proposed architecture scales, with respect to the size of process models and the number of hooks managed by the process engine. Another perspective for this work is the identification of specific collaboration scenarios, which can be described and stored as patterns, so as to assist project managers when creating and updating CMSPEM process models. Finally, a possible extension of this contribution is the exploitation of the CMSPEM server in core activities such as design and requirements engineering.

The authors would like to thank the French ANR Galaxy project, which funded this research work. The Galaxy ANR project is a joint collaboration between French universities (the University of

Toulouse, the University of Nantes, and the Pierre & Marie Curie University of Paris) and industrial partners (AKKA Technologies, Airbus, and Softeam) on the collaborative development of complex systems using the model-driven engineering approach.

REFERENCES

1. Robillard P, Robillard M. Types of collaborative work in software engineering. *Journal of Systems and Software* 2000; **53**(3):219–224.
2. Whitehead J. Collaboration in software engineering: a roadmap. *FOSE 07: 2007 Future of Software Engineering*, IEEE Computer Society: Washington, DC, USA, 2007; 214–225, doi:http://dx.doi.org/10.1109/FOSE.2007.4.
3. Ossher H, Harrison W, Tarr P. Software engineering tools and environments: a roadmap. *Proceedings of the Conference on the Future of Software Engineering*, ACM, 2000; 261–277.
4. Gruhn V. Process-centered software engineering environments, a brief history and future challenges. *Annals of Software Engineering* 2002; **14**(1):363–382.
5. Ambriola V, Conradi R, Fuggetta A. Assessing process-centered software engineering environments. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 1997; **6**(3):283–328.
6. Lanubile F, Ebert C, Prikladnicki R, Vizcaíno A. Collaboration tools for global software engineering. *Software, IEEE* 2010; **27**(2):52–55.
7. Miles-Board T, Carr L, Hall W. Looking for linking: associative links on the web. *Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, ACM, 2002; 76–77.
8. Sadjadi S, McKinley P, Cheng B. Transparent shaping of existing software to support pervasive and autonomic computing, vol. **30**. ACM, 2005.
9. Hupfer S, Cheng L, Ross S, Patterson J. Introducing collaboration into an application development environment. *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, ACM, 2004; 21–24.
10. DeMarco T, Lister T. Programmer performance and the effects of the workplace. *Proceedings of the 8th international conference on Software engineering*, IEEE Computer Society Press, 1985; 268–272.
11. Booch G, Brown A. Collaborative development environments. *Advances in Computers* 2003; **59**:1–27.
12. OMG. Software process engineering metamodel, version 2.0. 2007. http://www.omg.org/spec/SPEM/2.0/.
13. Kabbaj M, Lbath R, Coulette B. A deviation management system for handling software process enactment evolution. *Proceedings of the International Conference on Software Process*, Springer-Verlag, 2008; 186–197.
14. Kedji KA, Coulette B, Lbath R, Nassar M. Modeling ad-hoc collaboration for automated process support. Software Quality Days 2012. Springer: Berlin; 2012.
15. Kedji KA, Coulette B, Nassar M, Lbath R, Thon That MT. Collaborative processes in the real world: embracing their essential nature (regular paper). *International Symposium on Model Driven Engineering: Software & Data Integration, Process Based Approaches and Tools. Colocated with ECMFA 2011 conference, Birmingham*, 2011.
16. Farail P, Gaufillet P, Canals A, Le Camus C, Sciamma D, Michel P, Crégut X, Pantel M. The TOPCASED project: a toolkit in open source for critical aeronautic systems design. *Embedded Real Time Software (ERTS)* 2006; .
17. Fielding RT. Architectural styles and the design of network-based software architectures. PhD Thesis, University of California 2000.
18. MODELIOSOFT. Modelio. http://www.modeliosoft.com/.
19. EclipseProject. Papyrus. http://www.eclipse.org/modeling/mdt/?project=papyrus.
20. Kobialka H. Supporting the software process in a process-centered software engineering environment. *European Journal for the Informatics Professional* 2004; **5**:40–46.
21. Barghouti N. Separating process model enactment from process execution in Provence. *Proceedings of the ninth International Software Process Workshop*, IEEE, 1994; 70–73.
22. Matinnejad R, Ramsin R. An analytical review of process-centered software engineering environments. *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, IEEE, 2012; 64–73.
23. Wicks M. Tool integration in software engineering: the state of the art in 2004. 2004.
24. Wicks M, Dewar R. A new research agenda for tool integration. *Journal of Systems and Software* 2007; **80**(9):1569–1585.
25. Diaw S, Lbath R, Coulette B. Specification and implementation of SPEM4MDE, a metamodel for MDE software processes (regular paper). *International Conference on Software Engineering and Knowledge Engineering (SEKE), Miami - USA, 07/07/11-09/07/11*, Knowledge Systems Institute: http://www.ksi.edu, 2011; 646–653.
26. Kedji K, Lbathd R, Coulette B, Nassar M, Baresse L, Racaru F. Supporting collaborative development using process models: an integration-focused approach. *Software and System Process (ICSSP), 2012 International Conference on*, 2012; 120–129, doi:10.1109/ICSSP.2012.6225954.
27. Atkinson C, Kühne T. The essence of multilevel metamodeling. *Proceedings of the 4th International Conference on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, Springer-Verlag, 2001; 19–33.
28. Henderson-Sellers B. On the challenges of correctly using metamodels in software engineering. *Proceeding of the 2007 conference on New Trends in Software Methodologies, Tools and Techniques: Proceedings of the sixth SoMeT_07*, IOS Press, 2007; 3–35.

AUTHORS' BIOGRAPHIES

Komlan Akpédjé Kedji is a PhD who was, during his research, part of the MACAO team at the IRIT laboratory in Toulouse, France, and affiliated to the IMS team of the SIME laboratory of Rabat, Morocco. His research interests include modeling collaborative work and model-driven engineering. He can be reached at eric.kedji@univ-tlse2.fr or eric.kedji@gmail.com.

Redouane Lbath is teacher and researcher at University of Toulouse 1 Capitole, France. He is a member of MACAO team at IRIT laboratory, Toulouse, France. His first research works focused on artificial intelligence (AI) applied to technical domains (e.g., modeling technical systems and fault diagnosis and testing of embedded complex digital systems). Since 2003, he has been working on AI techniques applied to modeling and enactment of software development processes and model-driven engineering

Bernard Coulette works as a full professor at the University of Toulouse. He belongs to the IRIT laboratory and is currently in charge of the MACAO team. His research field of interest is software engineering, more particularly, integration of viewpoints in object-oriented analysis/design, composition of heterogeneous models, modeling and enactment of collaborative software processes, and model-driven engineering. He has directed many PhD students and is strongly involved in international collaborations (Vietnam and Morocco).

Mahmoud Nassar is Professor and Head of the Software Engineering Department at National Higher School for Computer Science and Systems Analysis (ENSIAS), Rabat, Morocco. He is also a Head of IMS Team of SIME Laboratory. He received his PhD in Computer Science from the INPT Institute of Toulouse. His research interests are integration of viewpoints in object-oriented analysis/design (VUML profile), context-aware service-oriented computing, and model-driven engineering. He can be reached at nassar@ensias.ma.

Laurent Baresse is a Senior Architect and Expert in n-tiers architecture design and specifically in critical and high availability environments; he is a permanent member of the AKKA Technologies 'Cellule Expertise et Conseil'. Expert in UML languages and associated methodologies, Laurent has a perfect knowledge of Java/J2EE platform. He works as consultant for AKKA Technologies clients and is in charge of technical dissemination for internal teams.

Florin Racaru graduated from ENSEEIHT-INP Toulouse in Telecommunications and Networks and received a Master's degree in Networks and Telecommunications in 2004. He was awarded the PhD in Computer Science and Networks from University of Toulouse in 2008. He is or has been involved in research projects related to QoS Networking (EuQoS FP6 project), e-learning and knowledge artifacts (KP-Lab FP6 project), model-driven engineering, collaborative development of complex systems (TOPCASED Aerospace Valley, GALAXY – ANR). Since 2008, he holds an engineer position at AKKA, being involved in architecture design and development of distributed applications and having experience in distributed architectures technologies: Java (J2SE, JEE), XML, UML, RCP, SOAP, REST, and GWT.