Semantically-Driven Workflow Generation using Declarative Modeling for Processes in Software Engineering

Gregor Grambow and Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
{gregor.grambow, roy.oberhauser}@htw-aalen.de

Manfred Reichert
Institute for Databases and Information Systems
Ulm University
Ulm, Germany
manfred.reichert@uni-ulm.d

Abstract—**Software engineering processes are a challenging domain for the application of workflow engines due to their high dynamicity, often evolutionary nature, abstract process models, and the informational and environmental dependencies of their activities. In order to offer automated and relevant process guidance to software developers, the operational-level guidance must be capable of situational adaptation as processes evolve. A declarative workflow modeling approach driven by semantic technology is described that contextually constructs workflows on-the-fly from candidate activities. Thus, automated process guidance in dynamic environments is facilitated while retaining correctness properties, simplifying modeling, and fostering reuse.**

Keywords-declarative workflow modeling; semantic technology; workflow management; process modeling; situational method engineering; evolutionary process support; software engineering workflows

I. INTRODUCTION

Software development continues to face delivery challenges within tight project constraints [1]. Process-centered software engineering [2] is one fundamental way to improve delivery in certain organizations. Establishing standardized process models across an organization can result in cost, schedule, quality, and productivity improvements [3]. Furthermore, Business Process Management (BPM) techniques that govern activity sequences for supporting and fostering process repeatability, reusability, and predictability have been shown to be beneficial in various industries [4][5]. Yet due to various factors, such as the dynamicity of tasks, the project and environment uniqueness, or the lack of workflow reusability, automated process governance in the software engineering (SE) domain for process models is not prevalent. Even when tailored, SE process models (such as OpenUP [6], VM-XT [7]) must necessarily remain relatively abstract to retain general applicability, and thus they provide minimal direct support for the actually executed operational tasks. Additionally, certain work done within a software development project is done extrinsic to the process. Thus, establishing automated process assistance and governance in this domain faces two problems: On the one hand, workflows executed within the development process (*intrinsic workflows*) must be rigidly predefined to be automatable, which is contrary to their dynamic real execution. On the other hand, work that is done outside the development process should also be covered by workflows (*extrinsic workflows*) to enable comprehensive support. Yet, their real execution is often even more dynamic.

The first problem of making *intrinsic workflows* more dynamic and automatable is covered by our prior work: While the general issue of adapting running workflow instances is addressed by dynamic process management [8], it relies heavily on manual customization by users, which is difficult for SE since the required information is typically unavailable to the user ahead of time and such customization is viewed as overhead. Our prior work [9][10] addressed this issue for the automated incorporation of quality assurance activities into the development process. Semantic technology was utilized to automatically adapt workflows that are specified as part of the SE process models (called *intrinsic workflows* hereafter).

The second problem, which is covered by this paper, concerns the activities and workflows executed in a project that are not covered by the SE process models and are often highly dynamic. These *extrinsic workflows* cover issues that frequently recur in SE projects, like bug fixing or refactoring (called "issue workflows" in the following), and are often neither explicitly governed nor supported (perhaps mentioned in best practices). They are typically not as foreseeable as the *intrinsic* ones and may overlap with other activities, often relying on different project parameters (time constraints, risks, etc.). This makes traditional workflow modeling for the issues difficult since many different activity sequences matching different situations would have to be integrated in one vast workflow model. Therefore, an approach to model *extrinsic workflows* differently was proposed in [11]: activity sequences are not modeled as predefined workflows covering all possible situations in one vast model, but rather as set of candidate activities for a certain issue such as bug fixing. On this basis, situational method engineering (SME) [12] is utilized - a paradigm that predicates that a method to solve a problem should be decomposed into fragments that should be combined based on the properties of the current situation. Thus, a set of possible activities for an issue can be specified and workflows for different occurrences of that issue with different subsets of activities can be automatically

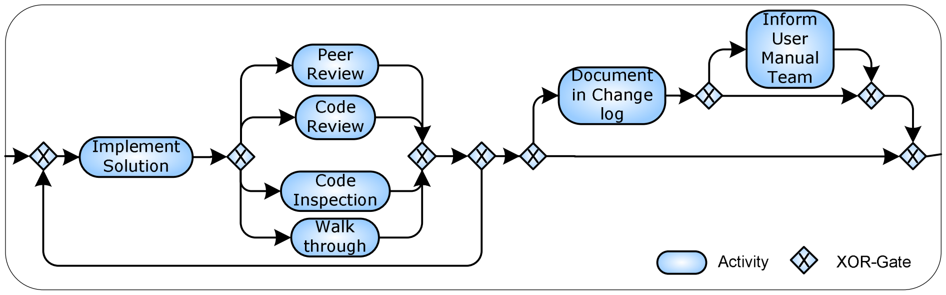

Figure 1. A workflow snippet for a bug fixing issue.

constructed using SME. As the properties of the situation (such as the level of risk) change, the workflow should be able to evolve with the situation during execution. Adaptation of the workflows after creation and instantiation is enabled to match the current situation utilizing SME. Our initial approach [11] focused on the connection of activities with SME properties and the selection of activity subsets matching various situations. This resulted in several limitations concerning the modeling and the enactment of the generated workflows. Only workflows with strict sequentially executed activities were possible and all selectable activities required pre-specified bi-lateral connections to enable correct sequencing, meaning that each activity belonging to an issue had to have a connection to all other activities of the issue. This made modeling a cumbersome task.

To remove these prior limitations, the approach described in this paper leverages semantic technology to enable more complex automated and contextual workflow construction. For reasonable workflow generation, the following requirements must be satisfied: (1) complex workflows, when needed, should be possible and support parallel or repeated execution of activities, (2) modeling of the issues (described below) should be supported, and (3) reuse of workflow fragments should be supported.

SE projects can contain various types of issues like bug fixing, refactoring, technology exchange, or infrastructural issues. As example, consider a predefined workflow for the issue of bug fixing [11]. From that workflow, a snippet is extracted and shown in Fig. 1 to illustrate modeling for such an issue in SE. The workflow snippet shows some activities that can be executed after the implementation of the bug fix itself: at first, one of four different review activities is chosen to verify the bug fix. If rework is necessary, verification must be repeated. When the bug fix is approved, different documentation activities are possible whose execution depends on the user impact of the bug fix. The workflow shows that for most cases more than one alternative is possible. Some workflow parts may also be applicable to other issues beyond bug fixing, such as refactoring. This example is also modeled using our approach in Section IV to enable a comparison with standard workflow modeling.

The remainder of this paper is organized as follows: the next section presents the solution approach and Section III discusses its realization. A scenario improvement is then illustrated in Section IV. Section V presents initial evaluation results. Section VI discusses related work and is followed by the conclusion.

II. Solution Approach

Our solution constitutes part of the CoSEEEK (Context-aware Software Engineering Environment Event-driven) frameworK [13], developed to aid SE projects by providing automated guidance for all project participants. AristaFlow [14], a highly flexible process management system, was used to support adaptations of running workflow instances. To enable the system to automatically apply situational adaptations to dynamically constructed workflows, contextual knowledge is required that is managed and utilized by semantic technology. CoSEEEK is integrated into the SE environment, providing information sharing facilities and supporting inference of new information from known facts. An OWL-DL [15] ontology is applied in combination with the semantic web rule language (SWRL) [16], using Pellet [17] as a reasoner and the Jena framework [18] for programmatic access to the concepts. The combination of OWL-DL and SWRL was chosen due to the practical availability of reasoners supporting both. Since there is the possibility that SWRL rule execution can violate the logic consistency of the OWL-DL ontology, Pellet was chosen, as it supports 'DL-safe' rule execution to avoid that issue.

The following subsections describe the new aspects of the concept: the first provides the facility to specify issue workflows based on very simple constraints without modeling the entire workflow. The second covers the automatic inference of additional constraints to construct workflows from subsets of the specified activities (since in different real situations not all activities will be necessary). The third introduces concepts that enable the specification of more complex workflows.

A. *Basic Workflow Modeling*

The candidate activities and their relations are modeled in the ontology, whereas the system later automatically generates executable workflows utilizing SME as described in our initial approach [11]. There, the following constraints were supported: 'before' and 'after' specifying a sequential relation between activities; 'required before' and 'required after' specifying that one activity requires the presence of another one; and 'mutual exclusion' specifying that two activities cannot occur together for the same workflow instance. These basic constraints are now extended and separated into different categories. Table I shows the currently supported constraints.

TABLE I. SEQUENCING CONSTRAINTS

Constraint	Meaning	Type
X hasSuccessor Y	if X and Y are present, X should appear before Y	sequencing
X hasParallel Y	if X and Y are present, they should appear parallel (like two branches that are connected by AND gates in classical process modeling)	sequencing
X requires Y	if X is present, Y must also be present	existence
X mutualExclusion Y	if X is present the presence of Y is prohibited	existence

The fusion of existence constraints (governing which activities are permitted in one workflow instance) with sequencing constraints (governing their arrangement) is now eliminated. Thus, the building of the workflow is separated into phases: one that checks the existence constraints to determine which activities are in place for the current workflow and, when the set of chosen activities is consistent, one that sequences them utilizing the sequencing constraints. Additionally, a constraint for parallel activities is added.

A simple example for the specification of a constraint-based workflow is shown in Fig. 2. It comprises the concurrent comparison and merging of two source code files. Here, only one sequencing and two existence constraints are needed to ensure that both activities are present in the workflow. The two existence constraints ensure that both activities are in place and the sequencing constraint governs how they shall be executed.

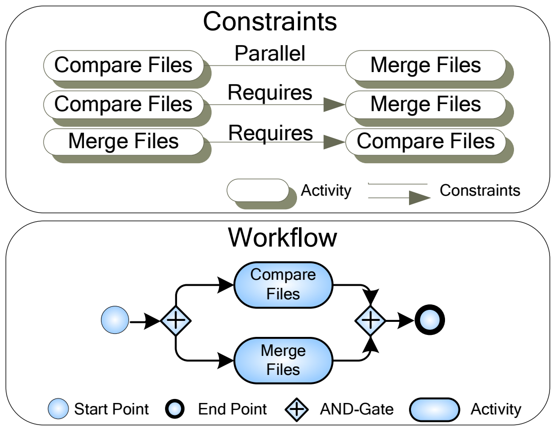

Figure 2. Workflow specification example for manually merging two source code files.

The candidate activities and the constraints are used to build workflows to automatically govern these activities and thus assist the user. Thus, different conditions have to be satisfied to be able to build a coherent workflow graph out of the activities:

Condition C1: Each workflow shall have a clear start point and end point. This promotes simple and understandable models as suggested in [19].

Condition C2: Each activity shall have at least one connection to other activities. This condition ensures that workflows are buildable, as a workflow cannot be built from unconnected activities since it cannot be determined when to execute this activity.

Condition C3: No cyclic sequencing shall be specified (this limitation enables simple modeling and workflow generation. In Section II.3, an activity modeling extension is added, enabling the definition of loops).

Condition C4: The activity structure shall be simple. An activity shall have only one successor and one predecessor. If multiple successors are needed, one can be defined as successor and the other shall be specified as parallel to that successor. This limitation enables specifying and building very simple workflows and is also addressed by the extended activity modeling in Section II.3.

Condition C5: An activity x shall not both require and mutually exclude the same activity.

On this basis, simple workflows are constructed utilizing the algorithm in Section III.

B. *Automated Workflow Completion*

In our initial approach, specification of bilateral connections between all activities was necessary since, based on the SME properties, any subset of these activities could be selected and it was required that a distinct workflow could be built from each subset. This proved to be a cumbersome modeling task. Therefore, we now integrate an auto-completion feature to infer the connections that were not automatically specified. Based on the defined conditions, this is possible and illustrated in Fig. 3.

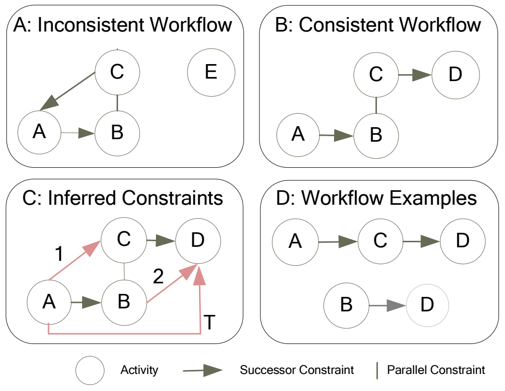

Figure 3. Workflow completion.

Fig. 3B shows a consistent workflow specified by the user in the ontology: Activity B should be executed after A, C should be parallel to B, and D should be executed after C. Fig. 3A illustrates an inconsistently specified workflow with a cyclic dependency. Fig. 3C depicts the connections automatically added by the system. Fig. 3D shows two examples of the workflows that can be created out of a subset of the activities from Fig. 3B using the inferred relations. Note that in future work a GUI will be provided for user modeling of workflows.

C. *Extended Activity Modeling*

The solution presented in the preceding sections supports simple activity sequence modeling and the generation of workflows for all possible activity subsets. Yet the modeling is too simplistic for the specification of complex workflows. Consequently, we introduce the concept of the *BuildingBlock* that is used in place of a simple activity in the modeling of workflows. It can be a single activity or a more complex construct:

- The *BuildingBlock* representing exactly one activity,
- The *Sequence* representing a sequence of other *BuildingBlocks*,
- The *Parallel* representing the parallel execution of other *BuildingBlocks*, or
- The *Loop* representing the repeated execution of other *BuildingBlocks,* allowing the specification of cyclic structures in a consistent way.

Thus, higher-level structures become possible and the workflow can be hierarchically decomposed. Each *BuildingBlock* is treated as an activity, hiding the complexity of the structure within it. This yields several advantages: defined *BuildingBlocks* can be more easily reused and activity structures with certain SME properties can be connected. Furthermore, workflow completion and workflow generation logic can retain simplicity by working recursively with *BuildingBlocks*. Fig. 4 illustrates how nested *BuildingBlocks* can create a more complex workflow structure. The nesting of *Sequence*, *Parallel,* and *Loop* creates the workflow structure on the right. Each of the concepts is treated as a simple activity from the outside. This way of modeling enforces proper nesting of workflow patterns as suggested in [19].

Currently the approach only supports *Loop*, *Parallel,* and *Sequence BuildingBlocks* and 'mutual exclusion' and 'require' constraints. Thus, the selection of executed activities is performed based on the parameters of the situation. To also enable the user to select activities or other parameters, a new *BuildingBlock Conditional* will be developed as part of future work to allow a user to select an activity.

To preserve the ability to consistently and simply build workflows with the specified constraints, conditions are defined for *Loop*, *Sequence,* and *Parallel*:

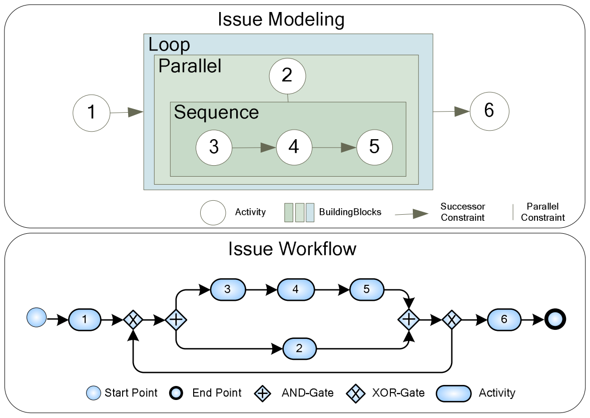

Figure 4. BuildingBlocks.

Condition C6: A *Loop* shall only contain one *BuildingBlock.* This can be a simple activity or any other *BuildingBlock*, thus enabling the looping of any structures. This simplifies the conversion to a workflow structure while allowing arbitrary structures to be looped using the *BuildingBlocks*.

Condition C7: A *Parallel s*hall contain at least two *BuildingBlocks.*

Condition C8: A *Parallel* shall contain only *BuildingBlocks* that are connected in parallel. Conditions C7 and C8 are introduced to avoid unnecessary complex modeling since they impose that a Parallel only contains two or more parallel *BuildingBlocks*. The latter enables the parallelization of arbitrary structures.

Condition C9: A *Sequence* shall contain at least two *BuildingBlocks*.

Condition C10: A *Sequence* shall contain only sequentially connected *BuildingBlocks*.

Condition C11: A *Sequence* shall contain a clear start and end point. As with conditions C7 and C8 for the *Parallel*, conditions C9, C10, and C11 shall ensure clear modeling for the *Sequence*.

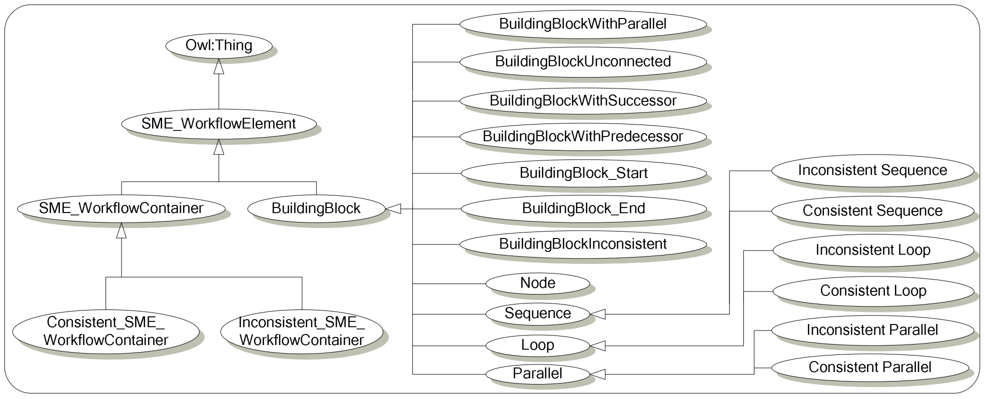

Figure 5. Ontology structure.

III. Realization

In the technical realization, the concepts for issue workflow specification are realized in the ontology, exploiting the logical capabilities of semantic technology. For the realization of the workflow specification, the ontology as well as its reasoning capabilities and logic (DL) was chosen to provide access to the many facts describing the current situation in the SE project. These facts are provided by our CoSEEEK framework that integrates a rich set of Hackystat [20] sensors to gather information [21], aggregate it, and store high-level events within the ontology. The selection of appropriate activities for various situations connecting contextual facts with SME properties is described in [11].

The reasoner can automatically classify workflows and their contained *BuildingBlocks* as consistent or inconsistent. Inconsistent items violate at least one of the defined conditions and are thus rejected by the system. To enable this classification, various specialized concepts have been created as depicted in Fig. 5. The ontology concepts are grouped into either template concepts that define an issue with all its potential characteristics, and the associated concrete concepts (not shown) holding the data for each concrete issue workflow execution. The top-level concept is the *SME_WorkflowElement* that has two disjoint children, the *SME_WorkflowContainer* that is equivalent to an issue workflow, and the *BuildingBlock* that realizes the elements of such a workflow. The *SME_WorkflowContainer* contains a number of *BuildingBlocks* by means of the *managesBuildingBlocks* object property. The *BuildingBlock* realizes the sequencing constraints via the following object properties: *hasParallel*, *hasSuccessor*, *mutualExclusion*, and *requirement*. A sequential relationship is always bilateral and thus affects the predecessor and successor *BuildingBlock*. For easier processing, an additional constraint *hasPredecessor* is introduced, which is defined as the inverse property of *hasSuccessor*. In addition, the two properties *hasInferredSuccessor* and *hasInferredPredecessor* are introduced to represent connections added by the reasoner to the workflows. Thus, the modeling of a single successor/predecessor (Condition C4) can be enforced on the *hasSuccessor* / *hasPredecessor* properties and the automated workflow completion feature can still add new connections as depicted in Fig. 3C. These properties are both transitive. Property *hasParallel* is defined as symmetric (as specified in OWL) since this always applies to all of the *BuildingBlocks* connected by the property. The same applies for *mutualExclusion*. There are also subclasses of the *BuildingBlock* realizing the *Loop*, *Sequence*, and *Parallel* elements. Each has an additional object property for other contained *BuildingBlocks*. The other various subclasses are described later. The approach utilizes these subclasses for automatic classification of the different concepts as consistent (as exemplified in Fig. 3B) or inconsistent (as exemplified in Fig. 3A). In addition, SWRL rules are used for conditions not supported in OWL and for the automatic addition of connections to workflows. The procedure for issue workflow creation follows:

Issue workflow creation

(1) The user specifies the workflow as illustrated in Fig. 3B.
(2) The reasoner executes the SWRL rules for consistency checking.
(3) The reasoner classifies the specified concepts and checks the conditions (specified using the different subclasses explained in Section III.1).
(4) The workflow is completed as depicted in Fig. 3C, first executing the SWRL completion rules that add new *hasParallel* and *hasInferredSuccessor* / *hasInferredPredecessor* connections. Thereafter, the reasoner adds further sequential connections using the transitive definition of the *hasInferredSuccessor* / *hasInferredPredecessor* properties.
(5) The check for cyclic dependencies (Condition C3) is performed utilizing the *hasInferredSuccessor* / *hasInferredPredecessor* and *hasParallel* properties. (According to OWL DL restrictions, the *hasSuccessor* / *hasPredecessor* properties cannot be transitive while restricting their cardinalities at the same time.)
(6) The workflow is verified, completed, and connected to issue templates and SME property templates to be used to govern concrete issues. These templates are also realized in the ontology: issue templates define various issues that may occur in a SE project, like bug fixing or refactoring; and SME property templates realize different properties of the situation, like risk or urgency. Typically, only a subset of the specified activities are executed (as shown in the two examples in Fig. 3D).

A. *Workflow Consistency Checking*

The process of workflow consistency checking entails SWRL rule execution followed by taxonomy classification by the reasoner. OWL axioms and SWRL rules are used for condition verification for the workflow construction algorithm as explained below. Due to space limitations, only a selection of the conditions is described.

Condition C1: To check whether a unique start and end point are specified, the *BuildingBlock* has two subclasses, *BuildingBlock_Start* and *BuildingBlock_End*. A *BuildingBlock* is classified as a *BuildingBlock_Start* if it has no predecessor. If multiple parallel *BuildingBlocks* are executed at the beginning of the workflow, none should have a predecessor. The same applies to *BuildingBlock_End* and successors:

$$\begin{gathered} \mathit{BuildingBlock\_Start} \equiv \mathit{BuildingBlock} \wedge \neg\exists \mathit{hasPredecessor} \\ \wedge \neg\exists \mathit{hasParallel}.\mathit{BuildingBlockWithPredecessor} \end{gathered}$$

And two concepts define a *BuildingBlock* with a successor or predecessor:

$$\begin{gathered} \mathit{BuildingBlockWithPredecessor} \equiv \mathit{BuildingBlock} \wedge \exists \mathit{hasPredecessor} \\ \mathit{BuildingBlockWithSuccessor} \equiv \mathit{BuildingBlock} \wedge \exists \mathit{hasSuccessor} \end{gathered}$$

To validate a modeled workflow, the concepts *Consistent_SME_Workflow_Container* and *Inconsistent_SME_Workflow_Container* are used as shown in Condition C2. The condition is that if a container contains two *BuildingBlock_Start* individuals that are not connected in parallel, it is an inconsistent container. Currently the check is implemented programmatically via the Jena framework.

Condition C2: For detecting an unconnected *BuildingBlock*, as exemplified in Fig. 3A, the *BuildingBlock_Unconnected* is introduced:

$$BuildingBlock\_Unconnected \equiv BuildingBlock \wedge \neg\exists hasParallel \wedge \neg\exists hasPredecessor \wedge \neg\exists hasSuccessor$$

A workflow container should contain a starting and an ending *BuildingBlock* and not contain unconnected *BuildingBlocks* unless there is only one of them in the container, meaning it is detected to be the start as well as the end of the workflow. A workflow would also be inconsistent if containing any inconsistent concept:

$$\begin{aligned}&Inconsistent\_SME\_Workflow\_Container \equiv \\ &SME\_WorkflowContainer \\ &\wedge (managesBuildingBlock = 1 \wedge \neg\exists managesBuildingBlock. \\ &(BuildingBlock\_Start \vee BuildingBlock\_End \\ &\vee BuildingBlock\_Unconnected)) \\ &\vee \exists managesBuildingBlock. \\ &(Inconsistent\_BuildingBlock \vee Inconsistent\_Sequence \\ &\vee Inconsistent\_Parallel \vee Inconsistent\_Loop)\end{aligned}$$

$$\begin{aligned}&Consistent\_SME\_Workflow\_Container \equiv \\ &SME\_Workflow\_Container \\ &\wedge \exists managesBuildingBlock.BuildingBlock\_Start \\ &\wedge \exists managesBuildingBlock.BuildingBlock\_End\end{aligned}$$

Condition C3: This condition is specified using SWRL rules. Simplistically modeled, three cyclic dependencies can be specified: specifying the successor of a *BuildingBlock* as also its predecessor can be done with the *hasSuccessor* / *hasPredecessor* constraints as well as involving the *hasParallel* constraint. If a *BuildingBlock* A has a successor B that has a parallel *BuildingBlock* C, specifying A as the successor of C would imply a cyclic dependency. That case is shown in Fig. 3A. Another possibility is to have *BuildingBlocks* A and B parallel and C as successor of B as well as predecessor of A. To capture this, three rules are specified:

Cyclic_Dependency_Seq: hasInferredSuccessor(?x, ?y) ∧ hasInferredSuccessor(?y, ?x) → Problem(?x, "yes")
Cyclic_Dependency_Par1: hasInferredSuccessor(?x, ?y) ∧ hasParallel(?y, ?z) ∧ hasInferredSuccessor(?z, ?x) → Problem(?x, "yes")
Cyclic_Dependency_Par2: hasInferredSuccessor(?x, ?z) ∧ hasParallel(?x, ?y) ∧ hasInferredSuccessor(?z, ?y) → Problem(?x, "yes")

The simple specification of the rules is possible due to the transitive definition of the *hasInferredSuccessor*, *hasInferredPredecessor*, and *hasParallel* constraints. If a cyclic dependency exists, the rules set the property *Problem* of the *BuildingBlock*, which lets the reasoner classify the *BuildingBlock* as inconsistent using the *BuildingBlock_Inconsistent* concept. Consequently, the workflow would also be classified as inconsistent. Modeling both the consistent as well as the inconsistent cases facilitates inference regarding the consistency of the workflow.

B. *Automated Workflow Completion*

SWRL rules realize the automated workflow completion, which are executed if the workflow is consistent and add sequential connections to the *BuildingBlocks*. For these new connections, the two additional properties *hasInferredSuccessor* and *hasInferredPredecessor* are used since the other sequential constraints are restricted to only one element. If two *BuildingBlocks* are parallel and one of them has a successor, rule C1 assigns that successor also to the other one. If a *BuildingBlock* has a successor that has a parallel *BuildingBlock*, rule C2 also adds the latter to the successors of the first *BuildingBlock*. Rule C3 adds the initial successor to the *hasInferredSuccessor* property, which now contains all successors of a *BuildingBlock*:

C1: hasSuccessor(?x, ?y) ∧ hasParallel(?y, ?z) → hasInferredSuccessor(?x, ?z)
C2: hasSuccessor(?x, ?y) ∧ hasParallel(?x, ?z) → hasInferredSuccessor(?z, ?y)
C3: hasSuccessor(?x, ?y) → hasInferredSuccessor(?x, ?y)

As illustrated in Fig. 3C, the application of rules C1 and C2 are marked with '1' and '2'. Marked with a 'T' is the third inferred constraint, which exploits the transitivity of the properties as described in step four of the issue workflow creation procedure. In this way, complex workflows are enabled using specialized *BuildingBlocks,* but the basic structure of the workflows remains simple.

C. *Workflow Instantiation*

After an issue workflow is specified, consistency checks applied, and the workflow generated, it can be used for concrete issues. As described in [11], based on the current situation a subset of activities is selected for the workflow. First, the existence constraints (e.g., mutual exclusion) are checked for the chosen subset of activities to ensure that required activities are not omitted in the chosen subset. If that is the case, two alternatives can be considered: required activities are added to the subset or the workflow generation is aborted. When the workflow construction algorithm shown in Listing 1 is started, all *BuildingBlocks* (or parts thereof) have been selected according to the properties of the situation (like the risk level or urgency) in an unordered list (`allBBs`) from which the real workflow is generated. First the starting point of a workflow is determined, which can be one or multiple parallel *BuildingBlocks* without predecessors, and these are removed from the list. These are added to the final workflow object that is later passed to process management for workflow generation. The selection is unambiguous since all conditions for a usable workflow were previously verified. All remaining *BuildingBlocks* in the list are thus successors of this selection. Therefore, *BuildingBlocks* without predecessors in the list are again determined as the direct successors of the starting *BuildingBlocks* (nextBBs) and added to the workflow object. This is repeated until the initial list is empty.

Listing 1. Workflow generation algorithm (in pseudo code).

```
startBBs = determineBBwithNoPredecessors(List
allBBs);
buildingBlockTreatment(startBBs, allBBs)
add startBBs to final workflow object
while(allBBs not empty)
   nextBBs = determineBBwithNoPredecessors(allBBs);
   buildingBlockTreatment(nextBBs, allBBs)
   add nextBBs to final workflow object
pass final workflow object to process management

determineBBwithNoPredecessors(list)
   while (BB not found)
      check if element has no predecessors in list
      if(no predecessors)
         get BBs that are parallel to element
         return BBs without predecessors and remove
from list

buildingBlockTreatment(list1, list2)
   for (elements in list1)
      getContainedElements(element, list1, list2)

getContainedElements(BuildingBlock, list1, list2)
   get all elements from list2 that BuildingBlock
   contains
   for each element
      getContainedElements(BuildingBlock, list1,
      list2)
      add element to list1
      remove element from list2
```

For *BuildingBlocks* that are not a simple activity, the function *buildingBlockTreatment* recursively retrieves all contained elements from the initial list and marks them according to the *BuildingBlock* that contained them. Thus, the resulting workflow object is structured in a way that enables block-structured generation of the workflow comprising all workflow patterns (e.g., loops).

IV. Improved Scenario

To exemplify our proposed way of modeling issues, we now show the workflow snippet of Fig. 1 modeled utilizing our approach. This is illustrated in Fig. 6.

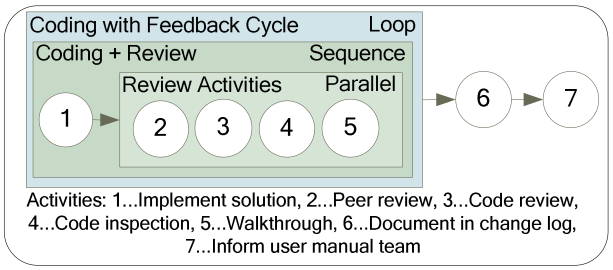
Coding with Feedback Cycle
Loop
Coding + Review
Sequence
Review Activities
Parallel
1
2
3
4
5
6
7
Activities: 1...Implement solution, 2...Peer review, 3...Code review, 4...Code inspection, 5...Walkthrough, 6...Document in change log, 7...Inform user manual team

Figure 6. Issue modeling for bug fixing.

This example illustrates several advantages of our approach. The review activities are combined in a *Parallel* block. Thus, it is possible to choose none, one, or several according to the situation, whereas the initial example in Fig. 1 rigidly prescribed the selection of one of these. The review activities are then combined with the coding activity in a *Sequence* block. The *Sequence* is nested in a *Loop* block. All of these blocks or activities can be individually reused in other issue workflows. Thus, our approach facilitates the establishment of a method library containing *BuildingBlocks* for different purposes such as a coding activity with an integrated feedback cycle. From this library, new workflow templates can be readily created, which the system, in turn, uses to custom build workflow instances matching the properties of different situations (such as risk, urgency, criticality, etc.). On the top level, this snippet contains only the three sequentially executed activities 'Coding with Feedback Cycle', 'Document in Change Log', and 'Inform User Manual Team'. Therefore, having a method library in place, simple workflows can be specified. Since each of the *BuildingBlocks* is connected with SME properties, they are only executed if the situation requires it. Thus, it is also possible to combine two review activities while the initial scenario workflow rigidly prescribed one activity.

V. Evaluation

To evaluate the general suitability of each of the defined conditions, test cases for C1-11 with various specific activities and constraints are specified. This is depicted in Table II. The correct classification of the test case inputs by the reasoner (consistent vs. inconsistent workflows) was verified by a human expert.

TABLE II. Evaluation of Issue Workflows and their Constraints

Issue Workflows			
W1	**W2**	**W3**	**W4**
n18 --> L1	n35	n26 - n27	n42 --> n43
L1 - n19	n36 --> n37	n27 - n28	n44 --> P2
n19 --> n20	n37 - P3	n28 --> L0	P2 --> S2
n20 - S1	P3 --> n36	L0 --> n29	S2 - n45
n20 - n21	n37 --> n38	n29 - n30	S2 - n46
n20 --> n22	n38 --> L2	n29 - n31	n45 --> S3
n22 --> n23	n38 --> n39	n31 --> n32	n46 --> n47
n23 --> n24	n39 --> S4	n30 --> n33	S3 M n45
n24 --> n25	L2 --> n40	n33 --> n34	S3 R n45
	n40 - n41		
BuildingBlocks			
L0	**P1**	**P3**	**S2**
S1	n3 - n4	n8	n11 --> 12
L1	n4 - n5	**S3**	n13 --> n14
P1	**P2**	n15 - n16	**S1**
L2	n6 --> n7	S4	n9 --> P1
n1 - n2		n17	P1 --> n10

Sequencing constraints (as defined in Table I) are specified for four issue workflows (W1-4) as well as for *Loop* blocks (L0-2), *Parallel* blocks (P1-3), *Sequence* blocks (S1-4), and simple activities (n1-n45) that were used in the workflow sets. '-->' stands for a sequential constraint, '-' for

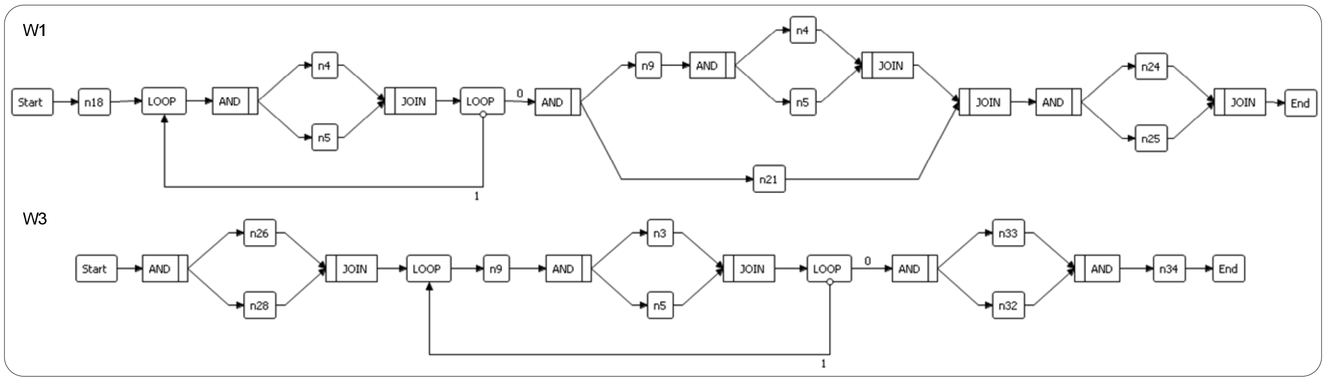

Figure 7. Workflows W1 and W3 as evaluated via AristaFlow.

a parallel constraint, 'M' for mutual exclusion, and 'R' for a requirement. The different components have been specified so that all conditions could be tested. For example, the Condition C7 is violated by P3 containing only one element and the Condition C1 is violated by W4 having more than one starting point. The condition evaluation is shown in Table III, where W2 and W4 violate various conditions and are rejected whereas W1 and W3 are accepted by the system.

TABLE III. WORKFLOWS (W) VS. CONDITIONS (C) WITH VIOLATIONS MARKED WITH AN X.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
W1	o	o	o	o	o	o	o	o	o	o	o
W2	o	x	x	x	o	x	x	o	x	o	o
W3	o	o	o	o	o	o	o	o	o	o	o
W4	x	o	o	o	x	o	o	x	o	x	x

For workflow construction, only a subset of all possible activities specified for a workflow is chosen based on SME properties. To evaluate workflow constructability of the consistent sets W1 and W3, an activity subset was arbitrarily selected as a test case (the activities underlined in red in Table II were omitted). The concrete workflows were constructed in AristaFlow as depicted in Fig. 7. The structural correctness of the workflows was verified via AristaFlow's correctness by construction technique [8] that prohibits the building of incorrect workflows.

This initial evaluation demonstrates the feasibility of a semantically-driven workflow generation using declarative modeling approach with regard to structural condition suitability and workflow constructability. Prior work [10][11] has already considered performance and scalability measurements of the different components of the system. Future work will evaluate the approach in live project usage with our industrial project partners.

VI. RELATED WORK

Several approaches consider the combination of semantic web and process management technology. [22] provides a comparison of different technical realizations. Process models have been combined with semantic concepts for various reasons: [23] combines semantic web services with BPM to provide a unified view on the organizational process space, while [24] provides a semantic business process repository for automating the business process lifecycle. The automated monitoring of business processes is addressed in [25]; it utilizes a combination of semantic and agent technology to facilitate effective management and evaluation of business processes. These approaches consider the semantic enhancement of process descriptions with various goals that all deal with some type of process analysis. The CoSEEEK approach uses semantic technology not only for analysis of given process models but also including machine-readable semantics to the process models to enable the system to actively manipulate and construct process models automatically, thus exploiting a capability of semantic technology.

Declarative approaches for workflow modeling like DECLARE [26] and ALASKA [27] focus on the logic that governs the interplay of actions in the process by describing: (1) activities that can be performed, and (2) the constraints prohibiting undesired behavior; i.e., workflows are modeled in a constraint-based fashion and, in a given instance state, all activities are offered to users that do not violate the given set of constraints. In comparison, CoSEEEK scales to larger workflow structures and supports constraint modeling with standard workflow structures that are dynamic. Yet the most significant difference is that our approach not only enables declarative modeling of candidate activities, but also supports the automated selection of activity subsets based on context knowledge and SME.

Regarding automated workflow adaptations, Agentwork [28] utilizes agent technology with event-condition-action rules to automatically adapt workflows for process exception handling. It does not deal with constructing workflows based on situational information.

To support end users during process execution and to make dynamic recommendations on possible next steps (i.e., activities), the information available in event logs can be exploited [29][30]. Such a recommendation service mines the log for already completed process instances (i.e., log traces) similar to the current process instance. These log traces are then used for calculating recommendations which, based on the historic information, can be expected to best attain certain performance goals. More precisely, when completing an activity during process execution, the recommendation service creates a list of ranked activities and offers it to a user who then selects the next activity to be executed.

For engineering workflows, [31][32][33] dynamically create a workflow schema from a given product structure and automatically adapt it if the product structure has changed at runtime. Graph rewriting and AI planning techniques are used. However, only simple workflow structures (e.g., no loops and conditional branches) can be handled.

VII. Conclusion

The main contribution of this paper is a declarative workflow modeling approach driven by semantic technology to support contextual workflow construction from candidate activities. The approach improves various aspects for the modeling and automated construction of situational workflows that can be utilized for evolutionary process management support. Complex workflow structures that incorporate every possible situation become unnecessary. The user only needs to specify the possible activities, with the system selecting the appropriate subset for each situation (as described in [11]) and automatically constructing the concrete workflow and adapting it in alignment with the current situation. Limitations of the prior approach have been removed, enabling more complex workflows to be generated.

The newly introduced concept of *BuildingBlocks* yields the advantages of simplified modeling and easier reuse. They encapsulate and hide the contained structure of activities and can be used as simple activities. For modeling issue treatment, defining *BuildingBlocks* that combining atomic activities and connecting them to SME properties builds a method library. For reuse, this library can be used to more easily generate new issue specifications with a simple structure. The system then chooses the appropriate activities based on situational information. Thus, the complexity and large number of activities associated with an issue are hidden during modeling and are not involved in the actual workflow once generated and then executed. That way it is possible to model and execute many of the dynamic activities that are executed outside of the development process in the SE domain. Since the inherent complexity of the approach is hidden within the system, it is possible to offer a simple way of modeling *extrinsic workflows* to the user.

Future work will include providing a GUI to allow the user to specify workflow constraints without directly accessing the ontology. Industrial application of the approach at the partner companies of the project is planned. The modeling will be extended and tailored to the partner situations and the practicality of the approach evaluated as case studies. Planned are also a *BuildingBlock* '*Conditional'* for the conditional execution of activities that rely on user decisions via the GUI, a predefined *BuildingBlock* library to support the users in modeling various issues, and a 'case learning' feature to record unspecified issues as they are executed.

Acknowledgment

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

References

[1] Yourdon E., 'Death March,' 2nd Ed. Prentice Hall, Upper Saddle River, NJ, 2004.

[2] Gruhn, V., 'Process-Centered Software Engineering Environments, A Brief History and Future Challenges,' Annals of SW Engineering, Springer, v. 14(1-4), 2002, pp. 363-382.

[3] Gibson, D., Goldenson, D., Kost, K., 'Performance Results of CMMI-Based Process Improvement,' Carnegie Mellon SEI, CMU/SEI-2006-TR-004, 2006.

[4] Lenz, R., Reichert, M., 'IT Support for Healthcare Processes - Premises, Challenges, Perspectives,' Data and Knowledge Engineering, 61(1), 2007, pp. 39-58.

[5] Müller, D., Herbst, J., Hammori, M., Reichert, M., 'IT Support for Release Management Processes in the Automotive Industry,' Proc. BPM'06, LNCS 4102, 2006, pp. 368-377.

[6] OpenUp http://epf.eclipse.org/wikis/openup/

[7] Rausch, A., Bartelt, C., Ternité, T., and Kuhrmann, M.: 'The V-Modell XT Applied - Model-Driven and Document-Centric Development'. Proc. 3rd World Congress for Software Quality, 2005.

[8] 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), 2009, pp. 81-97.

[9] Grambow, G. and Oberhauser, R., 'Towards Automated Context-Aware Selection of Software Quality Measures,' Proc. 5th Intl. Conf. on SW Eng. Adv., 2010, IEEE CS.

[10] Grambow, G., Oberhauser, R., and 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.

[11] Grambow, G., Oberhauser, R., and Reichert, M., 'Semantic Workflow Adaption in Support of Workflow Diversity,' Proc. 4th Int'l Conf. on Advances in Semantic Processing, 2010.

[12] Ralyté; J., Brinkkemper, S., Henderson-Sellers, B. (Eds.), 'Situational Method Engineering: Fundamentals and Experiences,' Proc. IFIP WG 8.1 Working Conf., 2007.

[13] Oberhauser, R., 'Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments,' In G. Wu (ed.) Semantic Web, In-Tech, 2010, pp. 157-179.

[14] Reichert, M. et al, 'Enabling Poka-Yoke Workflows with the AristaFlow BPM Suite,' In: CEUR Proc. of the BPM'09 Demo Track, Business Process Management Conf., 2009.

[15] World Wide Web Consortium, 'OWL Web Ontology Language Semantics and Abstract Syntax,', 2004.

[16] World Wide Web Consortium, 'SWRL: A Semantic Web Rule Language Combining OWL and RuleML,' W3C Member Submission, 2004.

[17] Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y., 'Pellet: A practical OWL-DL Reasoner,' Journal of Web Semantics, 2006.

[18] McBride, B., 'Jena: a semantic web toolkit,' Internet Computing, 2002.

[19] Mendling, J., Reijers, H.A., van der Aalst, W.M.P., 'Seven process modeling guidelines (7PMG),' Information and Software Technology 52, 2010, pp. 127-136.

[20] Johnson, P.M., "Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System," Proc. of 1st Int. Symposium on Empirical Software

Engineering and Measurement, IEEE Computer Society Press, 2007.

[21] Oberhauser, R.. "Leveraging Semantic Web Computing for Context-Aware Software Engineering Environments," In "Semantic Web", Gang Wu (ed.), In-Tech, Vienna, Austria, 2010.

[22] Lautenbacher, F. and Bauer, B., 'A Survey on Workflow Annotation & Composition Approaches,' Proc. Workshop on Semantic Business Process and Prod. Lifecycle Mgmt. (SemBPM) at the European Semantic Web Conference (ESWC), 2007,pp. 12-23.

[23] Hepp, M. et al, 'Semantic business process management: a vision towards using semantic Web services for business process management,' Proc. Int'l Conf. e-Business Eng., 2005.

[24] Ma, Z., Wetzstein, B., Anicic, D., Heymans, S., 'Semantic Business Process Repository,' Workshop on Semantic Business Process and Product Lifecycle Management, 2007.

[25] Thomas, M., Redmond, R., Yoon, V., Singh, R., 'A semantic approach to monitor business process,' Commun. ACM 48, 2005, pp. 55-59.

[26] Pesic, M., Schonenberg, H., van der Aalst, W.M.P., 'DECLARE: Full Support for Loosely-Structured Processes. Proc,' Int'l Enterprise Distr. Obj. Comp. Conf., 2007, pp. 287-298.

[27] Weber B., Zugal S., Pinggera J., Wild W., 'Experiencing Process Flexibility Patterns with Alaska Simulator,' Proc. BPM'09 Demos, 2009.

[28] Müller, R., Greiner, U., Rahm, E., 'AGENTWORK: A Workflow-System Supporting Rule-Based Workflow Adaptation,' Data Knowl. Eng. 51(2), 2004, pp. 223-256.

[29] Schonenberg, H., Weber B., van Dongen B. F., van der Aalst, W.M.P., 'Supporting Flexible Processes through Recommendations Based on History,' LNCS v.5240, 2008, pp. 51-66.

[30] Haisjackl, C., Weber, B., 'User Assistance During Process Execution - An Experimental Evaluation of Recommendation Strategies,' Proc. BPM'10 Workshops (BPI'10), 2010.

[31] Heimann, P., Joeris, G., Krapp, C.-H., Westfechtel, B., 'DYNAMITE Task Nets for Software Process Management,' Proc. 18th Int'l Conf. SW Eng., 1996, pp. 331–341.

[32] Conradi, R., Liu, Ch., Hagaseth, M., 'Planning Support for Cooperating Transactions in EPOS,' Information Systems, Vol. 20, No. 4, 1995, pp. 317–336.

[33] Müller, D., Reichert, M., Herbst, J., 'A New Paradigm for the Enactment and Dynamic Adaptation of Data-driven Process Structures,' in: 20th Int'l Conf. on Advanced Information Systems Engineering (CAiSE'08), Montpellier, France, Springer, 2008.