Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects

Gregor Grambow[1], Roy Oberhauser[1], Manfred Reichert[2]

[1] Computer Science Dept., Aalen University
{gregor.grambow, roy.oberhauser}@htw-aalen.de
[2]Institute for Databases and Information Systems, Ulm University, Germany
manfred.reichert@uni-ulm.de

Abstract. Software Engineering (SE) remains an immature discipline and SE projects continue to be challenging due to their dynamic nature. One problematic aspect is the coordination of and collaboration among the many individuals working in such projects. Numerous efforts to establish software engineering environments (SEEs) to address this aspect have been made. However, since SE projects depend on individuals and their intentions, their collaboration is still performed manually to a large degree. Manual tasks are subject to human error in omission or commission that can result in communication breakdowns which are compounded within multi-project environments. This paper describes a synergistic approach that extends a process-aware information system with contextual awareness and integrates this in a SEE. This enables the system to support the users with active and passive information and support collaboration. Context information is presented to the users, providing them with process navigability information relating to their current activities. Additionally, automated information distribution improves awareness about the actions of others. Finally, this approach enables the automatic initiation and governance of follow-up activities caused by changes implied by other activities.

Keywords: Computer-supported cooperative work; process-centered software engineering environments; process-aware information systems; context-awareness; semantic web applications

1 Introduction

Recently, a trend towards greater automation and process-centricity can be observed in various industries for achieving predictable quality and efficiency [1]. Typically, process automation is applied in domains with foreknown and predictable activity sequences such as production, business, and logistics. In the software development domain, low-level operational workflows involving collaborations typically aberrate sufficiently to make process automation especially challenging.

To enhance the automated coordination capabilities in software engineering environments (SEEs), various challenges must be addressed. Software development is

project-oriented and lacks the typical production stage with repeatable activities or interactions. Process-Centered Software Engineering Environments (PCSEEs) [2] support such projects with both tooling and processes, yet these must be tailored to the unique and diverse project and product needs (e.g., quality levels, team size, etc.). While common software engineering (SE) process models (e.g., VM-XT [3] or Open Unified Process [4]) have proven to be beneficial, they are typically manually implemented (especially in small-to-medium enterprises), often remain coarse in their granularity, are documented to an often general level, and rely on humans to follow and map actual low-level concrete actions and events to the appropriate higher-level process (*process navigability*).

In this paper, the following definition of process and workflow will be used: *Process Management* deals with the explicit identification, implementation, and governance of processes incorporating organizational or business aspects. *Workflow management*, in turn, deals with the automation of business processes or parts thereof. Consequently, a workflow is the technical implementation of a process (or part thereof).

A lack of automatic process guidance and support in an SEE can result in a disparity between the specified and the executed process, and lead to unpredictable process and product quality. Furthermore, uncoordinated activities may occur, affecting process efficiency. From the process perspective, activities and workflows can be roughly separated in two categories: *Intrinsic* activities are planned and executed as part of the SE process model (e.g., VM-XT [3] or Open Unified Process [4]). *Extrinsic* activities, in turn, are executed outside the reference process model and are thus unplanned and difficult to trace or support. For an example of *extrinsic* vs. *intrinsic* workflows, we refer to Fig. 1. The figure shows a source code modification activity (*intrinsic*) that causes necessary modifications on other artifacts. These modification activities are not part of the process (*extrinsic*).

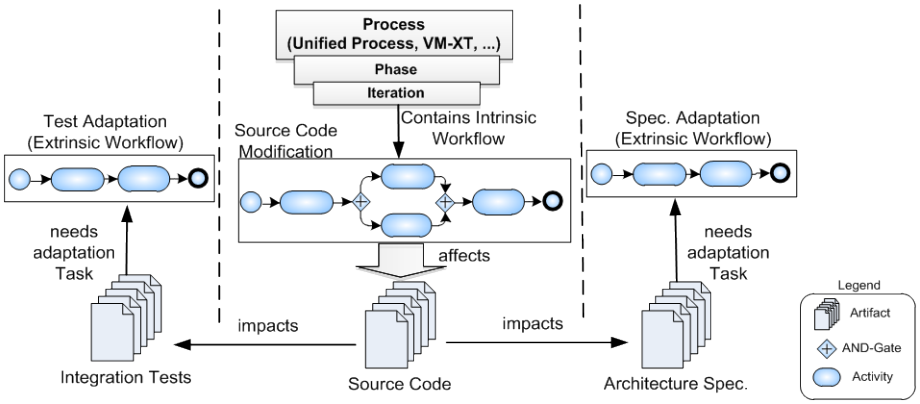

Fig. 1. *Intrinsic* and *Extrinsic* Workflows.

Our previous work has described a holistic framework that applies semantic technologies to SE lifecycles [5] and integrates context-awareness and PAIS (Process-Aware Information System) technology [6] to provide SE process support. [7] dealt

with the explicit modeling and execution support for *extrinsic* activities utilized for the automated treatment of specialized issues in SE projects (e.g., bug fixing or refactoring). [8] investigated consistency in the modeling of processes and workflows in SE to unite abstractly specified processes as well as the concretely and automatically supported workflows. Finally, automatic integration of quality aspects into processes was investigated in [9][10][11].

To comprehensively support the SE process, various other aspects should also be considered: As the SE process largely depends on individuals and their collaboration, the concrete triggering and orchestration of collaboration activities is desirable. To enable configurable collaboration support, various activity dependencies should be supported. For instance, direct follow-up actions may be necessary while in other cases notification to other team members may suffice. Extrinsic follow-up activities should be connected to the appropriate *intrinsic* activities that caused them to support traceability and integration into the SE process. In support of user contextual-awareness, automated guidance should not only be provided for the activities in one workflow (horizontal connections between the activities), but also vertically, making the hierarchical connections between processes and workflows explicit.

This paper presents an approach for collaboration support featuring different capabilities of active and passive information provision to users in an SE project. Furthermore, the connection of *intrinsic* and *extrinsic* activities is addressed, featuring a context-based reasoning process to automatically derive consequences of activities (e.g., impacts on other artifacts) and to govern follow-up activities. Additionally, the connection between abstract processes and concrete workflows is emphasized, providing this information to the user to support *navigability* and process awareness. The following three points sum up the contribution of this paper:

- Individuals working in multi project environments are supported by the automatic provision of extended activity information and process navigability information.
- Automatic information distribution is enabled to inform individuals about various events in a project including the actions of others.
- Automatic initiation and governance of related follow-up activities required by certain actions is provided.

The structure of the paper is as follows: the problems addressed are illustrated in the next section, followed in Section 3 with a description of our solution approach. Section 4 shows the application of our approach to the illustrated problems. Section 5 addresses the issue of the additional effort required. Section 6 then discusses related work, followed by the conclusion.

2 Problem Scenario

The issues being addressed will be illustrated using typical situations in a software company: various projects are executed in parallel by different teams of different sizes. People often have to switch between different projects, and within each project, larger numbers of people are working on the same artifacts. Without additional

coordination effort things can easily be forgotten. Activities mostly imply changes to artifacts, and thus not only relations between *intrinsic* and *extrinsic* activities exist, but there is also a continuously changing artifact base. These facts result in the three problems illustrated below:

Problem A. Project Switching. One issue reported by developers is related to frequent project switching. A person doing this in such a multi-team / multi-project environment has to manually gather context information after a switch to work effectively: Which assignment has to be processed for which project? Which are potential milestones and deadlines? What is the state of the currently processed assignment? What are upcoming activities to be completed?

Problem B. Change Notification. When cooperatively working on the same artifact base, activities and the accompanying changes to artifacts often remain unnoticed by other people. For example, if two teams (e.g. a development team and a test team) are working on the same source code artifacts, they might want to be informed about changes to the artifacts. Such information is often transferred manually and is therefore prone to forgetfulness.

Problem C. Follow-up Action Implications. Also when cooperatively working on the same artifact base, artifact changes often imply certain follow-up actions that are hitherto coordinated manually. This is typically dependent on the artifacts, their relations, and the type of change (e.g., interfaces concern the architect, implementation changes concern the testers, GUI changes concern the user manual author). Fig. 1 depicts a scenario detailing this: It concerns a source code artifact that is part of an interface component: since the file belongs to an interface component, the applied changes possibly not only affect the file's unit tests, but also other artifacts such as the architecture specification or integration tests. These additional activities are usually neither covered by the SE process nor governed by workflows; manual coordination can lead to impacts being overlooked and result in inconsistencies, e.g., between the source code and the tests or specifications. The fact that these activities belong to different project areas with often also different responsible persons makes this even more difficult. Even if not forgotten, follow-up actions could benefit from automated governance and support. Furthermore, it can be difficult to determine which stakeholder should be informed about which change and when, especially considering the dynamic and diverse nature of the artifact-to-stakeholder relationship and various information needs.

3 Automatic coordination support

This section starts with a brief introduction of the framework we continue to develop for supporting the SE process. In particular we want to make clear what capabilities this approach can draw on. For further technical details on its realization, we refer to [11]. The essence of our solution approach is the combination of an adaptive PAIS with semantic technology. A *Process Management* module is used to model both *intrinsic* and *extrinsic* workflows in an integrated way, while additional information about hierarchical dependencies and the context are stored and processed in a semantic-based context management module. To acquire information about the

environment, low-level events occurring during SE tool usage (e.g., saving a file or changing code) are extracted and combined to derive higher-level activities such as creating a unit test.

The realization of the solution approach is the Context-aware Software Engineering Environment Event-driven frameworK (CoSEEEK). It is comprised of modules in a service-based architecture: The *Process Management* module orchestrates SE activities for all project participants. Adaptive PAISs support the coordination of activities according to a pre-specified process model as well as dynamic process changes (e.g., to add, delete, or move activities) in order to cope with unforeseen situations [13][14][15][16]. To enable *Context Management*, semantic technology was chosen due to its many advantages [17], especially a vocabulary including logic statements about the modeled entities and relations as well as a taxonomy for these entities. Furthermore, well-structured ontologies also enhance interoperability between different applications and agents, fostering knowledge sharing and reuse as well as enabling automated consistency checking. The *Context Management* component makes heavy use of semantic technology, utilizing an OWL-DL [18] ontology as well as SWRL [19] for semantic rules processing and SPARQL [20] for semantic querying. Programmatic access to the ontology is supported by the Jena framework [21]. Automatic reasoning capabilities as well as the execution of SWRL rules [22] (while guaranteeing that their execution does not lead to violations of description logic statements) are enabled by Pellet [23].
Event Extraction primarily utilizes sensors for collecting contextual state changes in external elements via events and data associated with various SE tools. Therefore, the sensor framework Hackystat [24] is applied. These low-level atomic events and data are aggregated in the *Event Processing* module, which uses complex event processing (CEP) [25] to create high-level events with contextual semantic value.

The combination of these modules enables CoSEEEK to automatically manage ad-hoc dependencies of certain activities in an either active or passive information distribution fashion to provide coordination support.

3.1 Active coordination support

Active coordination support enables the system to automatically assign follow-up activities to responsible persons or teams. To realize this, the system must be aware of the *intrinsic* activities and workflows that may cause the need for coordination. These workflows, which are based on the users' planned activities (called *Assignments* here, e.g., develop some feature) and which are part of the SE process, are created within CoSEEEK or imported from external process management tools (e.g., MicroTool inStep) in use by an organization. In this paper, OpenUP is used as SE process model. *Assignments* concerning software development are executed by the 'Develop Solution Increment' workflow in that model and imply certain activities like 'Implement Solution' or 'Implement Tests' for the user. The detection of required follow-up activities is realized featuring a three-phased approach:

1. **Determine projects *area*s being affected by an activity**: The first step is configurable and can take into account various facts to determine which *area*s of a project are affected. For the third problem in Section 2 such a

configuration can be '*Search for affected areas in case of technical issues if an activity implies a change to an artifact and the artifact is a source code artifact belonging to an interface component*'.

2. **Determine the concrete target being affected within the *area***: The second step takes the selected *area*s and the target of the applied activity as input. This target can be a concrete artifact as in the given scenario or a more abstract *section* of the project as, e.g., a module. The concrete target is then determined via relations of the different *section*s. An example for this can be implementation and testing: the testing (structural or retesting) of an artifact relates to its implementation. In the given example, the relation does not need to be in place for the concretely processed component, but can be also found if one exists elsewhere in the hierarchy (e.g., the module the concrete artifact belongs to). If there is no direct relation from the processed source code artifact, the system looks for other components the file belongs to (e.g., the module).
3. **Determine the information recipient being responsible for the chosen target**: Once the target of the information distribution or follow-up action is determined, the responsible persons or teams have to be discovered. For example, if the target of the follow-up action is a source code file with no direct responsible party defined, the overlying *section*s are taken into account, e.g., the encapsulating module. If a team is responsible, the information is referred to the designated contact of that team for further distribution.

To enable such automated information distribution, a system must be aware of various facts of the project. Furthermore, to realize automated detection of follow-up actions, different concepts have to be present in the system in order to enable awareness of them:

(1) The project has to be hierarchically split up into components like areas or modules.
(2) Connections of relating components must be established; e.g., the fact that testing a module relies on implementing that module.
(3) Information that can be used to clarify under which circumstances one area affects another must be present.
(4) Different components must be classified; e.g., a package in the source code that realizes the interface of a component.

To support this, the CoSEEEK *Context Management* component contains representations of various project facts. To support awareness and to enrich workflow execution with context information, as shown in Fig. 2, workflows enacted within the *Process Management* module are annotated by concepts in the *Context Management* module. A workflow is mapped by a *Work Unit Container* and an activity is mapped by a *Work Unit*. These are in turn extended by *Assignments* and *Assignment Activities*, which explicitly represent the content of the work the user has to perform for the project. Different areas of a project (like 'Implementation' or 'Testing') are explicitly modeled by the *Area* concept (1), while further separation of the project into logical components is done by the *Project Component* (2). The latter is an abstract building block for structuring a project, which has various subclasses.

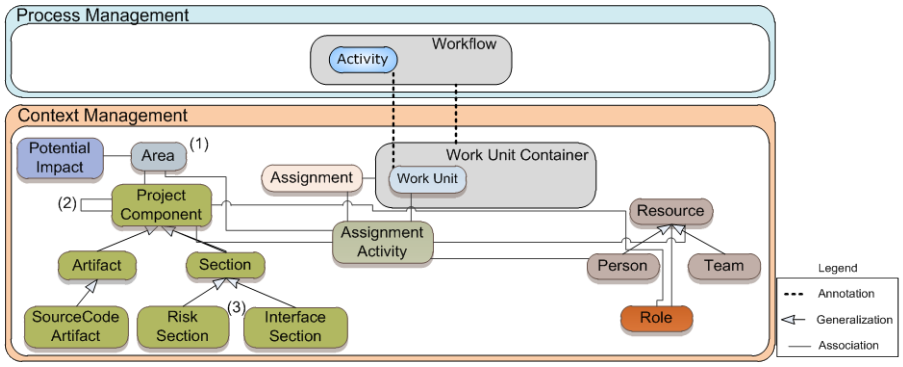

Fig. 2. Concepts enabling active coordination support

Fig. 2 shows two of the subclasses of the *Building Block*: *Artifact*, which is used for various types of processed artifacts (like documents or source code files), and the *Section* that is used for concrete structuring purposes (e.g., used to map a source code package). An *Assignment Activity* being executed by a *Person* processes a certain *Project Component*. A *Project Component*, in turn, has a responsible *Role* taken by a *Resource* that is a *Team* or a *Person*. To enable the configuration of various possible impacts of an activity within the system, different concepts are used: The *Potential Impact* captures potential impacts between *Areas,* like '*When a technical change happens to a component in Area a, this has an impact on Area b*'. *Project Components* of different *Areas* can be related to each other, like '*Testing of Module x relates to the implementation of Module x*'. Many of the concepts also have asserted subclasses for further classifying them. These subclasses of which two are shown in Fig. 2 (3) are dependent on certain conditions. For example, if a *Section* is connected to problems that were detected by the system (e.g., code problems indicated by static analysis tools), the integrated reasoner automatically infers that it belongs to concept *Risk Section*.

3.2 Passive coordination support

Passive coordination support comprises the provisioning of *process navigability information* and *automatic change notifications* for users.

Navigability information support is enabled since the workflows governing the users' activities are mapped by concepts in the *Context Management* module. Thus additional information becomes available to the user that can be useful, e.g., when switching between the activities of different projects. The additionally modeled activity information is illustrated in Fig. 3 and explained below. Additional information comprises the current user *Assignment*: the *Assignment Activity*, *Activity Steps*, the current *Task,* and the *Activity Group* to which the current *Activity* belongs. These concepts can be useful for capturing exactly what the user is doing at the moment as well as for additional support information coming from the process. An example for all additional information presented here is provided in Section 4.

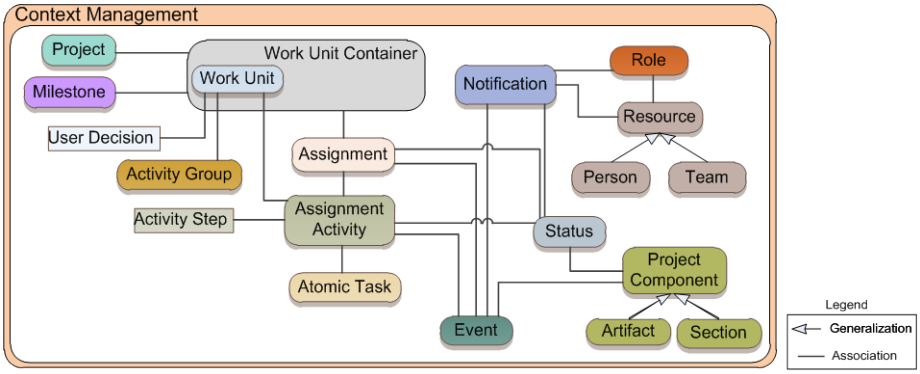

Fig. 3. Concepts enabling passive coordination support

In the *Context Management* module, a concept exists mapping internal variables used for workflow governance to so-called *User Decisions*. That way the user can decide how the workflow is actually executed, incorporating information of the current situation that cannot be known a priori. That way, the user not only has a more semantic and usable influence on the workflow, but also knows what lies ahead. As the more abstract process regions are connected to the operational workflows, the user can also directly receive information about them. This includes e.g., information about the current *Project* or its *Milestones*, which are also modeled in the *Context Management* component.

Automatic change notification is the second passive coordination ability provided by the system. To support users in their collaboration and to counteract forgetfulness, automatic notifications can be beneficial in the first case for two situations in SE projects: When events happen that relate to activities or artifacts and when status changes occur according to the latter. Therefore, several concepts in the *Context Management* component are involved as shown in Fig. 3.

To be able to easily add notification support for the aforementioned example in Section 2, explicit concepts for *Event* and *Status* are utilized. Primarily, *Events* relate to events that occur in the context of a SE project and that are automatically detected by the *Event Management* module. The *Status* concept has been introduced to explicitly model the status of various other concepts such as *Assignments* or *Artifacts*. In an SE project, various artifacts exist with different relations belonging to different areas of the project. Examples include requirements specifications or source code artifacts. To be able to explicitly describe this in the *Context Management* module, the *Project Component* is used as abstract building block for structuring of a project.

Specializations of this concept are the aforementioned *Artifacts* and *Sections*. As example consider a source code structure where the *sections* depict the source code packages. User management in the *Context Management* component includes concepts for roles, persons, and teams. A *Role* can be used as a placeholder for an activity when it is not yet known who should execute the activity. They can also be used in relation to *Project Components* to express, e.g., that a *Person* is responsible for a certain source code package. *Persons* and *Teams* are abstracted to a *Resource*

concept to enable the assignment of activities to *Teams* as well as single *Persons*. Utilizing all of the aforementioned concepts, it is easily possible with the *Notification* concept to configure user notifications relating to various events and status changes in a SE project. Two types of notifications are supported: General notifications that are abstractly pre-defined, e.g., a notification for a role in a process that has not yet started. This notification is distributed to the person executing the role when the process is running. The second type is user-related notifications that can be added by the users themselves, as when a user wants to be kept up to date on the status of a certain *Artifact*.

4 Application Example

For validating our solution, the problems from Section 2 are used. Prior work investigated the practicality of technical aspects such as performance with regard to CoSEEEK realization elements [7][9].

For the problem example (A) of a user switching between different projects, the solution illustrates the usability of additional process navigability information. In one project, she deals with requirements elicitation and executes the 'Identify and Refine Requirements' workflow from the OpenUP process [4]. In the other project, she develops software executing the 'Develop Solution Increment' workflow. Fig. 4 shows diverse supplementary information on the *Activities* as it is specified in the OpenUP process. There are supportive *Activity Steps* (as e.g., "Gather Information"), a so called discipline for the *Activity* (e.g., "Requirements", also provided by the OpenUP process), the current processed task (e.g., "Coding") and the specific *User Assignment* (as 'Develop Feature X'). Additionally, the specific project (e.g., 'Project A') and its milestones according to the OpenUP process (e.g., 'Initial Operational Capability') are also included. In the 'Develop Solution Increment' workflow there are many decisions for potential loops or optional activities. These decisions are dependent on internal workflow variables. In this example the mapping from workflow variables to user decisions is done in a way that the user can directly select the next upcoming activity. As shown in the example, after the 'Implement Solution' activity, there are four possible successors the user can directly choose.

The second problem (B) deals with information requirements relating to different people and teams working on the same artifact base. The solution for this is a pre-configured *Notification* to inform users or teams being responsible for source code packages of changes made to them. As the *Notification* is pre-defined, it does not relate to a concrete *Person* or *Team* but to a *Role* defined for a *Section*. This *Role* is later taken by a *Resource;* when detecting that changes to *Artifacts* contained in that *Section* are made, the *Resource* is automatically notified. However, users can configure personalized *Notifications* as well: Assume that a user is interested in a certain *Assignment* of another user as her work relies on it. Therefore, she registers for a new *Notification* relating to the state of the *Assignment*. When the *Assignment* reaches that state (e.g., 'completed') she is automatically notified.

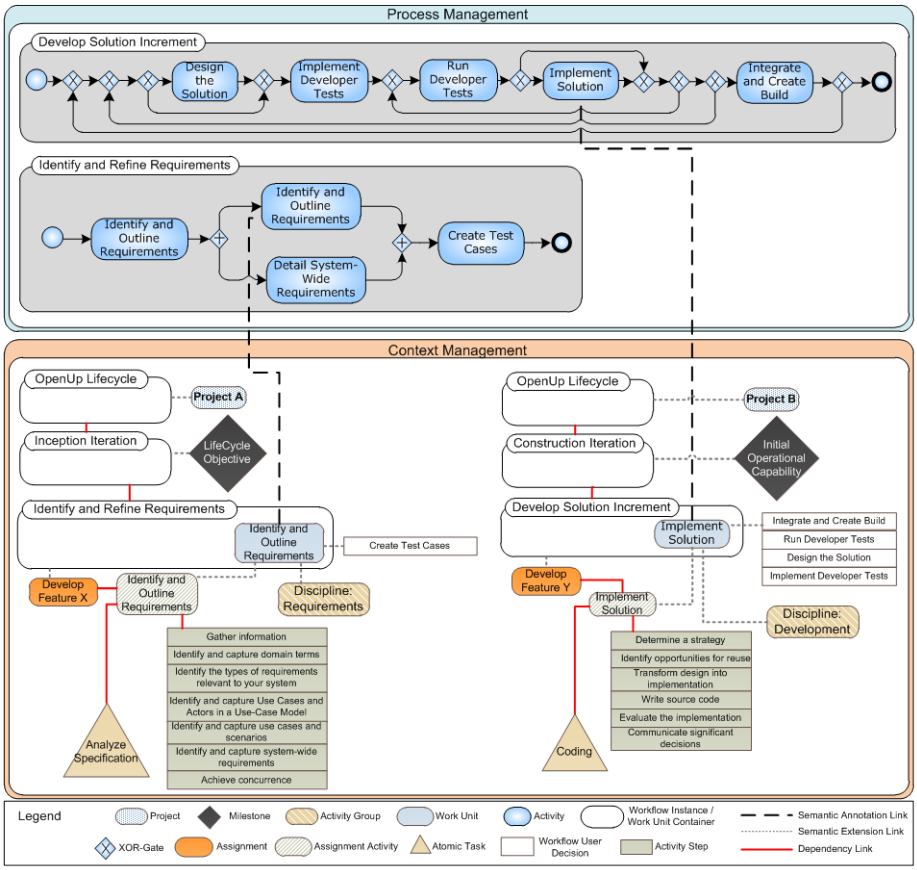

Fig. 4. Navigability information example

The third problem (C) deals with *intrinsic* activities whose outcome requires certain *extrinsic* follow-up activities. As illustrated in Section 2, the modification of a source code artifact that belongs to the interface of a component is the target. Such changes often require adapting integration tests or architecture documents. Dependent adaptations usually do not appear in the workflows belonging to SE processes and are thus *extrinsic* workflows. The given example illustrates the case for the follow-up actions regarding the tests as shown in Fig. 5.

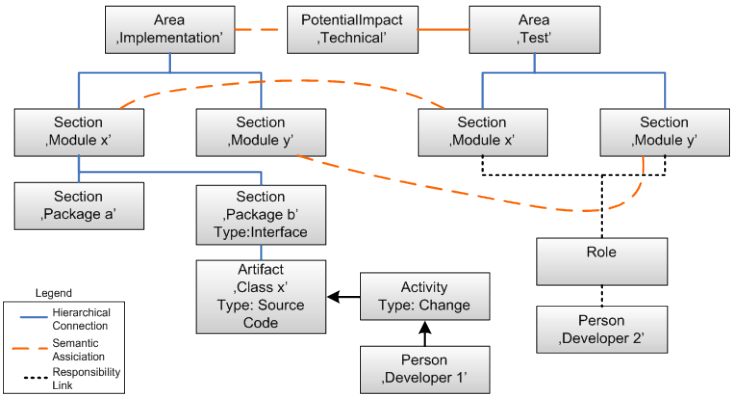

Fig. 5. Active coordination support example.

Fig. 5 shows two defined project *area*s 'Implementation' and 'Test'. There is a *PotentialImpact* configured for relating technical issues from 'Implementation' to 'Test'. For the implementation *area*, there are different modules with different packages. Modules x and y also appear in the test *area* and relate to the counterparts in the implementation *area* as indicated by the curved lines. Developer 2 is responsible for the tests of Modules x and y. Assume now that Developer 1 changes a class belonging to Package b, indicated by the change activity.

The information about the component, the kind of change applied to it, and the user ID of the responsible person are forwarded via an event to the *Process Management* module, which starts a workflow to govern the desired activities for the respective user. This workflow can be based on a predefined workflow template or be custom-built from a problem-oriented declarative definition as described in [7]. When a task of that workflow becomes available to a user, an event is automatically distributed to CoSEEEK's web GUI shown in Fig. 6.

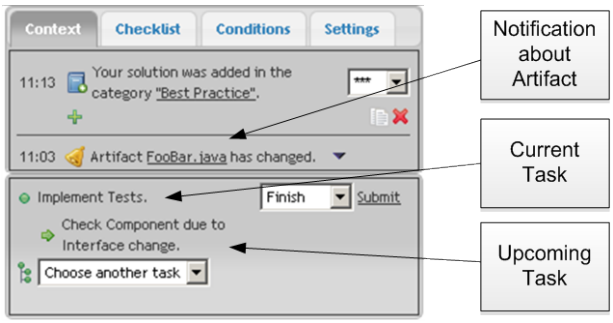

Fig. 6. CoSEEEK Web GUI.

All tasks are shown at the bottom of the GUI. In order to avoid subjecting a user to information overload, only the current task and the next upcoming task proposed by

the system are shown. The user may change the selection of the next upcoming task via a dropdown list. In this example, the current task is "Implement Tests" from an *intrinsic* workflow, while the next upcoming task is "Check Component due to Interface Change" from an *extrinsic* workflow. The upper part of the GUI contains information provided by the framework. Among other things, it can be used to display additional task information and notifications about components for which change notification is configured. This example shows the notification about the change of an artifact.

In summary, the resolution provides collaboration capabilities via coordination of *extrinsic* and *intrinsic* workflows in a PAIS and the availability and use of context information via semantic technology. Activities that are often omitted and not modeled in PCSEEs are explicitly modeled and automatically coordinated via CoSEEEK. Additional support is provided for software engineers working in multi-project environments by making navigability information available and fostering situational awareness. Finally, automatic information provision can keep users updated on artifacts states or other new events in the project.

5 Modeling Effort

Additional modeling effort is imposed by the approach. The processes are modeled not only in the PAIS but also in the ontology. Configuration is required for how various follow-up actions should be treated. To keep the effort reasonable, some default functions and definitions are provided in the framework. The semantic enhancements to process management (*WorkUnitContainers* and *WorkUnits*) are generated automatically from the workflow templates of the *Process Management* module. To gain an awareness of project artifacts, scans are conducted on specified folders. Since the system is aware of SE tools via sensors, it becomes aware of all processed and new artifacts, and the information is acquired on the fly. An initial set of *ProjectComponents* is provided and the structure of certain *Areas* can be imported, e.g., from a folder structure or a source code package structure. Examples include the *Areas* 'Implementation' and 'Test': the system can automatically read the package structure and thus import references to all artifacts into the ontology that are hierarchically organized under various *Sections* that are created from the different packages in the source code. The names of the packages can be automatically matched to those to which they may relate. For instance, relations between 'Test' packages and 'Implementation' packages can be automatically established.

6 Related Work

With regard to PCSEEs, [13] describe SOA-based extensible and self-contained sub-processes that are aligned to each task. A dynamic runtime selection is made depending on the context of the particular work instance. OPEN [26] is a CORBA-based PCSEE that addressed business, quality, model, and reuse issues. DiME [27] provides a proprietary, integrated, collaborative environment for managing product

definition, development, and delivery processes and information. CASDE [28] and CooLDev [29] utilize activity theory for building an environment supporting collaborative work. CASDE features a role-based awareness module managing mutual awareness of different roles. CooLDev is a plug-in for the Eclipse IDE that manages activities performed with other plug-ins in the context of global cooperative activities. CAISE [30] is a collaborative SE framework with the ability to integrate SE tools. CAISE supports the development of new SE tools based on collaboration patterns.

An industry approach for collaborative development is provided by the IBM Jazz / Rational Team Concert products [31]. Jazz offers an infrastructure for distributed development including the technical basis for integration of various clients as well as data and services. It enables comprehensive project, bug, and configuration management as well as event notifications, traceability, and other software development related tasks. Team Concert is a collaborative software development environment built on Jazz technology utilizing its capabilities to provide an integrated solution for software configuration management, work item management, and build management with additional features like customizable dashboards, milestone tracking, or process templates for common processes.

In contrast, CoSEEEK offers a combination of features not found in the aforementioned approaches: workflow guidance is not only offered for activities contained in development processes (*intrinsic*), but also for *extrinsic* activities, which are not explicitly modeled within those processes. The holistic combination of all project *area*s in conjunction with semantic technology also enables the framework to provide intelligent decisions and thus a higher level of automation. The tight integration of PAIS technology with context knowledge not only enables the distribution of information, but also the automated support and governance of activities in adapted workflows.

Modeling SE processes in semantic technologies can enhance reuse and leverage available tooling, as shown by [32]. [33] used an ontology for CMMI-SW assessments, and [34] used ontologies for the Software Engineering Body of Knowledge (SWEBOK). CoSEEEK leverages semantic usage for real-time contextual-awareness in SEEs to improve SE workflows and collaboration and for supporting *navigability* and situational-awareness. The main differentiation criterion to other approaches utilizing ontologies for collaboration is the holistic integration of all project *area*s to foster synergies, and in having collaboration not be the sole focus of the framework (e.g., software quality assurance is adaptively integrated as described in [11]). Other approaches have collaboration via ontologies as their focus [35][36]. [35] presents a workflow-centric collaboration system whereby the main component is an ontology repository with ontologies of different abstraction levels. The process model is based on enhanced Petri nets and thus lacks complementary support for dynamic adaptability. [36] presents an Ontology for Contextual Collaborative Applications (OCCA) that provides a generic semantic model specialized for distributed, heterogeneous, and context-aware environments. In contrast to these approaches, CoSEEEK utilizes querying and reasoning capabilities over an ontology and integrates these with process management to support automated dynamic process governance.

6 Conclusion

The high degree of dynamic collaboration in SE raises challenges for the automated support of process awareness and guidance in SEEs. Currently, SEEs lack contextual information and integration, especially with regard to adaptive collaboration and workflows. The presented CoSEEEK approach extends adaptive PAIS with semantic web technologies and advanced event processing techniques. CoSEEEK explicitly models and manages both *intrinsic* and *extrinsic* activities. These are coordinated, and the automatic initiation and distribution of activities can be individually configured. A dynamic information distribution strategy enables related components to be associated even if no direct relations between the source component and the target component exist. The person being responsible for a component can also be determined if no direct responsibility is defined. The procedure requires neither rigidly predefined information channels nor relies on comprehensive and fine-grained predefined information on relating artifacts or responsible persons. The configuration effort to enable automated coordination is reduced by the ability to automatically import needed information and via the inference and reasoning capabilities.

As the automatic initiation of new follow-on activities is neither necessary nor desired in all cases, the system also provides passive collaboration support abilities. These comprise automatic user notifications on various events in a project. Both general pre-configured notifications and user-configured personalized notifications are possible.

Extrinsic activities that have hitherto typically been excluded from modeling are now guided by workflows. These capabilities enable the integration of general process models with concrete activities even if they are *extrinsic* to a particular SE process. Support for situational awareness and *navigability* becomes vital as collaborations become more complex. Additional process navigability information can be automatically provided by CoSEEEK. Individuals working in multi-project environments can profit from this information since it supports them operationally, e.g., when they are switching contexts by providing all relevant information for the current activity.

The presented scenario demonstrated a situation where improved coordination and situational awareness were supported while providing process guidance and *navigability* for collaborating software engineers, enhancing process quality.

Automated support for coordinated collaborative software engineering, with its human interactions and continuously changing tool and process environment, will remain a challenge. Further research potential lies in the aggregation and utilization of available contextual information to increase process effectiveness and efficiency. Future work will investigate industrial usage in production environments with our project partners. For efficiency, a planned feature will aggregate related tasks and, when a predefined threshold is reached, trigger a workflow instance with the cumulated task information. More complex task treatments can also be designated: e.g., in an agile project, emergent uncompleted tasks can be collected and stored in a backlog to inform team members at the beginning of the next iteration. A GUI that enables the easy definition of rules for the automatic initiation of follow-up activities is planned. It will also support the easy registration for notifications on state changes of activities or artifacts or other events.

Acknowledgement

This work was sponsored by BMBF (Federal Ministry of Education and Research) of the Federal Republic of Germany under Contract No. 17N4809.

References

1. Mutschler, B., Reichert, M., Bumiller, J.: Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors, and implications. IEEE Transactions on Systems, Man, and Cybernetics, 38(3), pp. 280-291, 2008.
2. Gruhn, V.: Process-centered software engineering environments, a brief history and future challenges. Annals of Software Engineering, 14(1), pp. 363-382, 2002.
3. Rausch, A., Bartelt, C., Ternité, T., Kuhrmann, M.: The V-Modell XT Applied–Model-Driven and Document-Centric Development. Proc. 3rd World Congress for Software Quality, VOLUME III, pp. 131-138, 2005.
4. OpenUP, 2011. http://epf.eclipse.org/wikis/openup/
5. Oberhauser, R., Schmidt, R.: Towards a Holistic Integration of Software Lifecycle Processes using the Semantic Web. Proc. 2nd Int. Conf. on Software and Data Technologies, 3, pp. 137-144, 2007.
6. Oberhauser, R.: Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments. Semantic Web. In-Tech, Vienna, Austria, pp. 157-179, 2010.
7. Grambow, G., Oberhauser, R., Reichert, M.: Semantic workflow adaption in support of workflow diversity. Proc. 4th Int'l Conf. on Advances in Semantic Processing, pp. 158-165, 2010.
8. Grambow, G., Oberhauser, R., Reichert, M.: Towards a Workflow Language for Software Engineering. Proc. 10th IASTED Conference on Software Engineering, 2011.
9. Grambow, G., Oberhauser, R.: Towards Automated Context-Aware Selection of Software Quality Measures. Proc. 5th Intl. Conf. on Software Engineering Advances, pp. 347-352, 2010.
10. Grambow, G., Oberhauser, R., Reichert, M.: Contextual Injection of Quality Measures into Software Engineering Processes. Int'l Journal on Advances in Software, 4(1 & 2), pp. 76-99, 2011.
11. Grambow, G., Oberhauser, R., Reichert, M.: Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management. Proc. 2nd Int'l. Conf. on Adaptive and Self-adaptive Systems and Applications, pp. 58-67, 2010.
12. Grambow, G., Oberhauser, R., Reichert, M.: Towards Automatic Process-aware Coordination in Collaborative Software Engineering. Proc. 6th International Conference on Software and Data Technologies, pp. 5-14, 2011.
13. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A service-oriented implementation of dynamic flexibility in workflows. On the Move to Meaningful Internet Systems 2006: CoopIS, DOA, GADA, and ODBASE, LNCS, 4275, pp. 291-308, 2006.
14. Dadam, P., Reichert, M.: The ADEPT project: a decade of research and development for robust and flexible process support. Computer Science-Research and Development, 23(2), pp. 81-97, 2009.
15. Weber, B., Sadiq, S., Reichert, M.: Beyond rigidity–dynamic process lifecycle support. Computer Science-Research and Development, 23(2), pp. 47-65, 2009.

16. Reichert, M., Rinderle-Ma, S., Dadam, P.: Flexibility in process-aware information systems. Transactions on Petri Nets and Other Models of Concurrency II, LNCS, 5460, pp. 115-135, 2009.
17. Gasevic, D., Djuric, D., Devedzic, V.: Model driven architecture and ontology development. Springer-Verlag, 2006.
18. McGuinness, D.L., Van Harmelen, F.: OWL web ontology language overview. W3C recommendation, 2004
19. World Wide Web Consortium: SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 2004.
20. Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF. W3C WD 4, 2006.
21. McBride, B.: Jena: A semantic web toolkit. Internet Computing, IEEE, 6(6), pp. 55-59, 2002.
22. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. Web Semantics: Science, Services and Agents on the World Wide Web, 3(1), pp. 41-60, 2005.
23. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 5(2), pp. 51-53, 2007.
24. Johnson, P.M.: Requirement and design trade-offs in Hackystat: An in-process software engineering measurement and analysis system. Proc. 1st Int. Symp. on Empirical Software Engineering and Measurement, pp. 81-90, 2007.
25. Luckham, D.C.: The power of events: an introduction to complex event processing in distributed enterprise systems. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2001.
26. Henderson-Sellers, B.: Process metamodelling and process construction: examples using the OPEN Process Framework (OPF). Annals of Software Engineering, 14(1), pp. 341-362, 2002.
27. Koenig, S.: Integrated process and knowledge management for product definition, development and delivery. Proc. IEEE International Conference on Software-Science, Technology & Engineering, pp. 133, 2003.
28. Jiang, T., Ying, J., Wu, M.: CASDE: An Environment for Collaborative Software Development. Computer Supported Cooperative Work in Design III, LNCS, 4402, pp. 367-376, 2007.
29. Lewandowski, A., Bourguin, G.: Enhancing support for collaboration in software development environments. Computer Supported Cooperative Work in Design III, LNCS, 4402, pp. 160-169, 2007.
30. Cook, C., Churcher, N., Irwin, W.: Towards synchronous collaborative software engineering. Proc. 11th Asia-Pacific Software Engineering Conference, pp. 230-239, 2004.
31. IBM Jazz http://www.jazz.net.
32. Liao, L., Qu, Y., Leung, H.: A software process ontology and its application. Proc. ISWC2005 Workshop on Semantic Web Enabled Software Engineering, pp. 6–10, 2005.
33. Soydan, G.H., Kokar, M.: An OWL ontology for representing the CMMI-SW model. Proc. 2nd Int'l Workshop on Semantic Web Enabled Software Engineering, pp. 1-14, 2006.
34. Calero, C., Ruiz, F., Piattini, M.: Ontologies for software engineering and software technology. Springer-Verlag New York Inc, 2006.
35. Yao, Z., Liu, S., Han, L., Reddy, Y., Yu, J., Liu, Y., Zhang, C., Zheng, Z.: An ontology based workflow centric collaboration system. Computer Supported Cooperative Work in Design III, LNCS, 4402, pp. 689-698, 2007.
36. Wang, G., Jiang, J., Shi, M.: Modeling contexts in collaborative environment: a new approach. Computer Supported Cooperative Work in Design III, LNCS, 4402, pp. 23-32, 2007.