From Software Engineering Process Models to Operationally Relevant Context-aware Workflows: A Model-Driven Method

Roy Oberhauser
Computer Science Dept.
Aalen University
Aalen, Germany
roy.oberhauser@htw-aalen.de

Abstract - **Software engineering (SE)-specific process models and their notation, such as the Software & Systems Process Engineering Metamodel, are typically not specified or available in an executable form that can provide automated guidance in human-centric software engineering workflows. These SE process models generally remain abstract in order to be broadly applicable, and when any are concretized, they often exist only in the form of documentation. Thus, they are not actually relevant operationally, affecting process utilization and governance. On the other hand, common business process modeling notation such as BPMN is generalized and not conducive for providing the context-aware support needed for executable SE workflows. Thus, a practical method is needed that supports comprehensive SE process documentation, yet also provides an SE workflow modeling capability that can transform documented SE workflows into an enactable form executable in today's workflow management systems. The method presented in this paper can utilize an available comprehensive SE process documentation meta-model and automatically extract incorporated SE concepts and workflow concepts to a workflow model, specifically the Software Engineering Workflow Language (SEWL). From this the following are supported: 1) graphical-based workflow modeling, 2) model-based transformation of workflow concepts to diverse workflow management systems (WfMS), and 3) the semantic transformation of SE concepts to contextually-aware process-centered software engineering environments. The results show the viability and practicality of such a method to document, extract, graphically model, transform, and enact SE workflows in support of contextual guidance capabilities for software engineers.**

Keywords - process-centered software engineering environments; software engineering environments; software engineering process modeling; software engineering process model transformation; SPEM; UMA; Unified Method Architecture; model-driven software development.

I. INTRODUCTION

This article extends our previous work in [1]. In order to be generally applicable to various software development projects, most software engineering (SE) process models remain abstract and require tailoring to the specific project, team, and tool environment. Examples of SE process models include the V-Model XT [2] (specified for all public-sector IT development in Germany) and the Open Unified Process (OpenUP) [3]. Typical SE process models are documented to a great extent in natural languages, and are thus not easily executable in an automated form. The technical implementation of an executable process, whose sequence can be modeled with and automatically enacted in a workflow management system (WfMS), is called a workflow. SE workflows, many of which are human-centric, can cover some sequence of activities and steps related to requirements, design, testing, etc., for instance Activity Flows in VM-XT [4] or workflows in OpenUP [5].

Because they integrate SE concepts, SE process meta-models can be useful in the modeling and comprehensive documentation of such SE processes. For instance, the Eclipse Process Framework (EPF) [6] is an open source project for software process engineering that provides a framework and supporting tools, one being the EPF Composer (EPFC) [7] for method and process authoring and publishing. It utilizes a process meta-model, the Unified Method Architecture (UMA), a large extent of which was adopted into the Software & Systems Process Engineering Metamodel (SPEM) 2.0 [8].

Additionally, process-centered software engineering environments (PCSEEs) have attempted to investigate and address automated guidance and assistance mechanisms for SE processes [9]. Yet they remain intrusive, rigid, and inflexible [10], and fail to adequately support the human, creative, and dynamic aspects of software development. While more generalized automated process assistance and guidance for humans has been available in the form of process-aware information systems (PAIS) [11], this area has lacked satisfactory standards and SE support and often lacks the integration of the project and human context. Thus, such systems and capabilities have not been readily leveraged by software engineers.

A. Our Previous Work

To address these challenges for such human-centric SE processes, we created a PCSEE that we call the Context-aware Software Engineering Environment Event-driven Framework (CoSEEEK) [12]. Beyond SE tool sensors and other contextual knowledge, it utilizes workflows to understand the process context. That includes knowing which activities a software engineer performed, which activity is likely currently being worked on, which activity is next, and associating these with SE-specific concepts such as projects, teams, persons, roles, tools, and artifacts via an ontology and reasoner. While various facets were investigated, including collaboration [13], quality integration [14], and others, we still faced the problem of providing an easy way for software engineers to access, model, and transform SE workflows and integrate SE concepts without vendor lock-in to a specific WfMS. Considering possible SE workflow modeling notation, the SPEM is aimed primarily at defining a domain-specific notation for the documentation of SE processes, and does not completely address issues related to executable SE processes so that automatic support and guidance for software engineers in operational activities can occur. On the other hand, a general workflow language notation such as the Business Process Model and Notation (BPMN) 2.0 [15], while executable, lacks the inclusion and semantic meaning of various SE domain-specific concepts and thus becomes cumbersome.

Thus, to address the executable SE workflow language gap, our team created the text-based language SEWL [16] and previously targeted the adaptive WfMS AristaFlow [17] and YAWL [18] to evaluate its portability. Our previous work in [1] contributed various extensions to the original workflow concepts, including: a new graphical representation for SE-specific workflows blending BPMN and SPEM notation; a graphical editor for SE workflows; details on the model-driven generation of tailored artifacts that target the ontology and heterogeneous WfMS support, specifically the common of-the-shelf (COTS) WfMS jBPM [19] and Activiti [20]; and the workflow ontology generator, which addresses the aspect of contextual-awareness support for workflows in conjunction with CoSEEEK.

B. *Contribution*

This article extends our work in in [1] by expanding the scope of the original solution approach. It contributes an automated model-driven method for SE process modelers that incorporates a standard SE process meta-model, namely the UMA, thus supporting comprehensive SE process documentation capabilities while generating concrete enactable workflows that can be used in automated SE guidance support. Based on the information gleaned from the SE model, workflow concepts are transformed into an intermediate workflows language SEWL, from which further workflow transformations to specific WfMS can occur. An evaluation utilizes the EPF Composer with both existing and new SE process models and with Activiti and jBPM WfMS, the results showing the viability and practicality of the method for documenting an SE model with existing tooling, extracting SE concepts, graphically modeling SE workflows, transforming SE workflows to specific WfMS formats, and enacting SE workflows in support of contextual guidance capabilities for software engineers.

The summary of the paper is as follows: the following section discusses related work, and Section III describes the solution method. Section IV then describes our realization of the method. Section V presents an evaluation, followed by a discussion and then the conclusion.

II. Related Work

With regard to related work, SPEM 2.0 [21] was approved without supporting full process enactment. It proposes two possible approaches for enactment: One proposes a mapping to project planning tools. However, this does not support automated adaptation to changing project contexts during project execution. The other proposal is to use the Process Behavior package to relate SPEM process elements to external behavior models using proxy classes. Both approaches lack full workflow modeling and executability at the level of BPMN.

Other work related to enactment of SPEM includes eXecutable SPEM (xSPEM) [21]. Process execution is addressed via transformation to the Business Process Execution Language (BPEL), while process validation is addressed via transformation to a Petri net in combination with a model checker. [22] maps SPEM to the Unified Modeling Language Extended Workflow Metamodel (UML-EWM) in order to create a concretely executable workflow. [23] and [24] investigate transforming SPEM to BPMN, while [25] maps SPEM to the XML Process Definition Language (XPDL). xSPIDER ML [26] is an extension profile of SPEM 2.0 to enable process enactment.

The novelty of our solution method is that, in contrast to the above approaches, it targets a simple graphical as well as textual SE process language and notation for modeling, blending the strengths of BPMN and SPEM; it concretely generates executable workflows on different WfMS targets; and it generates an Web Ontology Language (OWL)-compliant ontology of SE concepts for context-aware PCSEE tooling support. This addresses prior hindrances and challenges for modeling and contextually integrating SE workflows in SEE. Furthermore, the model-driven integration of SE process meta-models provides a way to support comprehensive SE process documentation while providing an automated method to extract enactable SE workflows and contextual concepts.

III. Solution

This section describes our model-driven solution method (refer to Figure 1). The description below will refer to the four phases in the method shown at the top of Figure 1, namely *model*, *transform*, *deploy*, and *operate*.

The basis of the solution concept is an SE workflow model, such as SEWL workflows which we had previously developed. While the method supports the use of any SE workflow format, SEWL was used as an intermediate workflow model in our realization of this method. A SEWL workflow is modeled, either with the graphical SEWL editor or a textual editor, and provided as input for our Generator. To transform the input, our Generator utilizes various adapters we created that generate appropriate workflow templates tailored for a specific WfMS, while concurrently providing OWL-DL [27] output of the semantic concept instances. These templates are then deployed. During operations, a Process Manager Service we created abstracts, via an interface, the WfMS-specific integration and interaction details for our CoSEEEK (thus CoSEEEK does

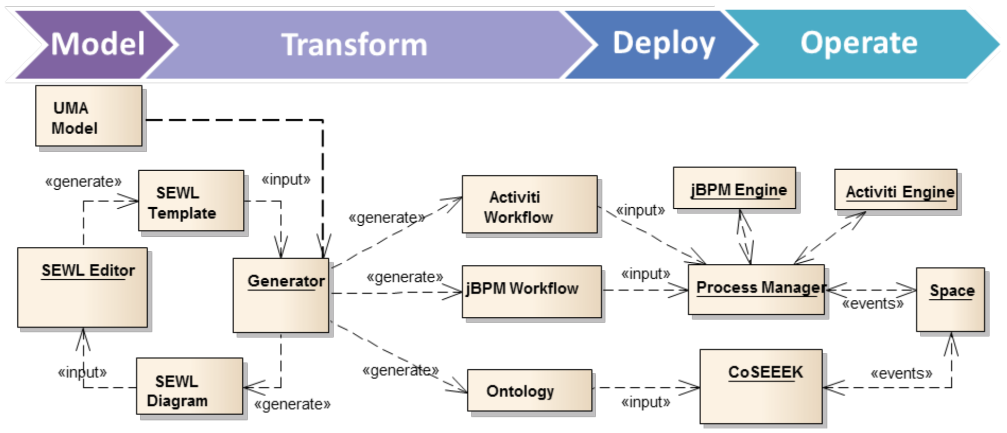


Figure 1. Solution method for SE process model transformation to executable and contextually-aware workflows.

not need to be a PAIS but only extend one) and the ontology is referenced internally during operations by CoSEEEK. Ontologies and semantic technology are advantageous in providing a taxonomy for modeled entities and their relations, a vocabulary, and supporting logical statements about entities [28]. Automated consistency checking and interoperability between different applications and agents also support SE environment concept reuse.

A. *The Model Phase*

In the *model* phase (see Figure 1), an SE process is modeled and documented based on an SE meta-model, such as a *UMA model* created using the EPF Composer. This model serves as an input to our Generator during the *transform* phase, which automatically generates a SE workflow model that maps all the correlating SE concepts to an intermediate SE workflow format such as SEWL. Alternatively, one could model or adapt the workflows directly in SEWL. In the process modeling phase, a graphical *SEWL Editor* assists the process modeler in creating the textual SEWL workflows, which maintain the essence of workflow concepts. Supplemental graphical diagram information (position, font, color, etc.) is retained in separately maintained diagram files, which are kept in sync with the SEWL workflows. Direct editing of the XML-based SEWL format is also possible; however, the Generator will remove all non-applicable elements from the graphical *SEWL diagrams* since the *SEWL template* XML file is considered the primary model source. Further details are provided in the next section.

B. *The Transform Phase*

As shown in Figure 1, in the *transform* phase SE workflow model inputs in a format such as SEWL are transformed by a Generator with a plug-in transformation adapter architecture to the executable workflow template format of a given WfMS target. An OWL adapter in the Generator also semantically transforms SE concepts in the workflow to produce an OWL-DL compliant ontology that it utilized for process contextual awareness by CoSEEEK.

To exemplify what the *transform* phase of our method does, Table I shows the mapping of common SE workflow concepts. Here WU stands for Work Unit and WUC for Work Unit Container. The primary difference between jBPM and Activiti concept mapping is that in Activiti loops are typically expressed via inclusive gateways and in jBPM via exclusive gateways. E.g., any concurrent tasks in an SE workflow would be modeled with the BPMN parallelGateway, which activates all branches simultaneously and, when merging, waits for all branches to complete. Most WfMS support such basic features.

TABLE I. MAPPING OF SE AND WORKFLOW CONCEPTS

SEWL	**Activiti**	**jBPM**	**Ontology**
Phase	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Activity	Service Task	Service Task	WUC + WU
Iteration	Service Task + inclusiveGateway	Service Task + exclusiveGateway	WUC + WU
Task	Service Task	Service Task	WU
Sequence	-	-	-
Parallel	parallelGateway	parallelGateway	-
Loop	inclusiveGateway	exclusiveGateway	-
XOR	exclusiveGateway	exclusiveGateway	-
Roles	-	-	Role Template
Artifacts	-	-	Artifact Template
Variables	-	-	Workflow Variables Template

To address and abstract the integration, communication, and coordination details of the specific WfMS, each Activity or Task is represented as a Service Task and, during generation, wrapped with code that supports the tracking or triggering of the start and finish of an activity or task via event sources and event listeners. This is done since a Process Manager Service abstracts the integration specifics of a WfMS for CoSEEEK, and a Space (a tuple space [29]) we developed is used to handle loosely-coupled communication during operation with CoSEEEK. Details on this are provided later in this and the next section.

C. *The Deploy Phase*

In the process deployment phase shown in Figure 1, workflows in a WfMS-specific format are deployed into their respective WfMS engine (e.g., jBPM or Activiti) and the workflow ontology deployed into CoSEEEK. Typically this implies transferring the workflow files to the expected locations for a given configuration.

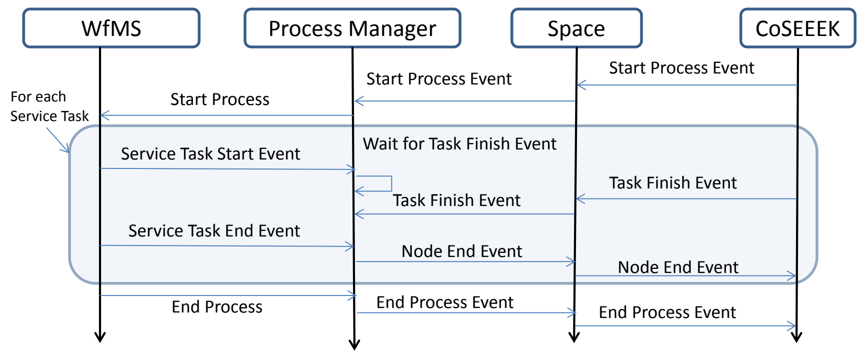


Figure 2. Primary runtime component interaction.

D. *The Operate Phase*

In the *operate* phase, a specific WfMS instantiates and executes workflow instances as prescribed by a Process Manager Service, which integrates a WfMS and abstracts its details and peculiarities. In our method, to support heterogeneous WfMS a Process Manager acts as an intermediary to support indirect and loosely-coupled event-based interaction between a Service Task and the context-aware PCSEE. In our method implementation, CoSEEEK uses our Space for this. Note that the *transform* phase needs to incorporate the expected operational semantics in order to generate the appropriate workflows for the operational phase.

Figure 2 provides an example of the operational interactions in our implementation of the method. The Process Manager has registered as a listener for certain events. CoSEEEK writes a Start Process Event into the Space. The Space notifies the Process Manager of this event, which in turn instantiates and starts a given process in the WfMS. For each Service Task in the workflow, a Service Task Start Event is sent to the Process Manager and the task waits until further notice. When CoSEEEK becomes aware of a context state change via tool sensors (e.g., a commit of source code was done, a test was started, or the software engineer manually chose a new activity) that affects this workflow and indicates that the current task is completed, CoSEEEK writes a Task Finish Event to the Space. The Space notifies the registered Process Manager of this event, and it in turn notifies the WfMS. The Service Task sends an End Event when it completes, and the Process Manager write a Node End Event in the Space. The Space notifies CoSEEEK of the event, which updates its state. When the final Service Task completes, the workflow completes, and the Process Manager writes an End Process Event to the Space, which notifies CoSEEEK, which in turn updates its state. As an aside, because all event history is kept in the Space, any CoSEEEK components or a Process Manager coming online after some absence (e.g., restart) can determine the context or catch up on any missed events.

IV. REALIZATION

This section provides details on the implementation of our method, the current contribution being primarily the integration of UMA support. To support loose-coupling with CoSEEEK, a service-oriented event-driven architecture was used in conjunction with a tuple space [29] composed on top of a native XML database eXist [30], and provides a Web Service for remote access. A Process Manager Service manages and abstracts the peculiarities of a WfMS, interacting indirectly with CoSEEEK via events in the Space.

The Eclipse Graphical Modeling Framework (GMF) [31] and Eclipse Modeling Framework (EMF) [32], which includes ecore, were utilized by the SEWL editor. Figure 3 shows a simplified snippet of the ecore-based metamodel.

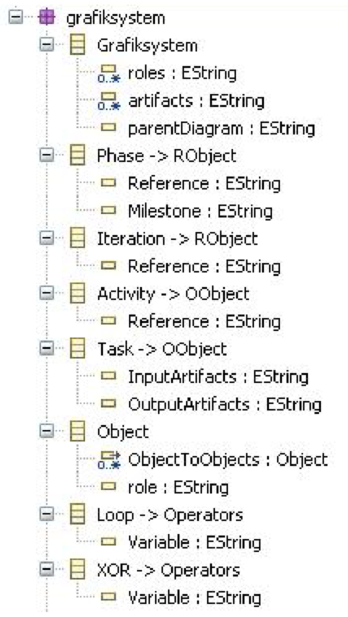


Figure 3. Simplified portion of the metamodel used with ecore.

Transformation Adapters. The Generator and associated pluggable transformation adapters (SEWL, UMA, OWL, jBPM, Activiti, AristaFlow, YAWL) were realized primarily in Scala. Unique IDs were generated for every element transformed and its target transformed element. This permits a clear mapping association, which is also useful for logging. The ontology adapter uses the Jena framework for programmatic ontology access [33] to generate the ontology instances for phases, activities, roles, artifacts, etc.

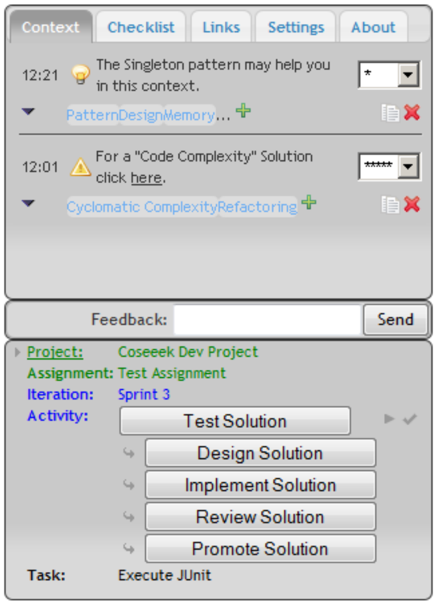
Figure 4 shows an example workflow snippet generated for a jBPM Service Task, while Figure 5 shows one for Activiti. For details on XML grammar of inputs or outputs, refer to the respective WfMS or UMA documentation.

```
<task id='2'name='RequestChange'tns:taskName='SEWL Task'>
 <extensionElements>
 <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>StartEventListener listener = new
StartEventListener();
     kcontext=listener.writeNodeStart(kcontext);</script>
 </tns:onEntry-script>
<tns:onExit-script
scriptFormat='http://www.java.com/java'>
  <script>EndEventListener listener = new
EndEventListener();
           listener.writeNodeEnd(kcontext);</script>
  </tns:onExit-script>
 </extensionElements>
</task>
```

Figure 4. Listing snippet of generated jBPM Service Task.

```
<serviceTask id='RequestChange' name='Request Change'
□activity:class='Service'>
 <extensionElements>
  <activity:executionListener event='start'
class='StartEventListener'/>
  <activity:executionListener event='end'
class='EndEventListener'/>
 </extensionElements>
</serviceTask>
```

Figure 5. Listing snippet of generated Activiti Service Task.



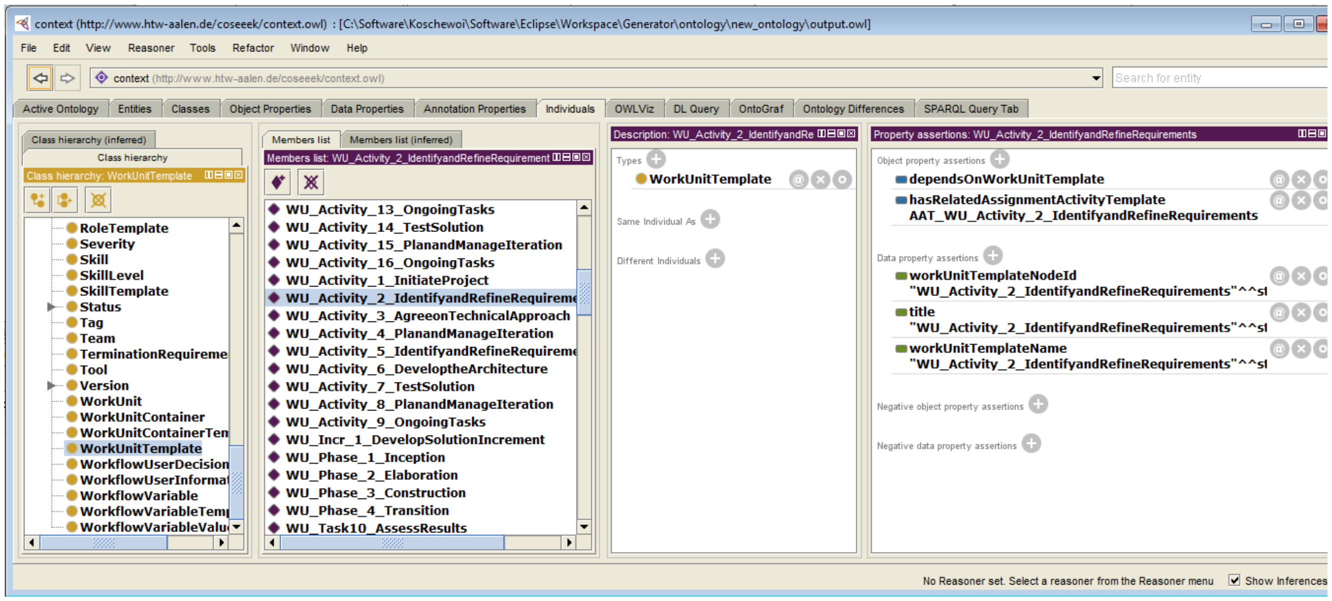
12:21 The Singleton pattern may help you in this context.
PatternDesignMemory...
12:01 For a "Code Complexity" Solution click here.
Cyclomatic ComplexityRefactoring
Feedback:

Project: Coseeek Dev Project
Assignment: Test Assignment
Iteration: Sprint 3
Activity: Test Solution
Design Solution
Implement Solution
Review Solution
Promote Solution
Task: Execute JUnit

Figure 6. CoSEEEK guidance GUI.

Generated OWL output was loaded into the Protégé ontology editor [34] and is shown for a work unit activity in Figure 7. Because the entire XML is very verbose, it is not shown. Figure 8 shows a small portion of the CoSEEEK software engineering environment ontology in graphical form to give an impression of how software engineering environment concepts, properties, and relations, such as work units and activities are tied into the larger project and environment, which is then utilized to provide contextual awareness.

These workflows and the associated ontology concepts serve as input to CoSEEEK, which then can provide contextual guidance for a software engineer during SE process execution, as can be seen in the screenshot of the HTML- and JavaScript-based CoSEEEK GUI (Graphical User Interface) shown in Figure 6. Context notifications are shown in the upper region, and process context guidance is in the bottom region, showing the current workflow activity (here 'Test Solution') and the next possible follow-on activity choices for guidance and/or manual selection by the user [35]. Details on CoSEEEK's holistic approach to support contextual guidance [12] and collaboration [13] during the SE process are published in other papers and beyond this paper's scope. For example, [14][36][37][38] provide details on the automated integration of software quality measures into executing SE workflows.


context (http://www.htw-aalen.de/coseeek/context.owl) : [C:\Software\Koschewoi\Software\Eclipse\Workspace\Generator\ontology\new_ontology\output.owl]

context (http://www.htw-aalen.de/coseeek/context.owl)

Class hierarchy: WorkUnitTemplate

RoleTemplate
Severity
Skill
SkillLevel
SkillTemplate
Status
Tag
Team
TerminationRequireme
Tool
Version
WorkUnit
WorkUnitContainer
WorkUnitContainerTem
WorkUnitTemplate
WorkflowUserDecision
WorkflowUserInformat
WorkflowVariable
WorkflowVariableTemp
WorkflowVariableValu

Members list: WU_Activity_2_IdentifyandRefineRequirement

WU_Activity_13_OngoingTasks
WU_Activity_14_TestSolution
WU_Activity_15_PlanandManageIteration
WU_Activity_16_OngoingTasks
WU_Activity_1_InitiateProject
WU_Activity_2_IdentifyandRefineRequireme
WU_Activity_3_AgreeonTechnicalApproach
WU_Activity_4_PlanandManageIteration
WU_Activity_5_IdentifyandRefineRequireme
WU_Activity_6_DeveloptheArchitecture
WU_Activity_7_TestSolution
WU_Activity_8_PlanandManageIteration
WU_Activity_9_OngoingTasks
WU_Incr_1_DevelopSolutionIncrement
WU_Phase_1_Inception
WU_Phase_2_Elaboration
WU_Phase_3_Construction
WU_Phase_4_Transition
WU_Task10_AssessResults

Description: WU_Activity_2_IdentifyandRe
Types
WorkUnitTemplate
Same Individual As
Different Individuals

Property assertions: WU_Activity_2_IdentifyandRefineRequirements
Object property assertions
dependsOnWorkUnitTemplate
hasRelatedAssignmentActivityTemplate AAT_WU_Activity_2_IdentifyandRefineRequirements
Data property assertions
workUnitTemplateNodeId "WU_Activity_2_IdentifyandRefineRequirements"^^st
title "WU_Activity_2_IdentifyandRefineRequirements"^^st
workUnitTemplateName "WU_Activity_2_IdentifyandRefineRequirements"^^st
Negative object property assertions
Negative data property assertions

No Reasoner set. Select a reasoner from the Reasoner menu

Figure 7. Generated OWL ontology for CoSEEEK shown in Protégé.

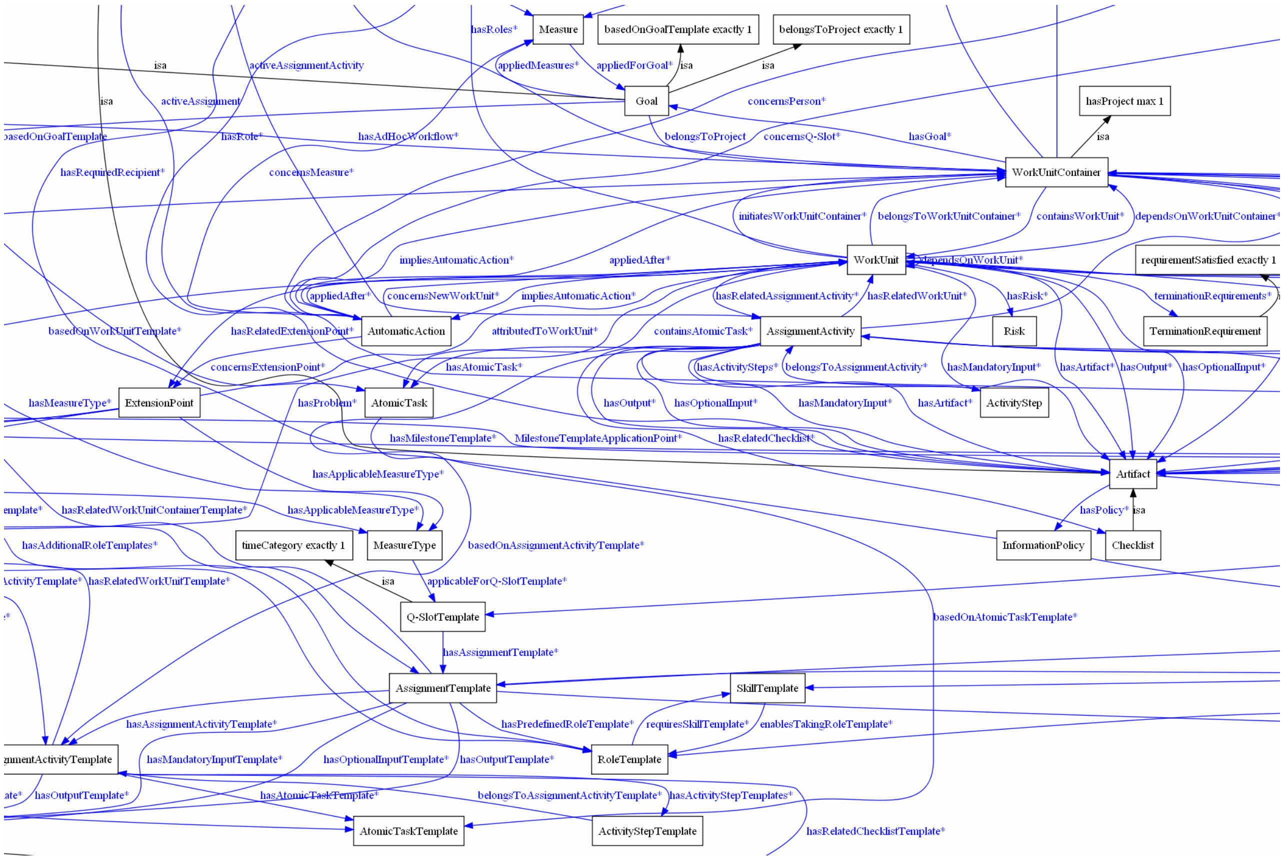
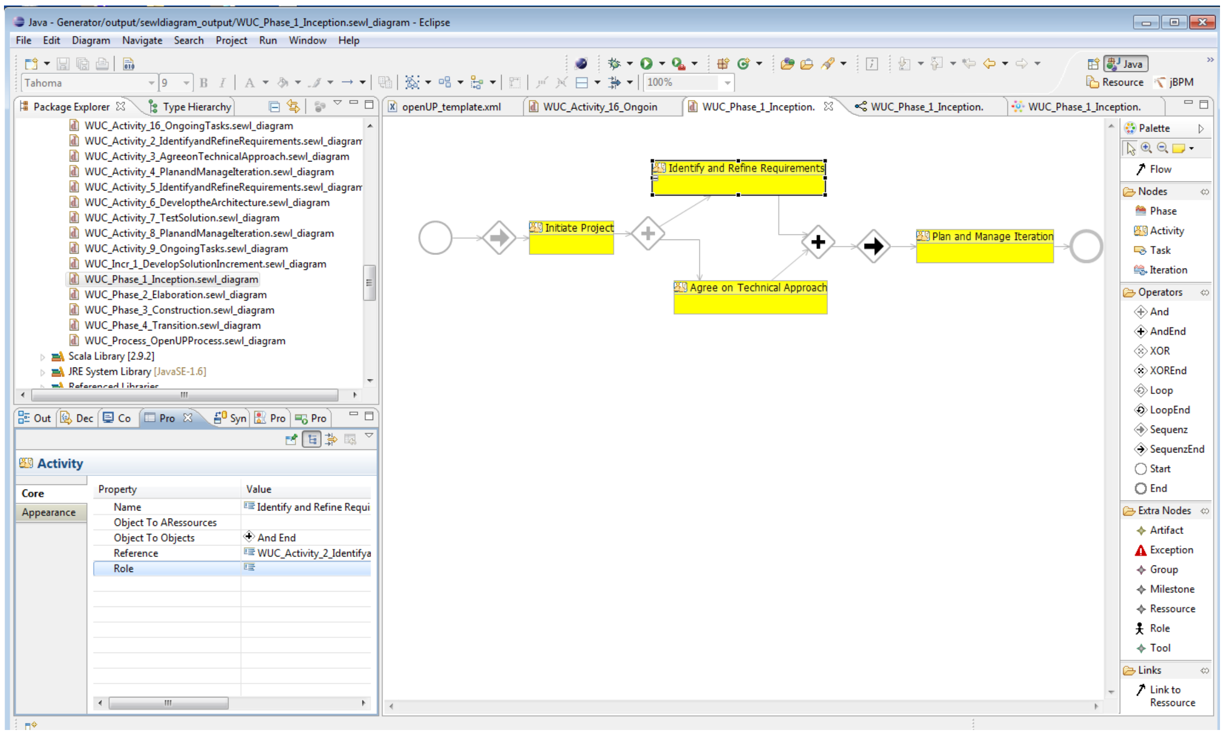


Figure 8. Screenshot of a portion of the CoSEEEK's software engineering environment ontology.



WUC_Activity_16_OngoingTasks.sewl_diagram
WUC_Activity_2_IdentifyandRefineRequirements.sewl_diagram
WUC_Activity_3_AgreeonTechnicalApproach.sewl_diagram
WUC_Activity_4_PlanandManageIteration.sewl_diagram
WUC_Activity_5_IdentifyandRefineRequirements.sewl_diagram
WUC_Activity_6_DeveloptheArchitecture.sewl_diagram
WUC_Activity_7_TestSolution.sewl_diagram
WUC_Activity_8_PlanandManageIteration.sewl_diagram
WUC_Activity_9_OngoingTasks.sewl_diagram
WUC_Incr_1_DevelopSolutionIncrement.sewl_diagram
WUC_Phase_1_Inception.sewl_diagram
WUC_Phase_2_Elaboration.sewl_diagram
WUC_Phase_3_Construction.sewl_diagram
WUC_Phase_4_Transition.sewl_diagram
WUC_Process_OpenUPProcess.sewl_diagram
Scala Library [2.9.2]
JRE System Library [JavaSE-1.6]
Referenced Libraries

Property	Value
Name	Identify and Refine Requi
Object To ARessources	
Object To Objects	And End
Reference	WUC_Activity_2_Identifya
Role	

Figure 9. SEWL Editor showing the OpenUP Inception Phase in the SEWL graphical format.

SEWL Editor. The SEWL textual language described in [16] supports the modeling of SE workflow concepts that a SE process may have. Multi-lingual support for referencing the same SE concept instance in various natural languages (e.g., German and English) was also implemented previously, supporting global software development (GSD) processes and their documentation in multiple languages.

The graphical notation used in the editor is extensible and can be adapted or "skinned" with icons to suit the preferences of the user, which can minimize notation confrontations between different user "tribes", e.g., BPMN purists or SPEM purists. In order to get the "best of both worlds", the SEWL Editor currently applied a mix of graphical notation as follows:

- SPEM icons for all SE concepts (e.g., phase, activity, iteration, task, role, artifact),
- BPMN icons for process notation, e.g., events, gateways, and connections.

As an example, an OpenUP Inception phase workflow modeled in the SEWL Editor is shown in its graphical (Figure 9) and textual (Figure 10) notation. One can see that various SE concepts such as roles, phases, artifacts, activities, inputs, and outputs can be modeled and sequenced.

```
<process base="default_process.xml" xmlns=...>
  <resources>
    <roles>
      <role id="1" name="Analyst" />
      <role id="2" name="Project Manager" />
...
  <elements>
    <element name="phase" base="container">
      <structure>
        <attribute name="repeatable">true</attribute>
      <rules>
        <contains element="activity" />
        <contains element="iteration" />
...
  <artifacts>
    <types/>
    <instances>
      <artifact type="Artifact">Project Plan</artifact>
...
  <tools/>
  <element type="sequence" name="OpenUP Process"
resource="6">
    <element type="phase" name="Inception"
milestone="Lifecycle Objectives">
      <element type="sequence">
        <element type="activity" name="Initiate Project">
          <element type="task" name="Develop Technical
Vision" resource="1">
            <output>
              <parameter name="vision"
tailoring="true">Vision</parameter>
              <parameter name="glossary"
tailoring="true">Glossary</parameter>
...
        <element type="parallel">
          <element type="activity" name="Identify and
Refine Requirements">
            <element type="sequence" resource="1">
...
                <element type="activity" name="Agree on
Technical Approach" resource="4">
...
      <element type="activity" name="Plan and Manage
Iteration" resource="2">
        <element type="sequence">
            <output>...
            </output>
```

Figure 10. Example OpenUP SEWL workflow snippets (end-tags omitted).

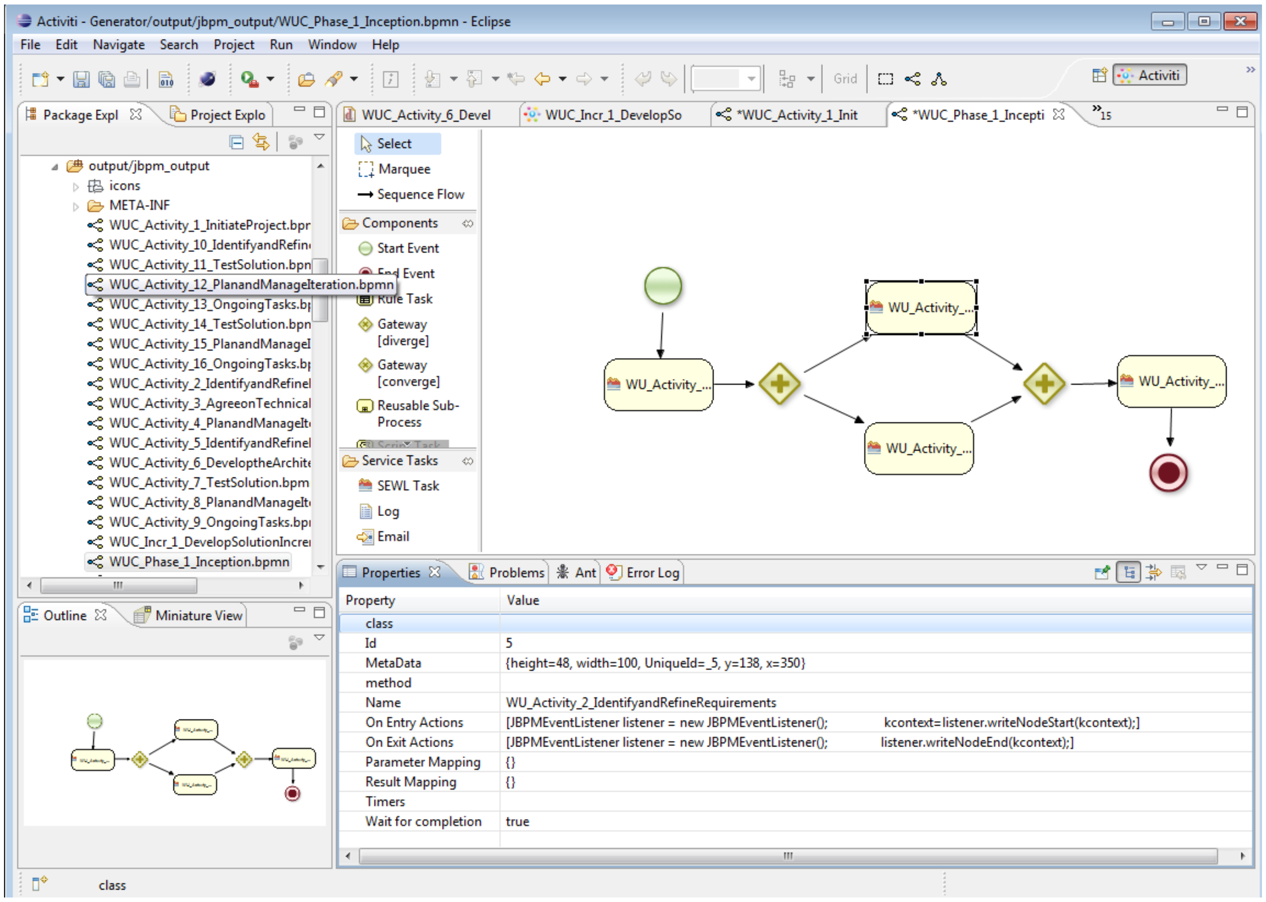
To retain the graphical details of the layout of nodes and edges, a separate file in XMI [39] notation was used. Figure 11 gives an example.

```
<graphicsystem:Graphicsystem
xmi:id='WUC_Phase_1_Inception'
parentDiagram='WUC_Process_OpenUPProcess.sewl_diagram' >
     <newObjects xmi:type='graphicsystem:Start'
xmi:id='startevent1' ObjectToObjects='sequenceStart1' />
  <newObjects xmi:type='graphicsystem:Sequenz'
xmi:id='sequenceStart1'
ObjectToObjects='WU_Activity_1_InitiateProject' />
   <newObjects xmi:type='graphicsystem:Activity'
xmi:id='WU_Activity_1_InitiateProject' Name='Initiate
Project'
Reference='WUC_Activity_1_InitiateProject.sewl_diagram'
ObjectToObjects='parallelGatewayStart1' />
...
 </graphicsystem:Graphicsystem>
 <notation:Diagram xmi:id='id_WUC_Phase_1_Inception'
type='SEWL' element='WUC_Phase_1_Inception'
name='Inception.sewl_diagram' measurementUnit='Pixel'>
    <children xmi:type='notation:Shape'
xmi:id='shape_startevent1' type='2043'
element='startevent1'>
...
   </children>
     <children xmi:type='notation:Node'
xmi:id='shape_WU_Activity_1_InitiateProject' type='2034'
element='WU_Activity_1_InitiateProject'>
     <children xmi:type='notation:DecorationNode'
xmi:id='4e841147-2f14-445a-b0b4-30e714be504e'
type='5039'/>
     <children xmi:type='notation:BasicCompartment'
xmi:id='0b62527e-b592-4e3d-a367-541f17843fb9'
type='7011'/>
     <styles xmi:type='notation:DescriptionStyle'
xmi:id='1b9fea72-5856-4be5-9203-1ef5cc58d000'/>
     <styles xmi:type='notation:FontStyle'
xmi:id='3051a516-b9f4-42c6-9698-8072fbe9a301'/>
     <styles xmi:type='notation:LineStyle'
xmi:id='7ea4d238-14fc-4068-a4ce-ed6bb08820af'/>
     <layoutConstraint xmi:type='notation:Bounds'
xmi:id='11135191-6e30-4c7a-a803-dfd437a058bc' x='440'
y='185' />
    </children>
...
 <styles xmi:type='notation:DiagramStyle'
xmi:id='_avAfkaznEeGl_a7M295XCw'/>
  <edges xmi:type='notation:Connector' xmi:id='flow23'
type='4020' source='shape_startevent1'
target='shape_sequenceStart1'>
   <styles xmi:type='notation:FontStyle'
xmi:id='8712763c-8e17-4285-948b-0b78f41f90af' />
     <element xsi:nil='true' />
     <bendpoints xmi:type='notation:RelativeBendpoints'
xmi:id='71805553-c9c1-46ff-8d13-56c6a3ab24fc'
points='[20, 0, -125, 10]$[130, -14, -15, -4]'/>
     <sourceAnchor xmi:type='notation:IdentityAnchor'
xmi:id='63f1b22c-d2fd-408e-9b8a-99044df18ce6' id='EAST'
/>
  <targetAnchor xmi:type='notation:IdentityAnchor'
xmi:id='0fd5db1f-daac-468a-a457-2dcf6bf1ee43'   />
   </edges>
```

Figure 11. Example SEWL diagram XMI code snippet.

An exemplary subset of the included constraints used to validate the model is listed here, i.e., audit rules. These were implemented in Java to allow usage outside of the GMF:

- verify phase/activity element has an output and a submodel,
- verify end element has no output,
- verify task does not target iteration/activity/phase,
- verify Loop has LoopEnd, Sequence has SequenceEnd, XOR has XOREnd, And has AndEnd.


Activiti - Generator/output/jbpm_output/WUC_Phase_1_Inception.bpmn - Eclipse
File Edit Navigate Search Project Run Window Help
Package Expl
Project Explo
output/jbpm_output
icons
META-INF
WUC_Activity_1_InitiateProject.bpr
WUC_Activity_10_IdentifyandRefin
WUC_Activity_11_TestSolution.bpn
WUC_Activity_12_PlanandManageIteration.bpmn
WUC_Activity_13_OngoingTasks.bj
WUC_Activity_14_TestSolution.bpn
WUC_Activity_15_PlanandManageI
WUC_Activity_16_OngoingTasks.bj
WUC_Activity_2_IdentifyandRefinel
WUC_Activity_3_AgreeonTechnical
WUC_Activity_4_PlanandManageIt
WUC_Activity_5_IdentifyandRefinel
WUC_Activity_6_DevelopttheArchite
WUC_Activity_7_TestSolution.bpm
WUC_Activity_8_PlanandManageIt
WUC_Activity_9_OngoingTasks.bpı
WUC_Incr_1_DevelopSolutionIncre
WUC_Phase_1_Inception.bpmn
WUC_Activity_6_Devel
WUC_Incr_1_DevelopSo
*WUC_Activity_1_Init
*WUC_Phase_1_Incepti
Select
Marquee
Sequence Flow
Components
Start Event
End Event
Rule Task
Gateway [diverge]
Gateway [converge]
Reusable Sub-Process
Service Tasks
SEWL Task
Log
Email
WU_Activity_...
WU_Activity_...
WU_Activity_...
WU_Activity_...
Outline
Miniature View
Properties
Problems
Ant
Error Log
Activiti
Grid
class

Property	Value
class	
Id	5
MetaData	{height=48, width=100, UniqueId=_5, y=138, x=350}
method	
Name	WU_Activity_2_IdentifyandRefineRequirements
On Entry Actions	[JBPMEventListener listener = new JBPMEventListener(); kcontext=listener.writeNodeStart(kcontext);]
On Exit Actions	[JBPMEventListener listener = new JBPMEventListener(); listener.writeNodeEnd(kcontext);]
Parameter Mapping	{}
Result Mapping	{}
Timers	
Wait for completion	true

Figure 12. jBPM generated output (rearranged by hand).

V. EVALUATION

The evaluation of the solution method focuses on its practicality and viability. Three usage scenarios were evaluated, namely: a) the ability to use the method without utilizing any UMA model (SEWL only), b) starting with a new customized UMA model that represents an organization's own tailored SE process, and c) using an existing publicized UMA model. Furthermore, the performance of the method realization should be evaluated to determine if a model-driven XML-centric approach is adequate.

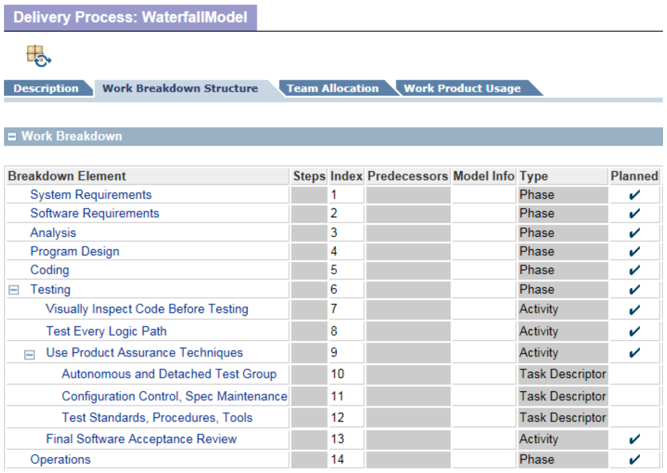
As to supporting a broad modeling spectrum, the Eclipse Process Framework (EPF) was used as a reference for modeling Scrum and OpenUP. These models were successfully modeled and transformed. Although the entire OpenUP process was modeled, only portions of the Inception Phase are shown below due to space constraints.

A. SEWL Non-UMA Process Model Example

The entire OpenUP process was modeled using the SEWL Editor as a starting point as was seen in Figure 9, and the generator was executed and jBPM and Activiti outputs were generated. Figure 12 was rearranged by hand. A snippet of the corresponding generated output is shown in Figure 14 for Activiti and in Figure 15 for jBPM. Thus processes that do not have or wish to use UMA can still utilize the method and SEWL for SE workflows.

B. New UMA Process Model Case Study

For this case, we started with a new EPF Composer project and modeled a portion of the Waterfall model based on Royce's original paper [40] as shown in Figure 13.



Breakdown Element	Steps	Index	Predecessors	Model Info	Type	Planned
System Requirements		1			Phase	✓
Software Requirements		2			Phase	✓
Analysis		3			Phase	✓
Program Design		4			Phase	✓
Coding		5			Phase	✓
Testing		6			Phase	✓
Visually Inspect Code Before Testing		7			Activity	✓
Test Every Logic Path		8			Activity	✓
Use Product Assurance Techniques		9			Activity	✓
Autonomous and Detached Test Group		10			Task Descriptor	
Configuration Control, Spec Maintenance		11			Task Descriptor	
Test Standards, Procedures, Tools		12			Task Descriptor	
Final Software Acceptance Review		13			Activity	✓
Operations		14			Phase	✓

Figure 13. Screenshot of a Waterfall process modeled in EPF.

```
<<process id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
    <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.Proces
sStartEndListener'></activiti:executionListener>
   <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.Proces
sStartEndListener'></activiti:executionListener>
 </extensionElements>
  <startEvent id='startevent1' name='Start'></startEvent>
  <endEvent   id='endevent1'  name='End'></endEvent>
    <serviceTask id='WU_Activity_1_InitiateProject'
name='Initiate Project'
activiti:class='coseeek.workflow.process.activiti.extensi
on.DummyService '>
    <extensionElements>
     <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
     <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
    </extensionElements>
    </serviceTask>
   <parallelGateway id='parallelGatewayFork1' />
      <serviceTask
id='WU_Activity_2_IdentifyandRefineRequirements'
name='Identify and Refine Requirements'
activiti:class='coseeek.workflow.process.activiti.extensi
on.DummyService'>
      <extensionElements>
       <activiti:executionListener event='start'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
       <activiti:executionListener event='end'
class='coseeek.workflow.process.activiti.extension.EventL
istener'></activiti:executionListener>
```

Figure 14. Example Activiti XML snippet.

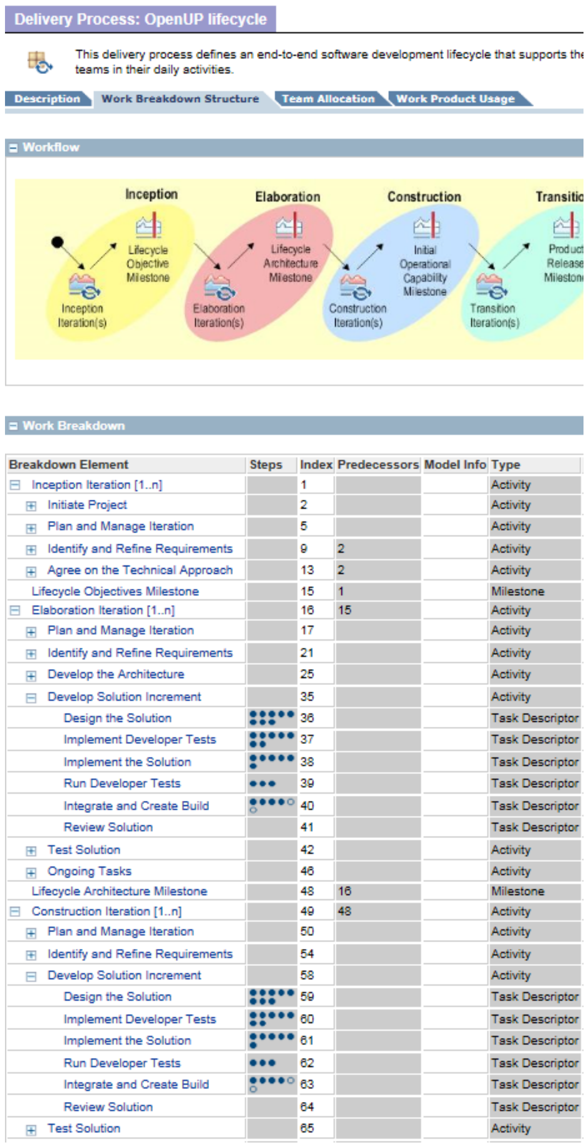
```
<process processType='Private' isExecutable='true'
id='WUC_Phase_1_Inception' name='Inception'>
 <extensionElements>
  <tns:import name='coseeek.workflow.process
.jbpm.extension.JBPMEventListener'/>
 </extensionElements>
  <startEvent id='_1' name='Start'></startEvent>
...
   <parallelGateway id='_3' gatewayDirection='Diverging'
/>
   <parallelGateway id='_4' gatewayDirection='Converging'
/>
  <task id='_5'
name='WU_Activity_2_IdentifyandRefineRequirements'
tns:taskName='SEWL Task' >
  <extensionElements>
  <tns:onEntry-script
scriptFormat='http://www.java.com/java'>
  <script>JBPMEventListener listener =
      new JBPMEventListener();
   kcontext=listener.writeNodeStart(kcontext);</script>
  </tns:onEntry-script>
  <tns:onExit-script
     scriptFormat='http://www.java.com/java'>
    <script>JBPMEventListener listener = new
JBPMEventListener();
        listener.writeNodeEnd(kcontext);</script>
    </tns:onExit-script>
  </extensionElements>
  <ioSpecification>
    <inputSet/>
    <outputSet/>
  </ioSpecification>
  </task>
<task id='_6' name='WU_Activity_3_AgreeonTechnicalApp...
```

Figure 15. Example jBPM workflow snippet.

This demonstrates that an organization's model can be used conveyed to UMA, and that as long as phases, activities, and/or tasks are modeled, default sequential workflows can be automatically generated from this in SEWL as shown in Figure 17. Here, the *phases* are shown as a sequential workflow. The Testing Phase is shown as a workflow of *activities* as shown in Figure 18. In Figure 19, *tasks* within the activity related to product assurance techniques specified by the Waterfall model are shown. If desired, workflows can then modified in the SEWL, e.g., for more complex non-sequential workflow models. The SEWL workflows were automatically transformed by the Generator into corresponding jBPM workflows (shown in Figures 20-22) and Activiti workflows (shown in Figures 23-25).

C. *Existing UMA Process Model Case Study*

The published UMA process model OpenUP, shown in Figure 16, was modified to simulate a customization scenario for an organization. Here a "Review Solution" task was added to the "Develop Solution Increment" activity.



This delivery process defines an end-to-end software development lifecycle that supports the teams in their daily activities.

Breakdown Element	Steps	Index	Predecessors	Model Info	Type
⊟ Inception Iteration [1..n]		1			Activity
⊞ Initiate Project		2			Activity
⊞ Plan and Manage Iteration		5			Activity
⊞ Identify and Refine Requirements		9	2		Activity
⊞ Agree on the Technical Approach		13	2		Activity
Lifecycle Objectives Milestone		15	1		Milestone
⊟ Elaboration Iteration [1..n]		16	15		Activity
⊞ Plan and Manage Iteration		17			Activity
⊞ Identify and Refine Requirements		21			Activity
⊞ Develop the Architecture		25			Activity
⊟ Develop Solution Increment		35			Activity
Design the Solution	●●●●●●●●	36			Task Descriptor
Implement Developer Tests	●●●●●●●●	37			Task Descriptor
Implement the Solution	●●●●●●	38			Task Descriptor
Run Developer Tests	●●●	39			Task Descriptor
Integrate and Create Build	●●●●○○	40			Task Descriptor
Review Solution		41			Task Descriptor
⊞ Test Solution		42			Activity
⊞ Ongoing Tasks		46			Activity
Lifecycle Architecture Milestone		48	16		Milestone
⊟ Construction Iteration [1..n]		49	48		Activity
⊞ Plan and Manage Iteration		50			Activity
⊞ Identify and Refine Requirements		54			Activity
⊟ Develop Solution Increment		58			Activity
Design the Solution	●●●●●●●●	59			Task Descriptor
Implement Developer Tests	●●●●●●●●	60			Task Descriptor
Implement the Solution	●●●●●●	61			Task Descriptor
Run Developer Tests	●●●	62			Task Descriptor
Integrate and Create Build	●●●●○○	63			Task Descriptor
Review Solution		64			Task Descriptor
⊞ Test Solution		65			Activity

Figure 16. OpenUP screenshot showing Review Solution customization.

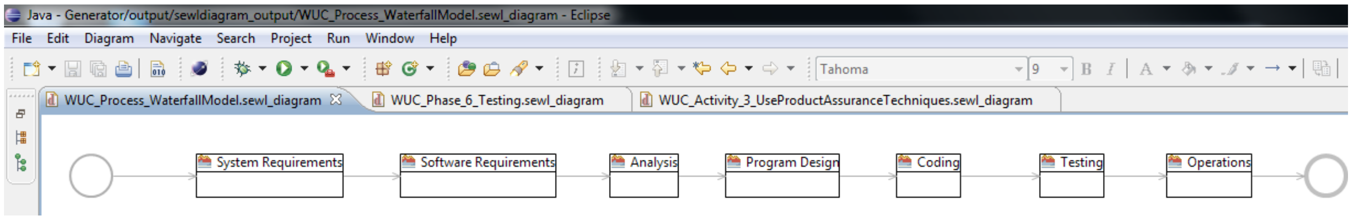

Java - Generator/output/sewldiagram_output/WUC_Process_WaterfallModel.sewl_diagram - Eclipse
File Edit Diagram Navigate Search Project Run Window Help
WUC_Process_WaterfallModel.sewl_diagram
WUC_Phase_6_Testing.sewl_diagram
WUC_Activity_3_UseProductAssuranceTechniques.sewl_diagram
System Requirements
Software Requirements
Analysis
Program Design
Coding
Testing
Operations

Figure 17. SEWL diagram of the Waterfall phases.

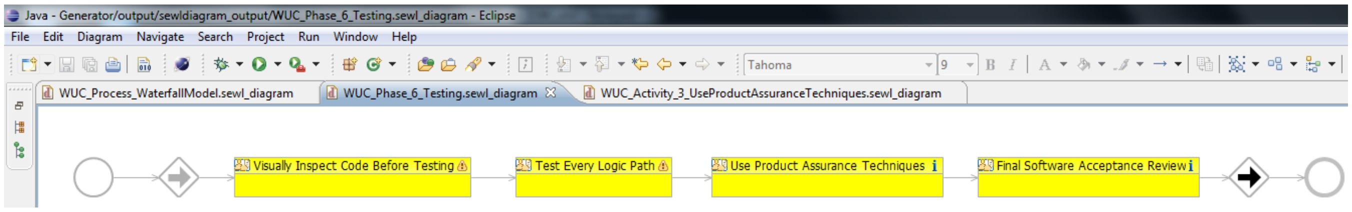

Java - Generator/output/sewldiagram_output/WUC_Phase_6_Testing.sewl_diagram - Eclipse
File Edit Diagram Navigate Search Project Run Window Help
WUC_Process_WaterfallModel.sewl_diagram
WUC_Phase_6_Testing.sewl_diagram
WUC_Activity_3_UseProductAssuranceTechniques.sewl_diagram
Visually Inspect Code Before Testing
Test Every Logic Path
Use Product Assurance Techniques
Final Software Acceptance Review

Figure 18. SEWL workflow diagram of Waterfall's Testing Phase.

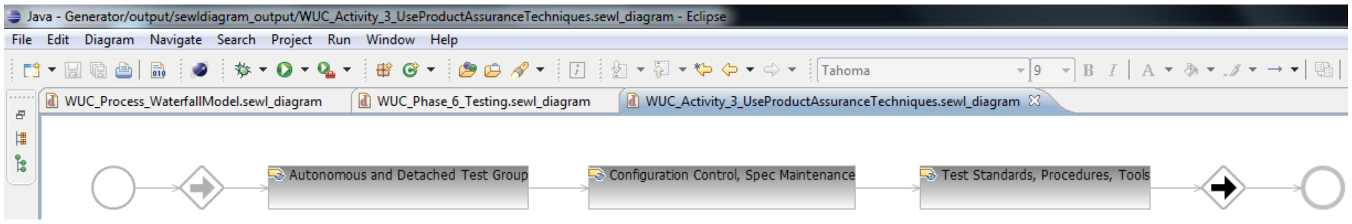

Java - Generator/output/sewldiagram_output/WUC_Activity_3_UseProductAssuranceTechniques.sewl_diagram - Eclipse
File Edit Diagram Navigate Search Project Run Window Help
WUC_Process_WaterfallModel.sewl_diagram
WUC_Phase_6_Testing.sewl_diagram
WUC_Activity_3_UseProductAssuranceTechniques.sewl_diagram
Autonomous and Detached Test Group
Configuration Control, Spec Maintenance
Test Standards, Procedures, Tools

Figure 19. SEWL workflow diagram showing tasks of the Waterfall's activity Use Product Assurance Techniques.

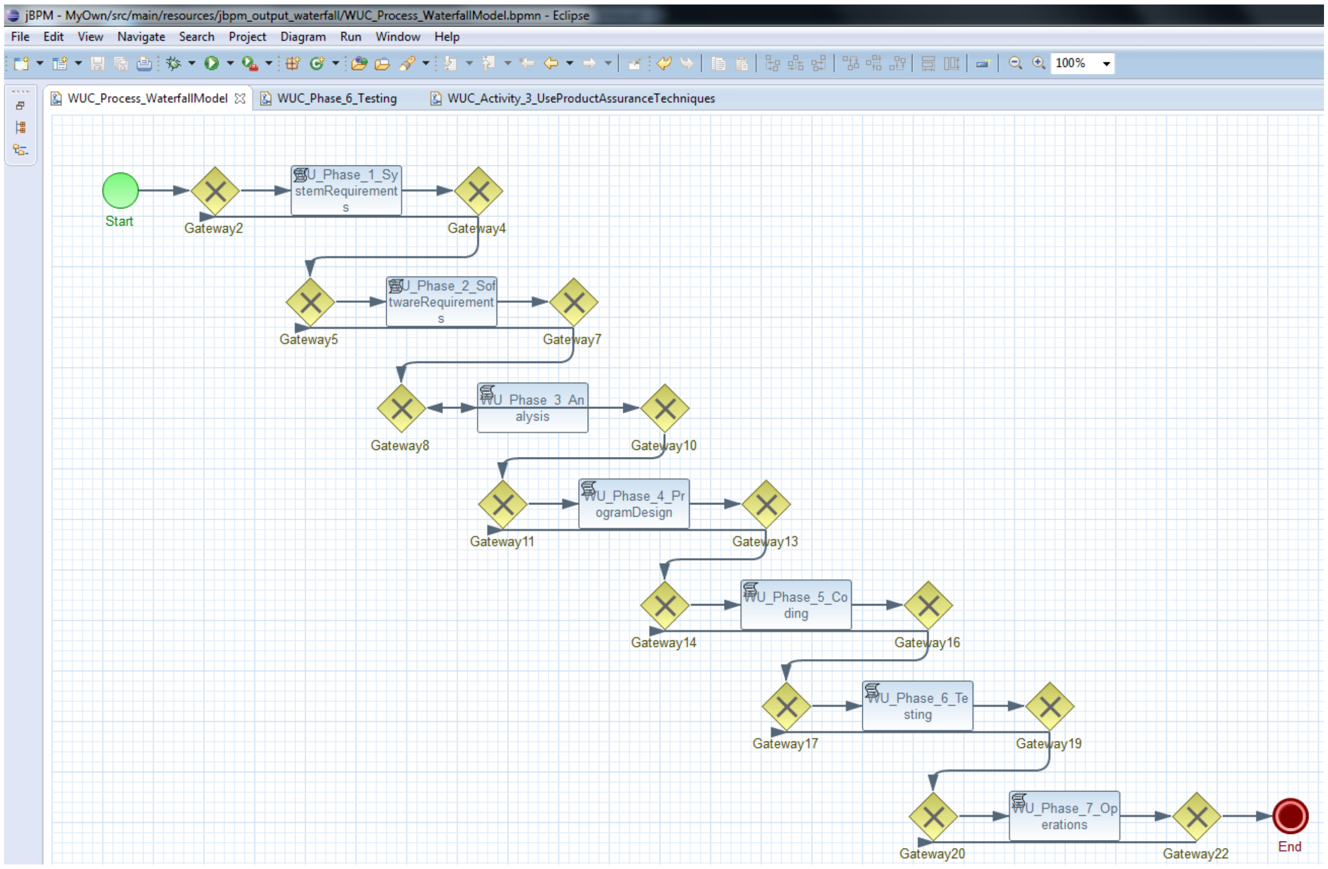

jBPM - MyOwn/src/main/resources/jbpm_output_waterfall/WUC_Process_WaterfallModel.bpmn - Eclipse
File Edit View Navigate Search Project Diagram Run Window Help
100%
WUC_Process_WaterfallModel
WUC_Phase_6_Testing
WUC_Activity_3_UseProductAssuranceTechniques
Start
Gateway2
U_Phase_1_SystemRequirements
Gateway4
Gateway5
U_Phase_2_SoftwareRequirements
Gateway7
Gateway8
U_Phase_3_Analysis
Gateway10
Gateway11
U_Phase_4_ProgramDesign
Gateway13
Gateway14
U_Phase_5_Coding
Gateway16
Gateway17
U_Phase_6_Testing
Gateway19
Gateway20
U_Phase_7_Operations
Gateway22
End

Figure 20. jBPM diagram of the Waterfall phases workflow.

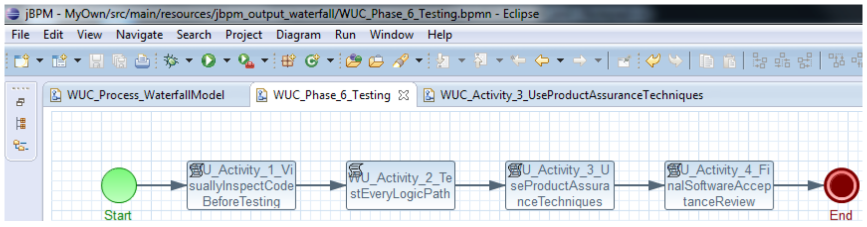

jBPM - MyOwn/src/main/resources/jbpm_output_waterfall/WUC_Phase_6_Testing.bpmn - Eclipse
File Edit View Navigate Search Project Diagram Run Window Help
WUC_Process_WaterfallModel
WUC_Phase_6_Testing
WUC_Activity_3_UseProductAssuranceTechniques
Start
U_Activity_1_VisuallyInspectCodeBeforeTesting
U_Activity_2_TestEveryLogicPath
U_Activity_3_UseProductAssuranceTechniques
U_Activity_4_FinalSoftwareAcceptanceReview
End

Figure 21. jBPM diagram of Waterfall's Testing Phase Workflow.

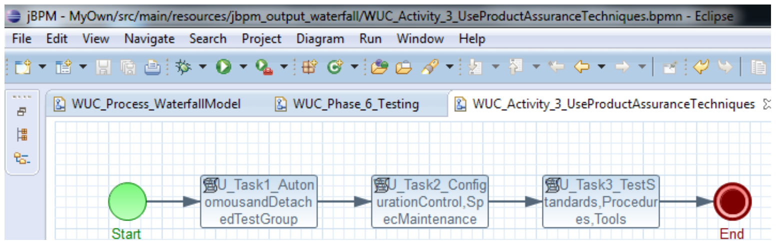

jBPM - MyOwn/src/main/resources/jbpm_output_waterfall/WUC_Activity_3_UseProductAssuranceTechniques.bpmn - Eclipse
File Edit View Navigate Search Project Diagram Run Window Help
WUC_Process_WaterfallModel
WUC_Phase_6_Testing
WUC_Activity_3_UseProductAssuranceTechniques
Start
U_Task1_AutonomousandDetachedTestGroup
U_Task2_ConfigurationControl,SpecMaintenance
U_Task3_TestStandards,Procedures,Tools
End

Figure 22. jBPM diagram of Waterfall's activity Use Product Assurance Techniques.

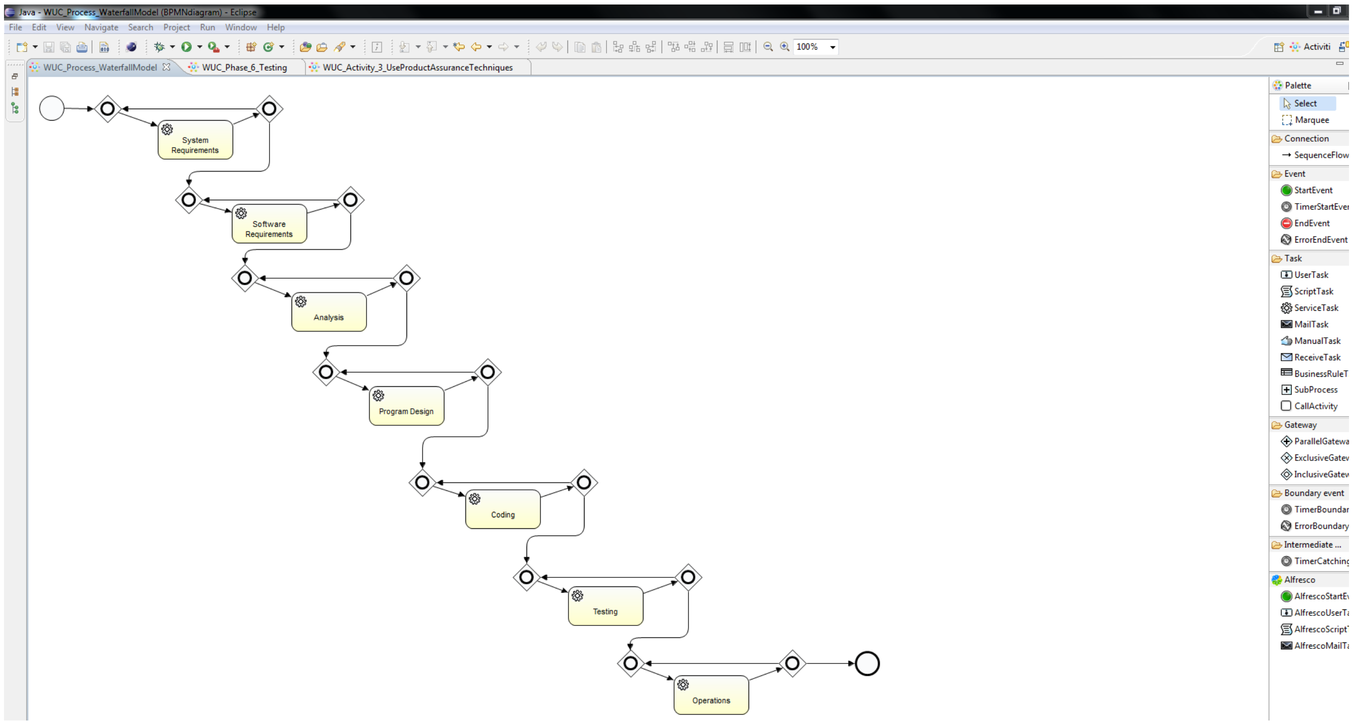


Figure 23. Activiti diagram of Waterfall's phases workflow.

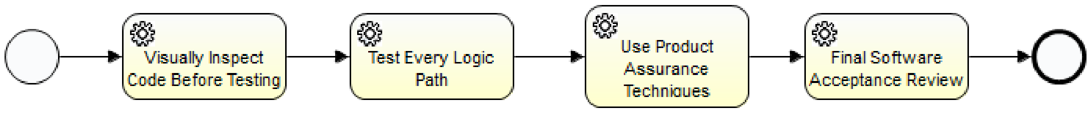

Visually Inspect Code Before Testing
Test Every Logic Path
Use Product Assurance Techniques
Final Software Acceptance Review

Figure 24. Activiti diagram of Waterfall's Test Phase workflow.

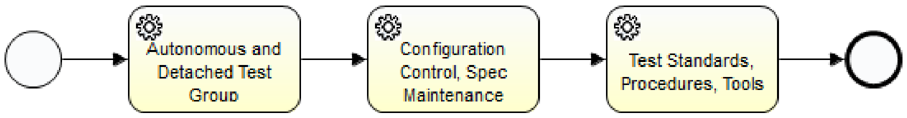

Autonomous and Detached Test Group
Configuration Control, Spec Maintenance
Test Standards, Procedures, Tools

Figure 25. Activiti diagram of the tasks in the Waterfall's activity Use Product Assurance Techniques.

This was done to demonstrate that an existing comprehensive UMA model from the community can be customized and default sequential workflows automatically generated in SEWL. In Figure 26, the phases are shown as a sequential workflow. The Construction Phase was modeled as a workflow of activities in Figure 27. In Figure 28, the tasks for the activity related to product assurance techniques are shown. If necessary, these workflows can then be modified in the SEWL graphical editor to suit the needs for more complex non-sequential workflow models. Nevertheless, a starting basis is automatically provided. From the SEWL model, we transformed these SEWL workflows using the Generator into corresponding jBPM (Figures 29-31) and Activiti (Figures 32-34) workflows.

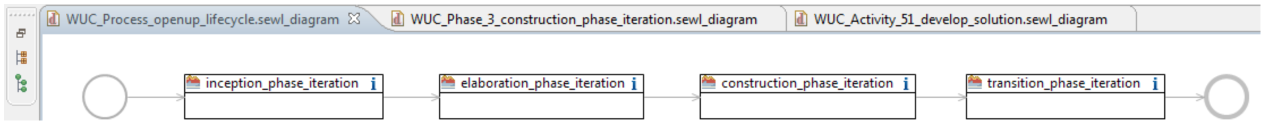

WUC_Process_openup_lifecycle.sewl_diagram
WUC_Phase_3_construction_phase_iteration.sewl_diagram
WUC_Activity_51_develop_solution.sewl_diagram
inception_phase_iteration
elaboration_phase_iteration
construction_phase_iteration
transition_phase_iteration

Figure 26. SEWL diagram of the OpenUP phases workflow.

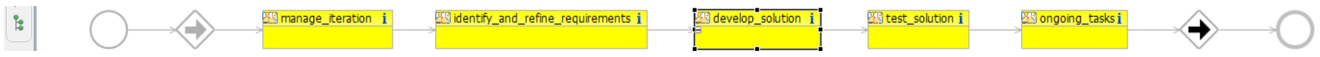


Figure 27. SEWL diagram of OpenUP's Construction Phase workflow.

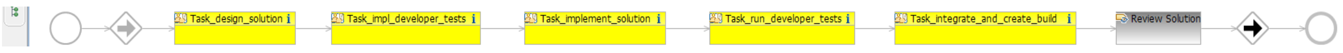


Figure 28. SEWL diagram of the OpenUP's Develop Solution Increment workflow.

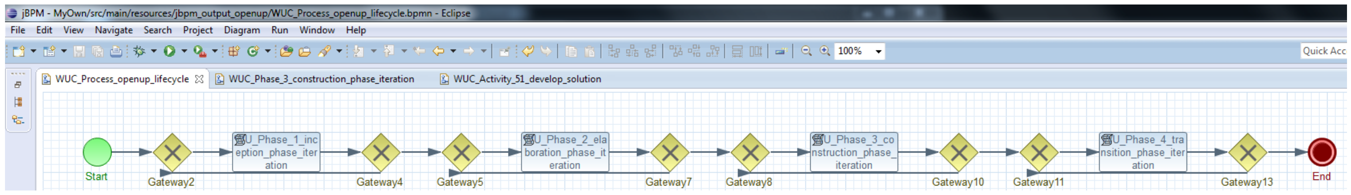

jBPM - MyOwn/src/main/resources/jbpm_output_openup/WUC_Process_openup_lifecycle.bpmn - Eclipse
File Edit View Navigate Search Project Diagram Run Window Help
WUC_Process_openup_lifecycle
WUC_Phase_3_construction_phase_iteration
WUC_Activity_51_develop_solution
Start
Gateway2
WU_Phase_1_inception_phase_iteration
Gateway4
Gateway5
WU_Phase_2_elaboration_phase_iteration
Gateway7
Gateway8
WU_Phase_3_construction_phase_iteration
Gateway10
Gateway11
WU_Phase_4_transition_phase_iteration
Gateway13
End

Figure 29. jBPM diagram of the OpenUP phases workflow.

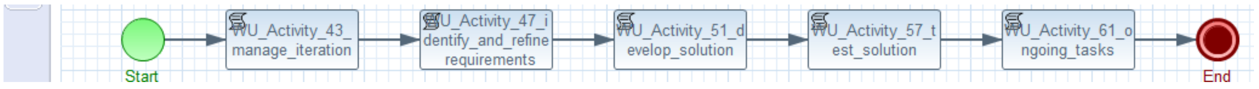

Start
WU_Activity_43_manage_iteration
WU_Activity_47_identify_and_refine requirements
WU_Activity_51_develop_solution
WU_Activity_57_test_solution
WU_Activity_61_ongoing_tasks
End

Figure 30. jBPM diagram of OpenUP's Construction Phase workflow.

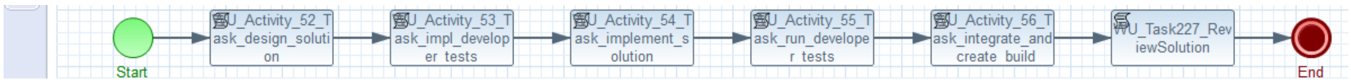

Start
WU_Activity_52_Task_design_solution
WU_Activity_53_Task_impl_developer tests
WU_Activity_54_Task_implement_solution
WU_Activity_55_Task_run_developer tests
WU_Activity_56_Task_integrate_and create build
WU_Task227_ReviewSolution
End

Figure 31. jBPM diagram of the OpenUP Develop Solution Increment workflow.

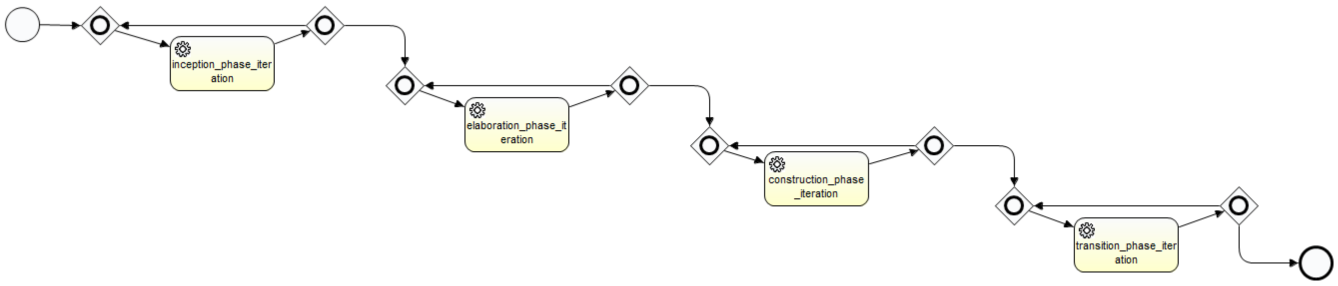

inception_phase_iteration
elaboration_phase_iteration
construction_phase_iteration
transition_phase_iteration

Figure 32. Activiti diagram of the OpenUP phases workflow.

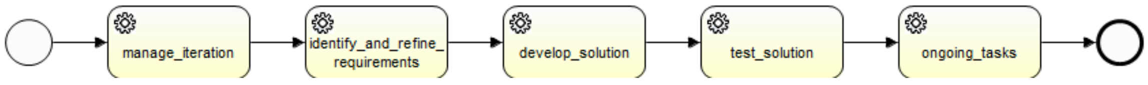


Figure 33. Activiti diagram of OpenUP's Construction Phase workflow.

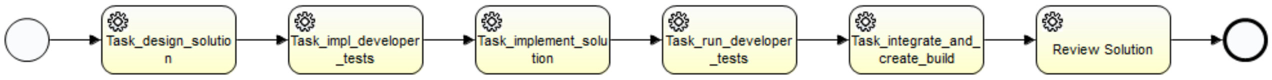


Figure 34. Activiti diagram of OpenUP's Develop Solution Increment workflow.

D. Performance

We evaluated our model-driven solution to determine if it exhibits acceptable transformation performance for expected usage.

1) Performance for provided SE Models: For this scenario we wanted to determine the performance one can expect for model transformation of basic and realistic SE process models that conform to the EPF XML Schema. The configuration for these measurements consisted of an Intel Core i7-3740QM CPU @2,70GHz, 16 GB RAM, Windows 7 Ultimate SP1 x64, Java 8, Scala 2.11.5.

The performance results presented in Table II differentiate the two UMA SE processes Waterfall and OpenUP, with Waterfall being a new relatively small model example, and OpenUP being a comprehensive model example. The second major column shows the resulting file sizes in bytes, lines, and number of files involved for each of the generation steps. The major center column then shows the duration in milliseconds for various transformations. The EPF input files are the starting point, with the generation steps being the EPF to SEWL transformation (EPF->SEWL), the SEWL to diagram transformation (SEWL->diagram), the SEWL to jBPM transformation (SEWL->jBPM), and the SEWL to Activiti transformation (SEWL->Activiti). The maximum time needed was about 1 second needed to create 83 XMI diagram files with over 10,0000 XML lines from the input of 995 lines of SEWL XML. This performance seems acceptable for such a comprehensive SE model.

TABLE II. EPF MODEL TRANSFORMATION PERFORMANCE

Generation Steps	**Duration (milliseconds)**		**Generated XML in bytes [lines] (files)**	
	Waterfall	***OpenUP***	***Waterfall***	***OpenUP***
EPF Input files	-	-	9,480 [75] (1)	1,671,816 [24109] (1)
EPF -> SEWL	341	505	3,084 [85] (1)	51,695 [995] (1)
SEWL -> diagram	811	1040	32,344 [332] (3)	1,008,362 [10277] (83)
SEWL -> jBPM	646	833	20,929 [442] (12)	222,632 [5438] (83)
SEWL -> Activiti	609	950	17,218 [183] (3)	383,087 [3747] (83)

2) Plugin Performance: We compared performance of the various generator plugins for their different types of output to determine if there are significant differences or issues with such a plugin concept. For this, a five node OpenUP process sequence was provided as the input to the SEWL Editor, and the performance of each of the adapters in the Generator measured. For each round, a loop of 1000 generations was averaged. For generating the SEWL template, the SEWL diagram files served as input. Otherwise the SEWL XML template alone was used as input. The configuration for these measurements consisted of an Intel Core 2 Duo CPU 2.26 GHz, 3 GB RAM, Windows XP Pro SP3, Java JDK 1.6.0-31, Scala 2.9.1, Activiti 5.8, jBPM 5.2, Jena 2.6.4, and Eclipse EMT (Helios) SR2. The performance results are presented in Table III, with the left column indicating the adapter measured, the center column the average duration, and the right column the size of the inputs and outputs in bytes, lines, and files.

TABLE III. GENERATOR ADAPTER PERFORMANCE

	Average Duration (millisec)	**XML File Size in bytes [lines] (files)**	
		Input	***Generated***
SEWL template	10.3	212,490 [2255] (22)	19,431 [416] (1)
SEWL-Diagram	65.1	19,431 [416] (1)	212,490 [2255] (22)
Activiti	23.8	19,431 [416] (1)	79,088 [822] (22)
jBPM	27.5	19,431 [416] (1)	86,169 [1856] (22)
Ontology	6917.8	19,431 [416] (1) 823,020 [12965] (1)	1,469,639 [15750] (1)

Generating a SEWL template from the diagram involves the least amount of writing, and is thus fastest. The generation of SEWL diagrams in XMI format is more verbose in bytes and lines by at least a factor of 2, and its duration is correspondingly longer compared to the jBPM or Activiti adapters. With regard to the Ontology adapter, two files serve as input for generating the OWL ontology; in addition to the SEWL template input, the Jena Semantic Web framework is used to parse and create internal objects from the existing ontology (a comparatively large file with its additional 12 related remote namespace schema), then the relevant ontology instances are updated based on the SEWL file, and finally a complete OWL file that contains the modifications is generated. Since much more XML is involved in both the input parsing and generation, and the use of specialized semantic OWL APIs, here the overall performance for generating semantic workflow context concepts is noticeable.

Because the RDF and OWL-DL XML formats are standardized, possible optimization strategies include partitioning the ontology to only those areas applicable for workflow ontology concepts, rather than the total ontology. Another possibility is the use of solid state disks on the devices involved in the ontology generation, e.g., by placing the adapter behind a web service.

In summary, the performance of the generators appears satisfactory for typical SE process transformation.

VI. Discussion

While there is potential for further automation of SE processes and automated guidance support, a number of practical hindrances remain.

In the past, since SE process documentation lacked contextually adapted and WfMS supported workflows, it has often seemed not to be operationally relevant, but rather something relatively abstract. It might serve to satisfy the appearance of the existence of a disciplined and professional method for approaching the software engineering work, with no real way to monitor actual usage or compliance with it. Thus, for most of the actors involved in using the documentation of SE processes, the documented workflows appear not to add significant value and most of the work is done without referring to the SE process documentation with any of its specified abstract workflows. Perhaps the swing in prior decades to overly documented und irrelevant SE processes caused an understandable and reactionary agile movement, as codified in the Agile Manifesto [1], to minimize tools and documentation.

However, if SE process documentation could include operationally concrete and WfMS-enactable workflows that provide contextually relevant guidance, and these workflows involved the actual tools and artifacts used and were tied in to the SE process documentation as well, then it would bring "life" to the relatively "dead" SE process documentation, since contextually adapted workflows would be mostly relevant and helpful to the software engineers in their actual work context. They would no longer have the hurdle of manually finding their current context in the abstract and perhaps quite comprehensive SE process documentation, and then manually determining the workflow portions they are supposed to follow and keep jumping from their work context to their process documentation context.

Activities and processes are an inherent part of SE and will continue to be used to accomplish SE work, be they explicit and documented or implicit and undocumented. As automation and intelligence in SE environments increases, a middle-ground may be found between these two extremes, so that the benefits that other domains have reaped from WfMS support for human-centric and hybrid human and automated workflows can be better integrated and leveraged in the SE landscape. This will involve documentation and workflow modeling, and a model-driven approach can automate many of the aspects in order to minimize the effort involved for software engineers.

The SE environment, tooling, and process area has seen relatively little standardization for various reasons. Efforts to better integrate the heterogeneous SE tooling landscape in a vendor-neutral manner, such as the semantic services approach of Open Services for Lifecycle Collaboration (OSLC) project, seem to have fizzled at the moment for all practical purposes. As tools continue to change, keeping the data models and integrations updated seems to involve significant effort without significant incentives. Future approaches may move more tooling to the cloud, enabling better context-aware SE integration.

VII. Conclusion and Future Work

This article contributed a practical model-driven methodology that supports the usage and transformation of software engineering process documentation to workflows executable in modern workflow management systems. The method can utilize comprehensive SE process documentation meta-models and automatically extract incorporated SE concepts and workflow models to an intermediate SE workflow format such as the Software Engineering Workflow Language (SEWL). These workflows can optionally be edited in a common SE format that is aware of SE process and environment concepts and can then transformed into various enactable WfMS formats and ontological concepts for context-aware support.

Automated workflow guidance for SE projects remains a challenge, and current SE process meta-models have hitherto not integrated support for intricate and enactable workflow modeling capabilities. With our practical model-driven method we showed that, beginning with only a rudimentary process documentation of a set of SE concepts conformant to some SE process meta-model such as the UMA, actually enactable workflows can be automatically generated for various common WfMS. Our method enables the utilization of currently available SE process documentation tooling such as the EPF Composer, without needing to deal with separate manual process modeling techniques for a vendor-specific WfMS due to our model-driven adapters. Such generated sequential workflows extracted and transformed from some SE process documentation can provide a starting point for more intricate operational SE workflow modeling in, for instance, our SEWL editor, should certain workflows be more complex or require branches or loops. These can be readily adjusted either with the SEWL graphical editor or directly in the SEWL text-based model, and then WfMS-specific workflows can be automatically generated.

This solution method provides an easy to use graphical modeling capability for executable SE workflows that can execute on commonly available WfMS, while retaining SE semantic information in a separate OWL file for contextually aware PCSEEs. The evaluation results show that such a model-based method for transforming SE workflows to common WfMS is both feasible and practical.

Future work includes case studies with industry partners in live settings as well as more comprehensive utilization of the ontological concepts extractable from such UMA-based models with those of CoSEEEK. Also, bidirectional workflow transformation support between SEWL and an engine-specific workflow format would allow editing in the workflow editor of choice. This entails providing reverse transformation support for engine-specific workflow templates, enabling engine-specific usage of features and editing capabilities via workflow engine-specific editors. For instance, changes made to jBPM and Activiti workflows could be automatically reflected in a SEWL template.

ACKNOWLEDGMENT

The author thanks and acknowledges Vitali Koschewoi and Julian Donauer for their work on the implementation and diagrams, and Gregor Grambow for his assistance with CoSEEEK-related concepts, ontology, and adaptations.

REFERENCES

[1] R. Oberhauser, "An Approach for Modeling and Transforming Contextually-Aware Software Engineering Workflows," Proceedings of the Ninth International Conference on Software Engineering Advances (ICSEA 2014), published by IARIA, ISBN: 978-1-61208-367-4 (2014), pp. 117-122.

[2] S. Biffl, D. Winkler, R. Höhn, and H. Wetzel, "Software process improvement in Europe: potential of the new V-modell XT and research issues," Software Process: Improvement and Practice, 11(3), 2006, pp. 229-238.

[3] P. Kroll and B. MacIsaac, Agility and Discipline Made Easy: Practices from OpenUP and RUP. Pearson Education, 2006.

[4] http://v-modell.iabg.de/v-modell-xt-html-english/ 2015.05.30.

[5] http://epf.eclipse.org/wikis/openup/ 2015.05.30.

[6] http://www.eclipse.org/epf/ 2015.05.30.

[7] P. Haumer, "Eclipse process framework composer," Eclipse Foundation, 2007.

[8] Object Management Group, "Software & Systems Process Engineering Metamodel Specification (SPEM) Version 2.0," Object Management Group, 2008.

[9] V. Gruhn, "Process-Centered Software Engineering Environments: A Brief History and Future Challenges," Annals of Software Engineering, 14(1-4), 2002, pp. 363-382.

[10] A. Fuggetta, "Software process: a roadmap," Proc. Conf. on the Future of Software Eng., ACM, May 2000, pp. 25-34.

[11] M. Reichert and B. Weber, "Enabling flexibility in process-aware information systems: challenges, methods, technologies," Springer Science & Business Media, 2012.

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

[13] G. Grambow, R. Oberhauser, and M. Reichert, "Enabling Automatic Process-aware Collaboration Support in Software Engineering Projects," Software and Data Technologies (Editors: J. Cordeiro, M. Virvou, B. Shishkov), CCIS 303, Springer Verlag, ISBN 978-3-642-29577-5, 2012, pp. 73-88.

[14] G. Grambow, R. Oberhauser, and M. Reichert, "Contextually Injecting Quality Measures into Software Engineering Processes," the International Journal On Advances in Software, ISSN 1942-2628, vol. 4, no. 1 & 2, 2011, pp. 76-99.

[15] Object Management Group, "Business Process Model and Notation (BPMN) Version 2.0," 2011.

[16] G. Grambow, R. Oberhauser, and M. Reichert, "Towards a Workflow Language for Software Engineering," Proc. of the The Tenth IASTED Int'l Conf. on Software Engineering (SE 2011), ISBN 978-0-88986-880-9, ACTA Press, 2011.

[17] P. Dadam et al., "From ADEPT to AristaFlow BPM suite: a research vision has become reality," in Business process management workshops, Springer, Jan. 2010, pp. 529-531.

[18] W. Van Der Aalst and A. Ter Hofstede, "YAWL: yet another workflow language," Information systems, 30(4), 2005, pp. 245-275.

[19] M. Salatino and E. Aliverti, jBPM5 Developer Guide, ISBN 1849516448, Packt Publishing, 2012.

[20] T. Rademakers, "Activiti in Action: Executable business processes in BPMN 2.0," Manning Publications Co., 2012.

[21] R. Bendraou, B. Combemale, X. Crégut, and M. Gervais, "Definition of an Executable SPEM 2.0," In Proc. APSEC 2007, IEEE, 2007, pp. 390-397.

[22] N. Debnath, D. Riesco, M. Cota, J. Garcia Perez-Schofield, and D. Uva, "Supporting the SPEM with a UML Extended Workflow Metamodel," Proc. IEEE Conf. on Computer Systems and Applications, AICCSA, 2006, pp. 1151-1154.

[23] D. Riesco, G. Montejano, N. Debnath, and M. Cota, "Formalizing the Management Automation with Workflow of Software Development Process Based on the SPEM Activities View," Proc. 6th Int'l Conf. on information Technology: New Generations, 2009, pp. 131-136.

[24] M. Perez Cota, D. Riesco, I. Lee, N. Debnath, and G. Montejano, "Transformations from SPEM work sequences to BPMN sequence flows for the automation of software development process," J. Comp. Methods in Sci. and Eng. 10, 1-2S1, (September 2010), 2010, pp. 61-72.

[25] Y. Feng, L. Mingshu, and W. Zhigang, "SPEM2XPDL: Towards SPEM Model Enactment," Proc. of SERP, 2006, pp. 240-245.

[26] C. Portela et al. "xSPIDER ML: Proposal of a Software Processes Enactment Language Compliant with SPEM 2.0," J. of SW Eng. & Applications, 5(6), 2012, pp. 375-384.

[27] D. McGuinness and F. Van Harmelen, "OWL web ontology language overview," W3C recommendation, 2004.

[28] D. Gasevic, D. Djuric, and V. Devedzic, Model driven architecture and ontology development. Springer, 2006.

[29] D. Gelernter, "Generative communication in Linda," ACM Transactions on Programming Languages and Systems, 7(1), 1985, pp. 80-112.

[30] W. Meier, "eXist: An open source native XML database. Web," Web-Services, and Database Systems, LNCS, 2593, 2009, pp. 169-183.

[31] http://www.eclipse.org/modeling/gmp/ 2015.05.30.

[32] http://www.eclipse.org/modeling/emf/ 2015.05.30.

[33] B. McBride, "Jena: a semantic web toolkit," Internet Computing, Nov. 2002, pp. 55-59.

[34] N. F. Noy et al., "Creating semantic web contents with protege-2000," IEEE intelligent systems, 16(2), pp. 60-71, 2001.

[35] G. Grambow, R. Oberhauser, and M. Reichert, "User-centric Abstraction of Workflow Logic Applied to Software Engineering Processes," Proceedings of the 1st Int'l Workshop on Human-Centric Process-Aware Information Systems (HC-PAIS'12), 2012.

[36] G. Grambow and R. Oberhauser, "Towards Automated Context-Aware Selection of Software Quality Measures," Proc. 5th Intl. Conf. on Software Engineering Advances, 2010, pp. 347-352.

[37] G. Grambow, R. Oberhauser, and M. Reichert, "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, 2010, pp. 58-67.

[38] G. Grambow, R. Oberhauser, and M. Reichert, "Contextual Quality Measure Integration into Software Engineering Processes," International Journal on Advances in Software, 4(1&2), 2011, pp. 76-99.

[39] Object Management Group, "MOF 2 XMI Mapping Version 2.4," 2010.

[40] W. W. Royce, "Managing the development of large software systems," Proceedings of IEEE WESCON, Vol. 26, No. 8, 1970, pp. 328-339.

[41] K. Beck et al., "The agile manifesto," 2001.