Event-driven Exception Handling for Software Engineering Processes

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. In software development projects, process execution typically lacks automated guidance and support, and process models remain rather abstract. The environment is sufficiently dynamic that unforeseen situations can occur due to various events that lead to potential aberrations and process governance issues. To alleviate this problem, a dynamic exception handling approach for software engineering processes is presented that incorporates event detection and processing facilities and semantic classification capabilities with a dynamic process-aware information system. A scenario is used to illustrate how this approach supports exception handling with different levels of available contextual knowledge in concordance with software engineering environment relations to the development process and the inherent dynamicity of such relations.

Keywords: Complex event processing; semantic processing; event-driven business processes; process-aware information systems; process-centered software engineering environments

1 Introduction

The development of software is a very dynamic and highly intellectual process that strongly depends on a variety of environmental factors as well as individuals and their effective collaboration. In contrast to industrial production processes that are highly repetitive and more predictable, software engineering processes have hitherto hardly been considered for automation. Existing software engineering (SE) process models like VM-XT [1] or the open Unified Process [2] are rather abstract (of necessity for greater applicability) and thus do not really reach the executing persons at the operational level [3]. In sparsely governed processes without automated data assimilation and process extraction, deviations from the planned process, exceptions, or even errors often remain undetected. Even if detected, an automated and effective exception handling is hard to find.

To increase the level of standardization (i.e., usage, repeatability, conformance, etc.) of process execution, automated support for SE processes is desirable. To enable

this in a holistic way, an automated solution should be capable of some kind of process exception handling so that the occurrence of exceptions does not deteriorate process performance. Further, automated process exception support will only be acceptable if it is not too complex or more cumbersome than manual handling [4]. Automated handling implies automated detection of exceptions that depends on the capabilities of the system managing the processes [5]. However, existing process-aware information systems (PAIS) are still rather limited regarding detection and handling of exceptions [6]. Exceptions can arise for reasons such as constraint violations, deadline expiration, activity failures, or discrepancies between the real world and the modeled process [7]. Especially in the highly dynamic SE process domain, exceptions can arise from various sources, and it can be difficult to distinguish between anticipated and unanticipated exceptions. Even if they are detected, it can be difficult to directly correlate them to a simple exception handler. Due to its high dynamicity, SE has been selected as first application domain, but the generic concept can also be applied to other domains.

Two fictional scenarios from the SE domain illustrate the issues:

- Scenario 1 (Bug fixing): In applying a bug fix to a source code file, the removal of a known defect might unintentionally introduce other problems to that file. E.g., source code complexity might increase if multiple people applied "quick and dirty" fixes. Thus, the understandability and maintainability of that file might drop dramatically and raise the probability of further defects.
- Scenario 2 (Process deviation): In developing new software, the process prescribes the development and execution of a unit test to aid the quality of the produced code. For various reasons, the developer omits these activities and integrates the produced code into the system. This could eventually negatively affect the quality of that system.

These scenarios demonstrate the various challenges an automated process exception handling approach for SE faces: Exceptions can arise relating to various items such as activities, artifacts, or the process itself. Many of these exceptions may be difficult to detect, especially for a PAIS without direct knowledge of the environment. It may also be unclear when exactly to handle the exception and who should be responsible. Generally, the knowledge about the exception can vary greatly, making unified handling difficult and the application of standardized exception handlers unsuitable. Both of the aforementioned scenarios will be used to show the applicability of our approach to SE processes and their exception handling.

The remainder of this paper is organized as follows: Section 2 introduces the novel exception handling approach, followed by Section 3 showing its technical realization. An application scenario is presented in Section 4 and related work is discussed in Section 5. Finally, Section 6 presents the conclusion.

2 Flexible Exception Handling

To respond to the special properties of dynamic SE process execution, this paper proposes an advanced process exception handling approach. It is grounded on two

properties: the ability to automatically gather contextual information utilizing special sensors and complex event processing; and second, an enhanced flexibility in the handling of the exceptions is achieved by the separation of different concerns regarding exception handling. These concerns include the determination of the responsible person or concrete insertion of counter measures into the process.

Our approach can be roughly understood as an extended flexible variant of ECA (Event-Condition-Action) [8]. The three phases are called *Recognition*, *Processing*, and *Action* here, as illustrated in Fig. 2. The steps involved in the phases of this approach rely on the following component definitions:

Event: *Event* is used to capture a multitude of possible events that may occur during an SE project. These include, but are not limited to, events that can be related to various exceptions. Examples include the saving of a source code artifact in an integrated development environment (IDE) or the execution of a static source code analysis tool that provides certain metrics. These metrics can be indicative of an arising problem and thus lead to an exception.

Exception: The notion of *Exception* is utilized to classify a deviation from the planned procedure that was recognized to have a potential negative impact on the process and thus should be dealt with to avoid such an impact. In literature [9], typically there is a distinction between anticipated exceptions, whose occurrence can be easily foreseen, and unanticipated exceptions. For anticipated exceptions, standard exception handlers can be defined. That is usually not possible for the unanticipated ones. Since SE projects typically feature a very dynamic process and it may be difficult so foresee a multitude of possible exceptions, our approach does not discriminate between anticipated and unanticipated exceptions. It also does not use standard exception handlers tied to specific exceptions. Flexibility is improved through the explicit separation of events, exceptions, handling of the exceptions, responsible persons, and the point in the process where a handling is invoked. Thus, occurring events can be classified and it can be separately determined whether exceptions shall be raised, what to do with them, when to do it, and who shall do that. Additionally, the approach manages different levels of knowledge about occurring events. Depending on that level of event knowledge, it can be decided whether a more generic exception shall be raised or rather a specialized one. Fig. 1 exemplifies different hierarchically structured exceptions belonging to three defined exception categories.

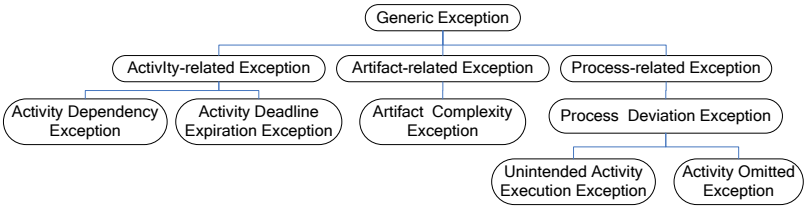

Fig. 1. Exception hierarchy extract

As stated in [10], anticipated exceptions occurring during the execution of pre-specified workflows include the following categories: activity failures, deadline expiration, resource unavailability, discrepancies (between a real-world process and

its computerized counterpart), and constraint violations. These can be covered by the exception types *Activity-related Exception, Artifact-related Exception,* and *Process-related Exception* depicted in Fig. 1. Consider Scenario 1 from the introduction: the code complexity of a source code artifact is very high and was introduced by some activity. The problem may be detected much later and relate more to the artifact than to the activity in that case. Furthermore, the appropriate person to deal with the problem could be the one responsible for the entire artifact rather than the last person who worked on it.

Handling: The notion of *Handling* is used to describe activities executed as countermeasures for a triggered exception. Since SE exceptions are usually complex and of semantic nature, no simple rollback of the activities that caused the exception can be done. As an example, consider the activity of bug fixing (Scenario 1): While fixing a bug, this activity can also introduce additional problems to the code such as increased code complexity. This can happen when the person applying the bug fix is not the one responsible for the processed artifact. As a countermeasure, an explicit refactoring can become necessary. *Handling* neither comprises the person to execute these activities nor the time or point in the process where they are to be executed.

Responsible: *Responsible* captures the responsible person for a *Handling*. As in Scenario 1, this can be the one who executed an activity introducing the exception or the one responsible for an artifact related to an exception.

Target: *Target* is the point in the process where the *Handling* is executed. For certain exceptions, it can be suitable to integrate *Handling* directly into the workflow where the exception occurred whereas in other cases a separate exception handling workflow has to be executed.

The procedure is illustrated in Fig. 2 and described in the following phases and steps.

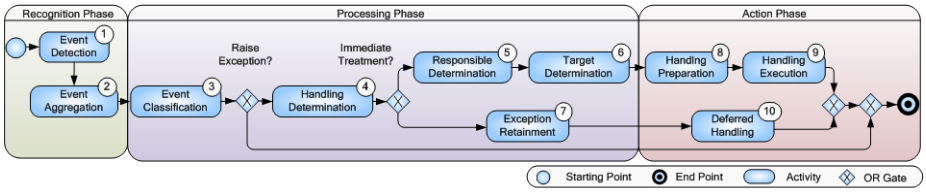

Fig. 2. Abstract Exception Handling Concept

Recognition Phase: In this phase, low and high level events are gathered from the environment in the following steps:

1. **Event Detection**: To enable automated assistance for exception handling, the detection of events related to exceptions must be automated. In a SE project, these events relate to processed activities and artifacts and thus also to supporting tools. Our exception handling approach utilizes a set of sensors that enable gathering of event information from various tools.
2. **Event Aggregation**: Automatically recognized events relating to the tools in an SE project provide information about currently executed activities. Nevertheless, these events are often of rather atomic nature (like saving file) and provide no information about the complex activity a person is processing.

Therefore, these atomic events need to be processed and aggregated to derive higher-level events of more semantic value (like the application of a bug fix).

Processing Phase: In this phase, all necessary parameters for the exception handling are determined using the following steps:

3. **Event Classification**: Event classification can be used to gain more knowledge about the event to be able to find a specific handling later. For example, if a static analysis tool detects deterioration in the quality of a source code artifact, it can be classified as to what kind of source code artifact it relates, e.g., an artifact that constitutes an interface of a component or a test code artifact. In order to effectively automatically the usage of the detected events, they must also be related to the current project. The current focus of the project should be considered, like the defined quality goals that can be important in various situations (the modeling of these for use with automated support has been shown in [11].) For example, if a static analysis tool detects a rise in code complexity of certain source code artifacts, and performance is very important for that project this may be no special event. However, it may be an important event if, for example, the most important quality goals are maintainability or reliability. These factors can be incorporated when deciding whether an exception shall be raised according to an event.
4. **Handling Determination**: When an exception has occurred, it has to be decided when and how to take measures against it. This also depends on the current project situation. The situation can be classified using different parameters like risk or urgency (as shown in [12]). If urgency is high, meaning there is a high schedule pressure on the project, one might decide not to address the exception immediately but to retain it for deferred handling. Since our approach, using event classification, can cope with different levels of knowledge about events, it might also be decided to retain an exception if the knowledge about it does not suffice for immediate automatically supported handling.
5. **Responsible Determination**: If it is decided to take immediate action in case of an exception, the person responsible for that action has to be determined. There can be different possibilities: For example, if an exception relating to an activity occurred, the processor of that activity can be responsible or, if an exception occurred relating to an artifact, the responsible person for that artifact (or, e.g. source code package) can be also responsible for handling the exception. There may not be a direct responsible for each processed artifact, but responsibilities can be hierarchically structured to simplify determination of the responsible party (as described in [13]).
6. **Target Determination**: When the responsible party for handling the exception is determined, the concrete point in the process has to be determined where the handling is applied. As in Scenario 1, if a person introduced an exception while performing an activity and the respective workflow is still running, it can be feasible to directly integrate the handling into that workflow. In other cases, a new workflow for the same or another person can be started.
7. **Exception Retainment**: If, due to various parameters of the situation, no immediate handling is favored, the exception is retained in a special exception

container. That container can be analyzed, e.g., at the end of an iteration by the project manager.

Action Phase: In this phase the concrete execution of the selected exception handling is done via the following steps:

8. **Handling Preparation**: After all parameters for the handling of an exception are determined, the concrete handling has to be prepared, i.e., a new workflow instance has to be created or the handling has to be integrated seamlessly into a running workflow instance.
9. **Handling Execution**: Finally, the prescribed handling is executed by the chosen person.
10. **Deferred Handling**: When exceptions are retained, a human can decide for which exceptions a deferred handling is preferred. Therefore, an additional GUI will be developed presenting a list of retained exceptions and enabling manual determination of a handling or discarding of the exception.

3 Proof-of-Concept Implementation

The realization of the presented concept is based on our previously developed framework CoSEEEK (Context-aware Software Engineering Environment Event-driven Framework) [14]. The framework is intended to provide holistic support for the software development process and this paper presents the newly added exception handling approach on the process level. The framework features a loosely coupled event-driven architecture incorporating various modules. The modules relevant to this new approach will now be described briefly.

Event Detection: This module builds upon the Hackystat framework [15], which provides a rich set of SE tool sensors, to enable the automatic detection of various SE events. Examples of these tools are IDEs or version control systems.

Event Processing: Complex Event Processing (CEP) is applied in this module utilizing the tool esper [16]. Thus, basic events like saving a file can be consolidated into higher-level events like bug fixing.

Context Management: The *Context Management* module incorporates various types of information concerning users, activities and processes, and aggregated events. It manages the connection between the project context and the workflows and is responsible for determination of the exceptions as well as the handlings to be applied. Information is managed via semantic web technology: an OWL-DL ontology [17] serves as an information store, while Pellet [18] is used for logical reasoning. Additionally, Pellet executes rules written in the semantic web rule language (SWRL) [19]. Note that the execution of SWRL rules does not endanger the decidability of the OWL-DL ontology in this case, since Pellet supports DL-safe rules execution [20]. For programmatic access to the ontology, the Jena framework [21] is used.

Process Management: The responsibilities of this module, in view of this scenario, include not only guarantees for correct process execution and reliability, but also adaptability of running workflows to be able to integrate contemporaneous measures for triggered exceptions. Therefore, AristaFlow [22] was chosen since it supports dynamic adaptations of running workflow instances. Further information on

correctness guarantees, adaptation facilities, and other features can be found in [22]. For CoSEEEK to automatically govern workflow execution, and to connect this with contextual facts and apply automated workflow adaptations, the workflows have been contextually annotated in the ontology. This is illustrated in Fig. 3. The concept of the *Work Unit* maps an activity in process management and the *Work Unit Container* maps a workflow in process management.

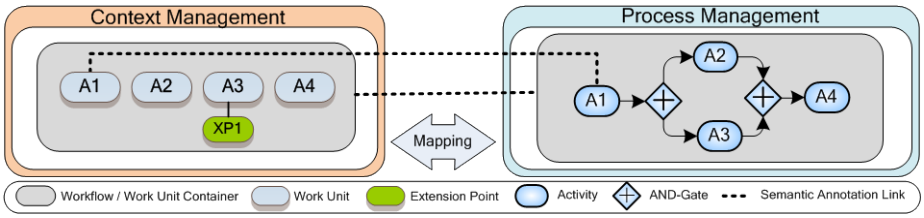

Fig. 3. Contextual process annotations

In the following, the realization of the process illustrated in Fig. 2 shall be briefly described. The process can be initiated by various events detected from tools or triggered by users. These events are aggregated using predefined CEP patterns and then received by the *Context Management* module. Therein, the reasoner further classifies the events as exemplified in the following:

$$SourceCodeEvent \equiv Event$$
$$(\wedge \exists relatedTo\,\mathrm{Pr}\,ojectComponent(SourceCodeArtifact)$$
$$\vee \exists relatedToTool(IDE \vee StaticAnalysis))$$

In the given example, a source code event constitutes an event that is related either to a source code file, an IDE, or a static analysis tool. After classification of the event, it is decided if an exception shall be raised due to the event. This is done by SWRL rules and exemplified in the following:

$$SourceCodeComplexityEvent(EventSCE)$$
$$\wedge hasGoal(currentProject, goalMaintainability)$$
$$\rightarrow raisesException(EventSCE, CodeComplexityException)$$

The example illustrates the raising of a ‘Code Complexity Exception’ if a ‘Source Code Complexity’ event occurs and one of the goals of the current project is maintainability. The creation of the individual exception in the ontology is done programmatically. Thereafter, it is determined with SWRL rules how this exception shall be handled. This decision can incorporate situational properties. In the aforementioned example of the ‘Source Code Complexity Exception’, it can be decided to retain the exception, e.g., if ‘Urgency’ is very high in the current project (or phase or iteration). This will connect the exception to a list associated to the project (or phase or iteration) to be decided upon later by a human. If the situation allows immediate handling, that handling is connected to the exception and the responsible party is determined. This is done with SWRL rules and depends on the type of exception as described in Section 2. The last fact to determine is the concrete target where the handling is to be applied. This is realized by *Extension Points* that are illustrated in Fig. 3. Via *Extension Points*, certain *Work Units* can be defined that enable extending the process. The former have certain properties to distinguish which kinds of extensions are possible (like the application of exception handling - for

another example of their use we refer to [23]). CoSEEEK automatically determines the next upcoming *Extension Point* and initiates automated integration into the running workflow as illustrated in Fig. 4.

The contextual extension of the process management concepts does impose additional configuration effort since workflows would have to be modeled as well as concepts in the ontology. However, this effort can be limited: The direct mappings of the process management concepts can be automatically generated. Future work will include the development of web based GUIs to model the other required concepts (e.g., *Extension Points*) and their connections in the ontology.

4 Application Scenario

This section illustrates the application of the approach by means of Scenario 2. In that scenario, new source code is developed and the respective developer omits prescribed testing activities. Fig. 4A shows an excerpt of a workflow governing these activities ('Implement Solution', 'Implement Developer Test', 'Run Developer Test', 'Integrate and Build') modeled in AristaFlow.

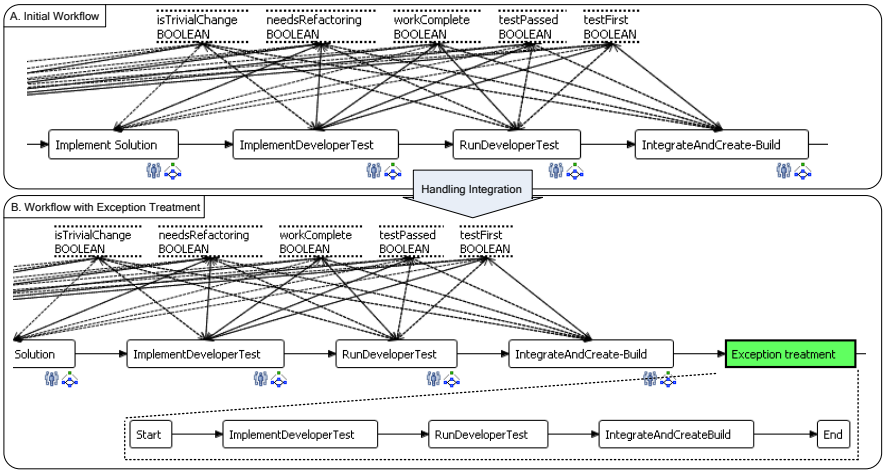

Fig. 4. Exception handling application

After implementing the solution, the developer directly integrates his source code. The steps the system executes to handle that deviation (according to Fig. 2) are explained in the following.

- **Event Detection**: The system detects that the user checks in certain artifacts by sensors in his IDE and the source control system.
- **Event Aggregation**: From the detected events, the system derives the activity 'Integrate and Build' for that user. Since this is not the next intended activity in the workflow, an 'Activity Omitted' event is created.

- **Event Classification**: That event is then contextually classified: the omitted activities relate to testing and thus the event is classified as a 'Testing Activity Omitted' event.
- **Handling Determination**: According to this event, an 'Activity Omitted Exception' is raised that includes information about the omitted activities and the executing person from the event.
- **Responsible Determination**: For this type of exception, the developer who omitted the activities is also responsible for the handling.
- **Target Determination**: In the given case, the workflow of the developer is still running. That means the respective *Work Unit* for the activity 'Integrate and Build' is still active. For that *Work Unit*, an *Extension Point* has been defined that can be used for handling extension integration. Thus, a direct integration into that workflow is chosen.
- **Handling Preparation**: Utilizing the dynamic capabilities of AristaFlow, the handling is integrated into the running workflow instance. This is done by the on-the-fly insertion of a new activity during runtime that is connected to a sub-workflow containing the handling as illustrated in Fig. 4B. Activity data dependencies are not shown for clarity and space reasons.

Technical aspects regarding performance and scalability for different components of the CoSEEEK framework have been previously evaluated in prior work [11][12][13].

5 Related Work

For automatically detecting exceptional situations and determining the actions (i.e., workflow adaptations) required to handle them, ECA-based (Event-Condition-Action) models have often been considered. Classically, many of these approaches limit adaptations to currently enabled and running activities (e.g., to abort, redo, or skip activity execution) [24]. One approach to enable automated adaptations of the unexecuted regions of a running workflow (e.g., to add or delete activities) is AgentWork [25]. It allows process adaptations to be specified at an abstract level and independent from a particular process model based on a temporal ECA rule model. Temporal estimates are made when an ECA rule fires during run-time to determine which parts of a running process instance are affected by the identified exception. For these parts, two types of changes are possible: predictive and reactive change. Predictive changes are applied immediately whereas reactive changes are applied at the time the concerned process fragments are entered. Another modern approach to workflow adaptation is presented in [26]. It consists of a rule-based and data-driven approach to workflow adaptation. Therein, hierarchical context rules are utilized to tailor workflows to changing data-contexts. Additionally, for environments involving eventing paradigms, an event-driven adaptation pattern catalogue is also presented. An example for this is the context-dependent cancelation of a workflow segment and the triggering of a special handler task. These approaches are both event- and rule-based as is CoSEEEK. However, they cannot utilize the variety of contextual events since they lack the environmental sensors integrated via Hackystat. Furthermore, these approaches are rather rigid in the way exceptions are handled since events,

conditions, and relating actions are statically connected. CoSEEEK not only separates exception treatment into additional refinement steps, including semantic classification, but also allows for flexible assignment of handlings based on various factors. That way, an appropriate handling can be found for various situations and different levels of knowledge about a situation. CoSEEEK also enables greater flexibility for the handling itself by adaptively combining what is to be done, who shall do it, and where / when it is to be applied.

Classical rule-based approaches concerning SE processes include MARVEL [27], OIKOS [28], or Merlin [29]. In MARVEL, rules are defined in its own language to enable forward and backward chaining. Thus, the system can request additional activities from a user executing an activity to satisfy the preconditions of the desired action. OIKOS features rules defined in Prolog that are utilized by agents. These cooperating agents operate in different workspaces and enable user cooperation. Merlin also processes different contexts that are assigned to roles. Between these contexts, artifacts are distributed to foster collaboration. As opposed to these approaches, CoSEEEK features the combination of an extended flexible rule-based approach with an advanced adaptive PAIS, semantic classification abilities, and sensors providing contextual information. Therefore, process execution is more robust and the discrepancies between the real world and the modeled process are minimized.

Exception handling could be accomplished utilizing only the PAIS. For example, most BPEL workflow engines support so-called fault handlers to enable some kind of exception handling, for instance [30]. However, these engines do not typically possess process adaptation abilities. While AristaFlow supports this capability and enables exception handling [31], yet in contrast to CoSEEEK the automatic exception handling abilities of these systems are rather limited because they lack both access to context information and semantic reasoning or classification capabilities.

6 Conclusion

SE is a very dynamic and yet immature domain and thus poses a significant challenge for process management. Process models are often abstract and document-centric and not directly utilized in process execution. Moreover, processes are dependent on a variety of environmental and contextual factors. Appropriate process automation could enhance quality and repeatability in SE to better connect the abstract processes with the operational level. However, such a process automation system must be able to accommodate these various aspects and be able to deal with a variety of unforeseen situations regarding process execution in order to provide real support and be relevant. This paper presents an extension to the CoSEEEK framework enabling a flexible exception handling approach incorporating diverse features to support the dynamic SE process:

- Exception occurrence detection is supported by a set of sensors gathering environment knowledge and by CEP that combines those events to derive higher-level events with more semantic value.
- Semantic web technology is integrated to enable classification of events based on various factors like the current situation or the goals of a project. The

proposed approach can deal with different levels of knowledge concerning events and exceptions and thus does not require the separation between anticipated and unanticipated exceptions.
- The combination of environmental awareness with the semantic capabilities also enables the discovery of links between activities and exceptions that have no direct connection.
- The flexibility of the handling is enhanced by separating the determination of the handling, the responsible party, and the target of the handling.
- Featuring the dynamic adaptation capabilities of AristaFlow, exception handling is automatically and seamlessly integrated into users' running workflows.
- If, due to various reasons, a contemporaneous handling is not favorable, deferred handling and analysis of exceptions are also enabled.

Future work will include the industrial application to evaluate the suitability of the approach for real life projects and to refine and extend the modeling in alignment with industrial requirements. It is also planned to extend the deferred handling with exception grouping and exception filters to cope with very high exception load situations or repetitive exceptions. Finally, the application in other domains is also considered, as the approach itself is generic. Therefore, facilities to gather contextual information in these environments have to be developed or integrated.

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. Rausch, A., Bartelt, C., Ternité, T., Kuhrmann, M.: The V-Modell XT Applied - Model-Driven and Document-Centric Development. In 3rd World Congress for Software Quality, VOLUME III, Online Supplement, pp. 131—138, 2005.
2. OpenUP, http://epf.eclipse.org/wikis/openup/ [May 2011]
3. Wallmüller, E.: SPI-Software Process Improvement mit Cmmi und ISO 15504. Hanser Verlag, 2007.
4. Ellis, C., Keddara, K., Rozenberg, G.: Dynamic change within workflow systems. In: Proc. ACM Conf. on Organizational Computing Systems (COOCS'95), pp. 10–21 (1995)
5. Luo, Z., Sheth, A., Kochut, K., Miller, J.: Exception handling in workflow systems. Applied Intelligence 13(2), 125–147, 2000.
6. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Exception Handling Patterns in Process-Aware Information Systems. In: Proc. CAiSE'06, pp. 288–302, 2006.
7. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow resource patterns. Tech. Rep. WP 127, Eindhoven Univ. of Technology, 2004.
8. Paton, N. (Ed.): Active Rules in Database Systems, Springer, Berlin, 1999.
9. Reichert, M., Weber, B.: Enabling Flexibility in Process-aware Information Systems – Challenges, Methods, Technologies, Springer (to appear)
10. Russell, N., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Workflow resource patterns. Tech. Rep. WP 127, Eindhoven Univ. of Technology, 2004.

11. Grambow, G., Oberhauser, R.: Towards Automated Context-Aware Selection of Software Quality Measures. In: Proc. 5th Intl. Conf. on SW Eng. Adv., IEEE CS, 2010.
12. Grambow, G., Oberhauser, R., Reichert, M.: Semantic Workflow Adaption in Support of Workflow Diversity. In: Proc. 4th Int'l Conf. on Advances in Semantic Processing, 2010.
13. Grambow, G., Oberhauser, R., Reichert, M.: Towards Automatic Process-Aware Coordination in Collaborative Software Engineering. In: Proc. of the 6th Int' Conf. on Software and Data Technologies (ICSOFT 2011), NSTICC Press 2011 (to appear)
14. Oberhauser, R.: Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments. In: G. Wu (ed.) Semantic Web, In-Tech, pp. 157-179, 2010.
15. Johnson, P.M.: Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System. In: Proc. of 1st Int. Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society Press, 2007.
16. Espertech Event Stream Intelligence. http://www.espertech.com/products/esper.php [retrieved April 2011]
17. World Wide Web Consortium, OWL Web Ontology Language Semantics and Abstract Syntax, 2004. [retrieved April 2011]
18. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL Reasoner. Journal of Web Semantics, 2006.
19. World Wide Web Consortium: SWRL: A Semantic Web Rule Language Combining OWL and RuleML W3C Member Submission, 2004. [retrieved April 2011]
20. B. Motik, U. Sattler, R. Studer.: Query Answering for OWL-DL with Rules. In: Proc. of the 3rd International Semantic Web Conference (ISWC 2004), pp. 549-563, 2004.
21. McBride, B.: Jena: a semantic web toolkit. Internet Computing, 2002.
22. Dadam, P., Reichert, M.: The ADEPT Project: A Decade of Research and Development for Robust and Flexible Process Support - Challenges and Achievements. Springer, Computer Science - Research and Development, 23(2), pp. 81-97, 2009.
23. Grambow, G., Oberhauser, R., Reichert, M.: Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management. In: Proc. 2nd Int'l. Conf. on Adaptive and Self-adaptive Systems and Applications, 2010.
24. Casati, F., Ceri, S., Paraboschi, S., Pozzi, G.: Specification and implementation of exceptions in workflow management systems. ACM TODS 24(3), 405–451, 1999.
25. Müller, R., Greiner, U., Rahm, E.: AGENTWORK: A workflow system supporting rule–based workflow adaptation. Data & Knowledge Engineering 51(2), 223–256, 2004.
26. Döhring, M., Zimmermann, B., Godehardt, E.: Extended Workflow Flexibility using Rule-Based Adaptation Patterns with Eventing Semantics. LNI P-175, 2010.
27. Barghouti, N.S.: Supporting cooperation in the marvel process-centered sde, in: H. Weber (Ed.), Fifth ACM SIGSOFT Symposium on Software Development Environments, Vol. 17 of Special issue of Software Engineering Notes, Tyson's Corner VA, pages 21–31, 1992.
28. Montangero, C., Ambriola, V.: OIKOS: constructing process-centred SDEs. In: Software Process Modelling and Technology, A. Finkelstein, J. Kramer, and B. Nuseibeh, Eds. Research Studies Press Advanced Software Development Series. Research Studies Press Ltd., Taunton, UK, 131-151, 1994.
29. Junkerman, G., Peuschel, B., Schäfer, W. and Wolf, S.: Merlin: Supporting cooperation in software development through a knowledge-based environment. In: Software Process Modelling and Technology, Research Studies Press Ldt., Ch. 5, 103–130, 1994.
30. Kloppmann, M., Konig, D., Leymann, F., Pfau, G., Roller, D.: Business process choreography in websphere: Combining the power of BPEL and J2EE. IBM Systems Journal 43, 270-296, 2004.
31. Lanz, A. and Reichert, M. Dadam, P.: Making Business Process Implementations Flexible and Robust: Error Handling in the AristaFlow BPM Suite. In: CAiSE'10 Demos, Hammamet, Tunisia, 2010.