Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 6 (4): 79-88, 2012
ISSN 1819-4311 / DOI: 10.3923/jse.2012.79.88

Exploiting Process Events for the Integration of Collaborative Software Development Tools

[1]K.A. Kedji, [1]R. Lbath, [1]B. Coulette and [2]M. Nassar
[1]IRIT, University of Toulouse, Toulouse, France
[2]SIME, Mohamed V University, Rabat, Morocco

Corresponding Author: Komlan Akpedje Kedji, Humanité 1- Apt 219 BP 34776, Cedex 9, 31047 Toulouse, France

ABSTRACT

"Hooks" are an important part of tool integration in software engineering. They allow any development tool to broadcast a development event to some subscribing tools. Most of the existing software development tools have a rich catalog of well-defined events which can be exploited by third parties. This allows any tool to have a complete view of the development environment, without forcing the team to adopt a monolithic, all-encompassing tool. However, process-support tools have been rather weak as contributors to such integration strategy, giving preference to a style of integration where the process-support tool is the central orchestrator of the development environment. This study argues that not only do process support-tools have a rich catalog of events of interest to third party tool but the availability of such events can also significantly improve the overall level of development support. It thus proposed formalism for modeling process events, identified a set of process events of interest for other development tools and described an implementation of the approach in a process server.

Key words: Process-support, tool integration, collaborative development, software engineering environment

INTRODUCTION

Dealing with the complexity of software development requires adequate tool-support. The multitude of development concerns that need to be addressed (configuration management, defect tracking, communication, testing, deployment, etc.) raises the problem of integration, which comprises, among others, data integration and control integration (Wasserman, 1990). Data integration, on the one hand, is concerned with how any tool can, on its own initiative, request some information about the aspect of software development managed by another tool. For example, a dashboard tool can query a test management tool for the results of tests executed on an arbitrary date. Control integration, on the other hand, deals with hot reactions to a happenstance, an event, in a software development tool. For example, a defect ticket status change is an event for which reactions can range from the simple notification email to the automatic execution of some specific test cases (Kiper, 1987; Wasserman, 1990).

This study is focused on control integration. It first demonstrates how events are used as lightweight control integration mechanisms by development tools and applies that understanding to process-support tools. This leads to a conceptualization of events for process models and the identification of process-events of interest to other development tools. It then proceeds to describe the implementation of a process server capable of broadcasting such events, in the context of

GALAXY, a French research project carried out by a consortium of academics and industry partners, with the goal of improving collaboration support for complex projects using the model-driven engineering approach.

There is an extensive literature on tool integration in software engineering environments. The following paragraphs, on the one hand, survey some contributions to the understanding of common integration strategies. On the other hand, existing reviews of process-support tools are shown to confirm their poor support for integration with other software engineering tools.

Kiper (1987) discussed some pioneering ideas about tool integration. Cooperation (control integration), communication (data integration) and commonality (presentation integration) have been distinguished. The study also identified parameters by which the integration capabilities of a tool can be measured, namely, granularity, cohesion and harmony. The current widely used terminology and hierarchy of platform, presentation, data, control and process integration has been introduced by Wasserman (1990) and a layered structure for software tools has been derived from it (shared repository, object management, functionality, user interface, presentation).

Reviewing the state of the art in tool integration Wicks (2004) surveyed 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 (Wicks and Dewar, 2007), 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 (for example, because a team previously failed to deliver to customers the required software components for a specific software release). This study puts forth the alternate hypothesis which links how much an activity is integrated to how easily it allows dispatching events that structure it.

Ambriola *et al.* (1997), Gruhn (2002) and Matinnejad and Ramsin (2012) are investigations of the capabilities of existing PSEEs. The tool integration assessment by Ambriola *et al.* (1997) shows that the surveyed PSEEs (OIKOS, EPOS and SPADE) consider control integration as a matter of controlling other tools (not the other way around) and is mainly about invocation. As such, these tools do not concern themselves with broadcasting process events to other development tools but react to events broadcasted by other tools. Some environments like Provence (Krishnamurthy and Barghouti, 1993) try to be as least intrusive as possible, by listening to external events at the file system level (on files manipulated by other tools), while still not broadcasting any events themselves. Gruhn (2002) argued that deriving support for tool integration is one of the goals of a PSEE, which supports the idea that the PSEE is responsible for the integration of the work environment. Matinnejad and Ramsin (2012) compared seven PSEEs for features classified into intrinsic PSEE requirements (enactment, consistency, flexibility, etc.), criteria derived from proposed PSEE critiques (deviations, human-dimension, new technology adoption) and general requirements. Among the general requirements is the ability of a PSEE to provide extension points for tool integration. However, all three projects that score well on that criterion (SPACE, Transforms and the Model-Driven integrated approach) are specific to the support of 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 edition, transformation engines, etc.). In other words, the relative ease with which such Process-centered software engineering environment (PSEEs) can work with external tools is a direct consequence of the fact that such external tools have been explicitly designed to support an MDE workflow. Therefore, such integration style does not apply to generic software engineering tools.

EVENTS IN SOFTWARE DEVELOPMENT SUPPORT TOOLS

Software Configuration Management (SCM) tools, bug trackers, continuous integration servers, mailing-lists, discussion rooms, etc., can each generate events which correspond to an interesting occurrences.

Events in an SCM tool for example include a commit on a repository, creation of a development branch, merging two branches, sending (push) or receiving (pull) a series of commits to or from a remote repository, etc.

In bug trackers, the lifecycle of bug reports can be naturally described as a series of events, which correspond to transitions. Examples are creating a bug report, verifying a report, assigning a report to a developer, marking a report as a duplicate, commenting on a report, proposing a fix, confirming a fix, closing or reopening a report, etc.

Mailing-lists can also generate events, corresponding to adding or deleting subscribers, creating new discussion threads, etc. For software projects which, like the Linux Kernel Project, use mailing-lists to exchange patches, further analysis of mailing-list messages reveals other events like the availability of a new patch. Likewise, on special mailing-lists set up for continuous integration systems for example, each message is an event corresponding to a build success or failure.

A wide range of automation solutions can be built by wiring the events exposed by the aforementioned development tools. For example, it is common to have software agents automatically analyze commit messages, looking for references to bug tracking tickets identifiers and take the appropriate action on behalf of the user, like closing a report or commenting on it. The same mechanism allows the reception of a series of commits (on an official repository) to trigger automated deployments, announcements in chat rooms, or company dashboard refreshes.

Integration solutions which exploit events (commonly named "hooks") generated by software development tools are decentralized by design, as they do not rely on a central tool. Each tool can broadcast interesting events as it sees fit and other tools can subscribe to those events. This departs from the workspace integration style traditionally implemented by process-centered software engineering environments (PSEEs), where the PSEE acts as a central control hub, listening to events on some tools and reacting by invoking some other tools. This study argues that such common design decision is responsible for the slow adoption of PSEEs and instead proposes that process-support tools broadcast interesting events, just like any other tool (Kedji *et al.*, 2012a). The proposal increases the integration capabilities of process-based tools by enabling other tools to react to process events.

MODELING DEVELOPMENT PROCESS EVENTS

The CMSPEM (Collaborative Model-Based Software and Systems Process Engineering Metamodel) (Kedji *et al.*, 2011) metamodel has been defined as an extension of the SPEM (Software and Systems Process Engineering Metamodel) standard. CMSPEM introduces finer-grained concepts like Actor (a single project participant), ActorSpecificWork (a work item assigned to a single Actor) and ActorSpecificArtifact (a private copy of a work product). These concepts enable, on the one hand, a more precise description of the various relationships which embody collaboration, such as a trainer-trainee relationship between two actors. On the other hand, these extensions result in a more natural mapping between process-related concerns (like a work item) and concepts manipulated in other development tools (like a commit in a version control system of a bug report in an issue tracker). An in-depth discussion of structural concepts (Fig. 1) in CMSPEM is done by Kedji *et al.* (2011, 2012b). The rest of this section focuses on behavior modeling.

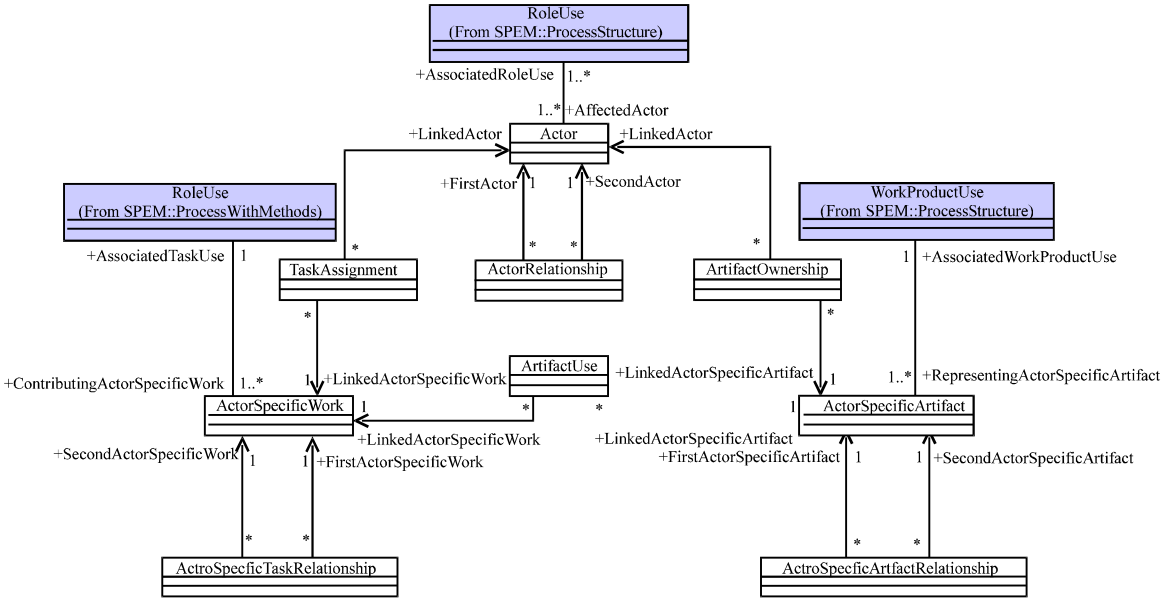

Fig. 1: Structural concepts in the CMSPEM metamodel. The stars and numbers are the standard multiplicities of UML class diagrams

CONCEPTS FOR BEHAVIORAL MODELING

In CMSPEM, the behavior of process models at enactment time is handled using an action and reaction approach: Something happens and subscribers are notified so they can react to it. Model elements which can raise events are called "event sources" and those which can listen to events (and react to them) are called "event listeners". Event listeners are notified only for the events they subscribe to and this is conceptualized as an "event subscription" (Fig. 2). Last, "event bubbling" enables flexible event subscription, by making it possible to consider a set of model elements as a logical group.

Event: Event (from UML::CommonBehaviors::Communications) is an occurrence of something of interest that may trigger a reaction, if an appropriate EventSubscription has been defined. Events have parameters and parameter values describe the event. Events are raised by EventSources and received by EventListeners provided the appropriate EventSubscription exists prior to the occurrence of the event.

EventSource: An EventSource is a model element which can generate events. Generally speaking, an EventSource triggers an event to inform possible listeners of a change in its internal state.

EventListener: An EventListener is a model element which can react to events. A listener can receive events and react to them. To be able to receive an event, a listener must have already subscribed to it prior to its occurrence (through an EventSubscription). Only events of the types subscribed to are sent to an event handler.

EventSubscription: This is a conceptualization of the subscription made by an event listener, to a group of events, on an event source. Events of the specified type, which occur after the

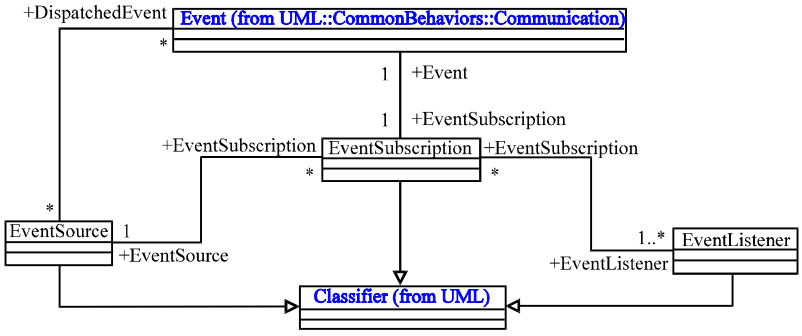

Fig. 2: Event-related metaclasses in CMSPEM. UML concept names are in blue, the stars and numbers are the standard multiplicities of UML class diagrams

subscription, are sent to the event listener, if and only if, they are raised by the specified event source. For each combination of event type and event source, several event subscriptions can be defined.

EVENT HANDLING

The event handling mechanism is the process which starts when an event is generated and ends when the execution of all registered event listeners has ended. For the purpose of explaining this mechanism, the concepts of "containment hierarchy" and "event bubbling" need to be defined.

Containment hierarchy: A simple containment link is defined between model elements in CMSPEM. Each model element has at most one "parent" element. Elements with no parent element are said to be "top-level elements". If an element A is the parent of an element B, B is said to be a "child element" of A. An element A is said to be an "ancestor" of an element B if and only if A is the parent of B, or there exists an element C such that C is the parent of B and A is the ancestor of C.

Event bubbling: An event can be handled, not only by handlers defined on the event source but also by listeners defined on any ancestor of the event source. This is because an event triggered on an element will also be triggered on all its ancestors. The event is said to "bubble" upwards. This mechanism is inspired by event handling as done in the "Document Object Model" in browsers.

Event bubbling is necessary to account for the evolutionary nature of software processes, where the model can be enriched at any time with new model elements. To be able to listen to an event, one must specify the element which generates such event. This makes it impossible to specify that a listener is to receive an event which may be emitted by a model element that is not yet present in the model. Concretely, a listener can ask to be notified any time some attribute is changed on any ActorSpecificWork that is "contained" in some specific TaskUse. Without event bubbling, this can only be done for ActorSpecificWorks that are already present in the model. With event bubbling, the listener can simply listen to the event on the TaskUse, as all events generated by any present of future ActorSpecificWork will reach the TaskUse. When an event is bubbling towards its containers, any container on which relevant listeners have not been defined will simply not react.

The event handling mechanism: The generation and handling of events can be illustrated with the following sample event handling sequence (Fig. 3).

E is an event type and e is a specific event of type E. L1 and L2 are event listeners. S1 and S2 are event sources and S2 is an ancestor of S1. Sub1 and Sub2 are event subscriptions. The steps needed, from event subscription to event handling, in a typical scenario, are as follows:

Step 1: Sub1 is defined as an event subscription on S1, for events of type E, with the listener set to L1. Sub2 is an event subscription on S2, for events of type E, with the listener set to L2. The order in which the subscriptions Sub1 and Sub2 are defined does not matter. The only requirement is that Sub1 and Sub2 are defined before the event e occurs

Step 2: The event e (of type E) occurs and S1 is its original source

Step 3: Defined event subscriptions with the source set to S1 are checked. For each subscription whose event type is set to E, the associated listener is called, with the parameters of the specific event e. In this case, L1 is the only listener which matches. If several listeners were defined, they would be called in the order of definition

Step 4: All the listeners called in the previous step finish execution. In this case, the execution of L1 returns

Step 5: If the listener L1 did stop the propagation of the event e, event handling stops here. If not, the event bubbles upwards

Step 6: The event e is sent to the parent element of S1, for handling. The procedure in step 3 is repeated for all subscriptions defined on the parent element (that is, subscriptions for which the source was set to the parent element). When handling on the parent element returns, the event bubbles upwards once more

Step 7: Eventually, the event reaches the ancestor S2 of S1. Event subscriptions with the source set to S2 are checked. For each subscription with the event type set to E, the associated listener (in this case, L2) is called. Each listener called is provided with the event parameters

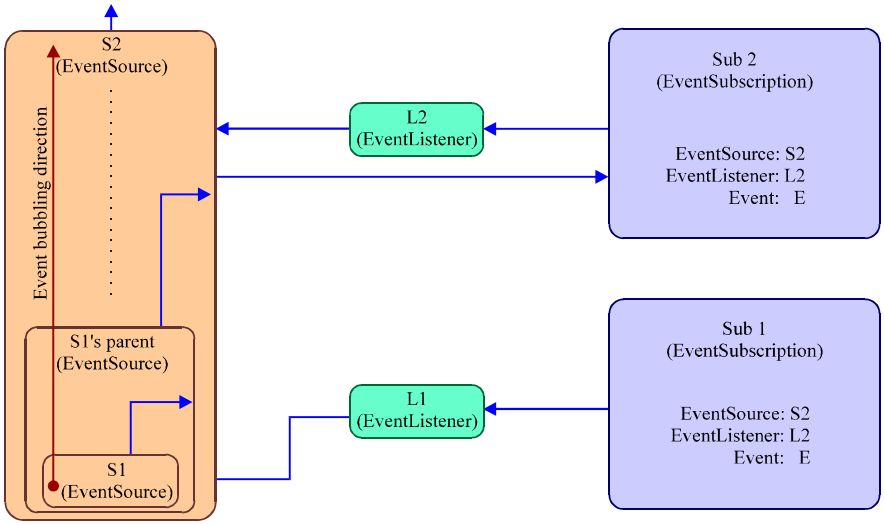

Fig. 3: Sample event handling sequence. Blue arrows show the flow of execution during event handling

Events can have arbitrary parameters, depending on their type. However, for the generic purpose of event handling and event bubbling, two standard parameters are always transmitted to handlers:

- **Source:** This is the element (EventSource) on which the listener was defined. When an event bubbles up, this parameter is continually updated: its value is always set to the element on which the currently processed listeners were defined
- **OriginalSource:** This is the element where the event actually occurred, prior to any bubbling. For each event, this parameter stays the same, even when the event is sent to listeners defined on ancestors, during event bubbling. When an event is initially generated by an event source, OriginalSource and source are identical. They differ only during event bubbling, as the event is handled by listeners defined on ancestors of the original event source

EXPLOITING PROCESS EVENTS FOR SOFTWARE DEVELOPMENT SUPPORT

The formalism defined in the previous section can be exploited to make process support tools participate in the integration of software development tools. To this end, a set of interesting development events are identified, with a description of how such events can be used, in an implementation of a process server, to enhance tool-support in software development activities.

PROCESS EVENTS

A process model which is continually updated to reflect how work is actually being carried out in a software project (people join and leave teams, tasks are started and finished, etc.) has a lot of information of interest to other development tools. We identified the following hierarchy of interesting events that could be raised on a process model and processed by external tools:

- **ActorEvent:** Any event generated by an actor
  - **NewActorEvent:** A new actor has been added to the model
  - **ActorAvailabilityChangeEvent:** A previously available actor becomes unavailable, or a previously unavailable actor becomes available. An actor can, for example, become unavailable because he/she is on a sick leave
- **ActorSpecificWorkEvent:** Any event generated by an actor specific task
  - **NewActorSpecificWorkEvent:** A new actor specific task has been added to the model
  - **ActorSpecificWorkStartEvent:** The execution of an actor specific task has started
  - **ActorSpecificWorkEndEvent:** The execution of an actor specific task ends
- **ActorSpecificArtifactEvent**
  - **NewActorSpecificArtifactEvent:** A new actor specific artifact has been added to the model
  - **ActorSpecificArtifactChangeEvent:** The content of an actor specific artifact has changed. This can simply mean that a file has been modified
  - **ActorSpecificArtifactRemovalEvent:** An actor specific artifact has been removed from the model
- **RelationshipEvent:** Any event generated by a relationship (ActorRelationship, ActorSpecificWorkRelationship, ActorSpecificArtifactRelationship, ArtifactOwnership, TaskAssignment, ArtifactUse)
  - **NewRelationshipEvent:** A new relationship has been added to the model

- **RelationshipValidityChangeEvent:** A relationship is disabled or enabled. Disabling a relationship is a practical way of informing tools to temporarily disregard it
- **RelationshipRemovalEvent:** A relationship has been removed from the model

IMPLEMENTATION AND USE OF PROCESS EVENT HANDLING IN THE CMSPEM SERVER

To make it possible for CMSPEM models to generate the previously identified events, a process engine (based on ECLIPSE/EMF, in Java) which acts as a central server, communicating with process modeling tools and other development tools over HTTP, has been implemented. The CMSPEM server is responsible for updating the authoritative version of a process model, following a request made from a process editor (typically, by the team manager). The server also exposes a REST-style API, which enables third party tools to subscribe to process events. When an event occurs on the process model, the CMSPEM server notifies tools with existing (matching) subscriptions using web hooks (Fig. 4). Further, description of the architecture of the CMSPEM server and its application to a scenario from AKKA Technologies (an European engineering and consulting firm, partner of the Galaxy project) is provided by Kedji *et al.* (2012a).

Using the CMSPEM server, some integration scenarios become possible. For example, the descriptions of a set of defect tickets can list the task (ActorSpecificTask) that is responsible for fixing those particular defects. A subscription can thus be made, by the bug tracker, to the CMSPEM server, for the ActorSpecificTaskEndEvent on the referenced ActorSpecificTask. When the ActorSpecificTask ends (for example, the developer marks it as done), the CMSPEM server will notify the bug tracker, so that the linked defect tickets can be marked as "resolved", with the appropriate context information (who changed the task status, when, etc.). This automation frees developers from some manual bookkeeping work and enhances the contextual information

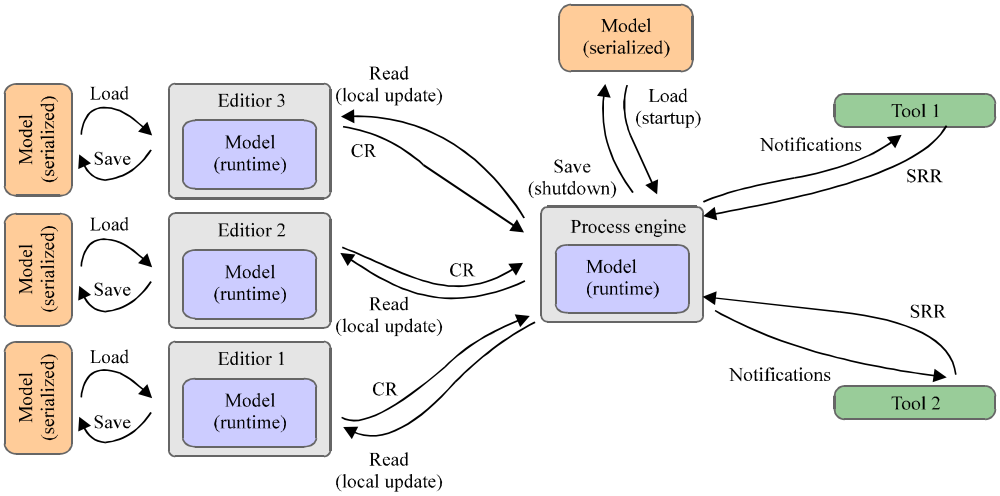

Fig. 4: Abstract architecture of the CMSPM server. A CR (Change request) is a model update transmitted from a model editor (editor1, editor2 and editor3 in the example) to the CMSPEM server, SRR (subscribe, raise event, read model) denotes how third party tools (tool1 and tool2 in the example) use the CMSPEM server API: They can subscribe to events, raise custom events and retrieve any information from the model

available on defect tickets. This improvement is made possible by the fact that the CMSPEM server exposes process events, thus allowing other development tools to react to them.

CONCLUSION

This study made the case for events as a basic mechanism for software engineering tool integration. This study showed how this strategy is prevalent in existing tools and then proposed to apply the same strategy to the integration of process-support tools with other tools. To this end, it extended the SPEM OMG standard with an event-handling formalism, suitable for the description of collaborative processes. It described how this formalism can be used to expose process events to third party tools.

The Galaxy project is the broader context of this contribution and is concerned with supporting collaborative development of complex systems using the MDE approach. The end result of the project is a software development environment which comprises model editors, a communication engine, a project repository (the Galaxy Server), a pluggable process engine, etc. The CMSPEM server described in this study is a process engine, based on events, which can interact with the rest of the Galaxy framework. The engine has been applied to a subset of the standard process of the "Software and Systems" pole of AKKA Technologies is available in a previous study.

Future work, on the one hand, consists in applying the CMSPEM formalism to existing open source projects, so as to demonstrate, on practical cases, how well process-events can be wired with other development events and the benefits of exposing process events to all software development tools. On the other hand, this contribution questions the assumption that process-related preoccupations are special in software engineering and should thus be solved on a higher level of abstraction than the one used for other development preoccupations (like configuration management and defect tracking). This is a first step in a broader study on the relationships between, on the one hand, the perceived abstraction level at which a development concern occurs (platform issues, implementation issues, planning issues, etc.) and, on the other hand, the strategies (data integration only, control integration with fire-and-forget notifications, control integration with hooks that can cancel the current action, etc.) suitable for integrating tools addressing that particular concern with other software development tools.

ACKNOWLEDGMENT

The authors would like to thank the French ANR Galaxy Project which provided funding for this research.

REFERENCES

Ambriola, V., R. Conradi and A. Fuggetta, 1997. Assessing process-centered software engineering environments. ACM Trans. Software Eng. Methodol., 6: 283-328.

Gruhn, V., 2002. Process-centered software engineering environments, a brief history and future challenges. Annal. Software Eng., 14: 363-382.

Kedji, K.A., B. Coulette, M. Nassar, R. Lbath and M.T.T. That, 2011. Collaborative processes in the real world: Embracing their essential nature. Proceedings of the International Symposium on Model Driven Engineering: Software and Data Integration, Process Based Approaches and Tools and Colocated with ECMFA 2011 Conference, June 6-7, 2011, Birmingham, UK.

Kedji, K., R. Lbath, B. Coulette, M. Nassar, L. Baresse and F. Racaru, 2012a. Supporting collaborative development using process models: An integration-focused approach. Proceedings of the International Conference on Software and System Process, June 2-3, 2012, Zurich, Switzerland, pp: 120-129.

Kedji, K.A., B. Coulette, R. Lbath and M. Nassar, 2012b. Modeling ad-hoc collaboration for automated process support. Proceedings of the 4th International Conference on Software Quality, Process Automation in Software Development, January 17-19, 2012, Vienna, Austria, pp: 205-216.

Kiper, J., 1987. The integration of software development tools. Technical Report No. 87-001, Miami University.

Krishnamurthy, B. and N. Barghouti, 1993. Provence: A process visualization and enactment environment. Proceedings of the 4th European Software Engineering Conference Garmisch-Partenkirchen, Software Engineering, September 13-17, 1993, Germany, pp: 451-465.

Matinnejad, R. and R. Ramsin, 2012. An analytical review of process-centered software engineering environments. Proceedings of the 9th Annual IEEE International Conference and Workshops on Engineering of Computer Based Systems, April 11-13, 2012, Novi Sad, Serbia, pp: 64-73.

Wasserman, A., 1990. Tool integration in software engineering environments. Proceedings of the International Workshop on Environments and Software Engineering Environments, September 18-20, 1989, Chinon, France, pp: 137-149.

Wicks, M. and R. Dewar, 2007. A new research agenda for tool integration. J. Syst. Software, 80: 1569-1585.

Wicks, M., 2004. Tool integration in software engineering: The state of the art in 2004. http://www.macs.hw.ac.uk/cs/techreps/docs/files/HW-MACS-TR-0021.pdfs