JAVA Mag inkl. CD

Mule ESB 3.0 – Die neuen Features »19

Deutschland €8,50 Österreich €9,80 Schweiz sFr 16,80 11.2010

Java magazin

Java • Architekturen • SOA • Agile

www.javamagazin.de

CD-INHALT

42 für Softwarearchitekten

Video von der W-JAX 2009 in voller Länge

HIGHLIGHTS

Hazelcast 1.9

Activiti 5.0 Beta

WEITERE INHALTE

Apache libcloud

Apache Commons Lang

Apache MyFaces Orchestra Core

uvm.

Alle CD-Infos ab Seite 3

Apache Wicket

World Wide Wicket
Das Tutorial »38

Hazelcast

Open Source In-Memory Data Grid »80

W-JAX 10

Alle Infos im Heft ab Seite 51

Social Media API

Was Java-Entwickler wissen sollten »48

Apache Maven

Dependency-Magie »100

Java EE 6 ausgepackt

Bean Validation »74

Liferay Portal

Grundlagen »56

4 194586 708507 11

Datenträger enthält Info- und Lehrprogramme gemäß §14 JuSchG

Open Source BPM on the Next Level

Activiti 5.0

Im November erscheint Activiti 5.0, die erste quelloffene BPM-Plattform, die nativ BPMN 2.0 unterstützt. Activiti soll alle am Projekt beteiligten Rollen einbeziehen, sowohl Manager und Fachabteilungen als auch Java-Entwickler oder den Betrieb. Damit bietet Activiti weit mehr als „nur" eine in Java geschriebene Workflow Engine, nämlich auch Werkzeuge zur Modellierung, Überwachung und Unterstützung des BPM-Kreislaufs.

von Nils Preusker und Bernd Rücker

Wer BPM-Tools kennt, weiß natürlich, dass sich Activiti mit diesem Versprechen viel vorgenommen hat. Und natürlich wird die Version 5.0 – Überraschung – noch keine eierlegende Wollmilchsau werden. Aber das Projekt hat eine klare Vision und bewegt sich in die richtige Richtung. Aber erst einmal zurück auf Los. Denn neben dem Überblick über das Activiti-Projekt wollen wir anhand eines konkreten Beispiels vorstellen, wie ein Geschäftsprozess mit Activiti umgesetzt werden kann. Dazu bedienen wir uns eines imaginären Incident-Management-Systems zur Abwicklung von Supportanfragen eines Softwarehauses. Das Beispiel haben wir auch zur BPMN-2.0-Spezifikation beigesteuert, wo es im offiziellen Beispieldokument „BPMN 2.0 by Example" veröffentlicht wurde [1]. Aus Platzgründen können wir nur Auszüge vorstellen, das gesamte Beispiel inklusive Erläuterungen finden Sie auch in unserem Blog [2]. **Abbildung 1** zeigt die Abwicklung einer Supportanfrage als High-Level-BPMN-Diagramm. Für die technische Umsetzung müssen wir dieses Diagramm später noch verfeinern, die Abbildung dient als erster Überblick.

Wir sehen, dass Kunden ihre Supportanfragen an einen Accountmanager richten, der dann entscheiden muss, ob er sie selbst bearbeiten kann oder sie an den 1st Level Support weitergibt. Auch der Verantwortliche im 1st Level Support kann die Frage entweder selbst beantworten oder an den 2nd Level Support delegieren. Dieser hat schließlich die Möglichkeit, die Frage an einen Softwareentwickler weiterzuleiten, wenn er sie selbst nicht beantworten kann. Wenn einer der Mitarbeiter die Antwort parat hat, wird diese direkt an den Accountmanager zurückgegeben, der dann dem Kunden die Lösung des Problems erklärt.

Prozessautomatisierung

Das Diagramm zeigt einen Geschäftsprozess. Nehmen wir an, dieser soll in Zukunft automatisiert werden. Vielleicht, weil er aktuell zu viele Qualitätsprobleme – z. B. liegengebliebene Anfragen – verursacht, oder weil das Management ein Monitoring einführen will, das Aufschluss darüber geben soll, wie viele Anfragen auf welchem Weg beantwortet werden können. Oder tatsächlich auch der schönste aller Gründe, um den Mitarbeitern Werkzeuge an die Hand zu geben, sich auf ihre Arbeit zu konzentrieren und den Kopf frei zu bekommen.

Die erste Frage lautet also: Was wollen wir wie automatisieren? Hier ist das Prozessmodell sehr hilfreich, da es als Diskussionsgrundlage dient. Offensichtlich gibt es verschiedenste Tiefen der Automatisierung. Das Schöne an der BPMN ist, – seit Version 2.0 steht dies für „Business Process Model and Notation – dass sie sich ideal eignet, diese Tiefen auch korrekt darzustellen. Das Zauberwort an dieser Stelle heißt „Pools". **Abbildung 2** zeigt ein Prozessmodell, das genau angibt, wer welche Aufgaben übernimmt und dabei auch definiert, was die Softwarelösung genau umsetzen muss, nämlich alles, was in dem so genannten Pool „Trouble Ticket System" dargestellt wird. Für eine genauere Herleitung sei wie gesagt auf unseren Blog verwiesen.

Die nächste Frage lautet: Wie wollen wir den Prozess automatisieren? Neben dem reinen Ausprogrammieren können wir hierzu Activiti als Process Engine verwenden. Aber welche Vorteile bringt der Einsatz einer

Workflow Engine? Beim klassischen objektorientierten Ansatz wäre die naheliegende Lösung, ein Trouble Ticket zu modellieren, das Zustandsinformationen als Attribut im Objekt abspeichert. Die Menge der möglichen Zustände kann dann als Konstante definiert werden. Zustandsübergänge und deren Reihenfolge würden ausprogrammiert werden. Und hier liegt der erste Hase im Pfeffer: Der grafische Prozess aus unserem Diagramm wird in Java-Code versteckt! Bei Änderungen muss immer wieder zwischen Grafik und Code übersetzt werden, und das Prozessmodell wird irgendwann etwas ganz anderes zeigen als implementiert wurde. Und Geschäftsprozesse besitzen eine für die IT vielleicht unangenehme Eigenschaft: Sie tragen „Geschäft" im Namen und sind damit meist auch in Fachabteilungen oder gar auf Vorstandsebenen sichtbar. Prozesse müssen darüber hinaus immer flexibler werden, Kennzahlen sollen auf Knopfdruck zur Verfügung stehen und der Status einer Instanz – also zum Beispiel unseres Incidents – muss auch jederzeit einsehbar sein. Daher macht der Einsatz einer BPM-Plattform wie Activiti hier sehr wohl Sinn. Eine solche Plattform bietet viele dieser Funktionen „Out of the box". Nicht nur die grafische Darstellung der Prozesse, sondern auch Themen wie Aufgabenlisten, Prozessvariablen, Persistenz, Transaktionssteuerung, Archivierung von Prozessen, statistische Auswertungen, Administration, Betriebswerkzeuge und vieles mehr sind adressiert. Kurzum: Das sollte man heute nicht mehr selbst entwickeln – oder schreiben Sie auch noch Ihr eigenes Hibernate?

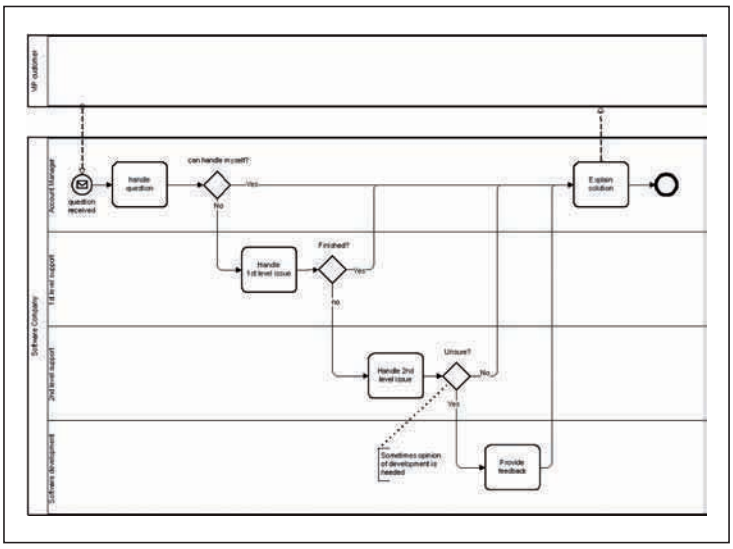

Abb. 1: High-Level-Prozess Incidient Management

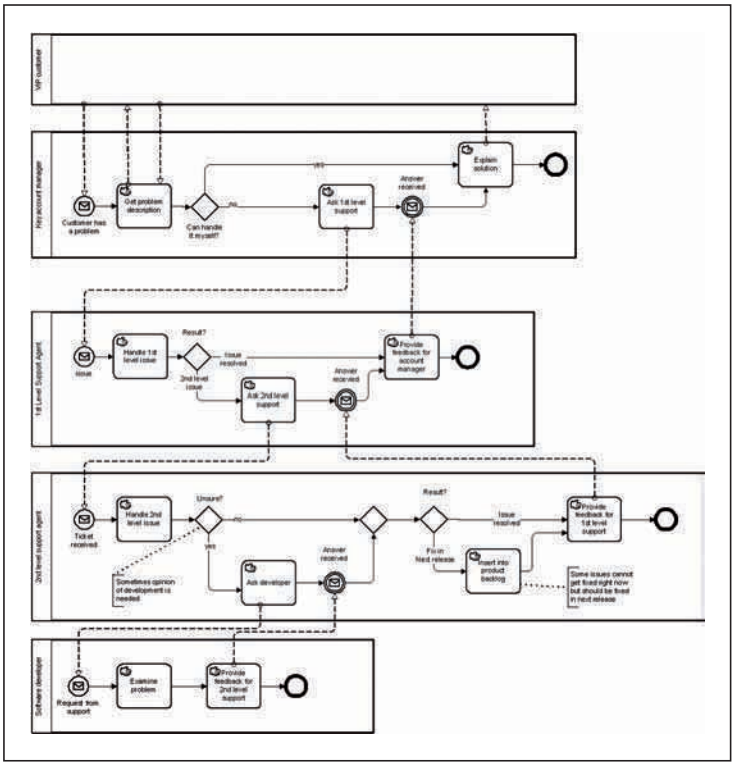

Abb. 2: Darstellung der Zuständigkeiten und Kommunikationsbeziehungen über Pools

BPMN 2.0

BPMN hat sich als akzeptierter Industriestandard bei der Geschäftsprozessmodellierung durchgesetzt. Der Knackpunkt ist, dass die BPMN beiden Welten gerecht werden, die in Softwareentwicklung aufeinanderstoßen: Business und IT. Damit bildet das BPMN-Prozessmodell die Brücke zwischen den Fraktionen, was in der Praxis auch sehr gut funktionieren kann. Die Neuheit der Version 2.0 ist nun, dass auch eine so genannte Ausführungssemantik definiert wurde. Das bedeutet, man kann ein BPMN-2.0-Prozessmodell direkt als XML abspeichern und in einer BPMN 2.0 Process Engine ausführen. Beispielsweise in Activiti, wenn es Open Source sein soll, aber auch in Produkten der großen kommerziellen Hersteller. Wie SQL soll die Engine dann austauschbar sein. Und wie bei SQL wird dies unserer Einschätzung nach nur in Maßen funktionieren. Trotzdem würden wir heute bei der Evaluierung einer Process Engine zu einer BPMN 2.0 Engine raten, zumal es hier auch sehr leichtgewichtige Vertreter gibt.

Java Process Engines und Activiti 5.0

Die wichtigste Unterscheidung bei einer Process Engine ist aus unserer Sicht, ob sie einem frameworkartigen Ansatz entspricht, oder ob die Engine als eigener Server daherkommt. Erstere sind meistens auch Open Source und können spätestens dadurch gut in eine Java-Landschaft oder die eigene Anwendung integriert werden. Die Engines sind selbst in Java geschrieben, verfügen über ein Java-API und können an definierten Stellen im Prozessablauf Java-Code ausführen. Typischerweise kann ein solches Tool in die eigene Java-EE-Architektur integriert werden und beispielsweise die Transaktionssteuerung des Containers (JTA) mit verwenden. Neben JBoss jBPM, Nova Bonita oder auch Enhydra Shark ist Activiti hier dann die neueste Engine. Wer übrigens jBPM kennt, wird sich auch in Activiti schnell zurechtfinden, denn da der jBPM Project Lead und ein Core Developer in dieses Projekt gewechselt sind, fühlt sich

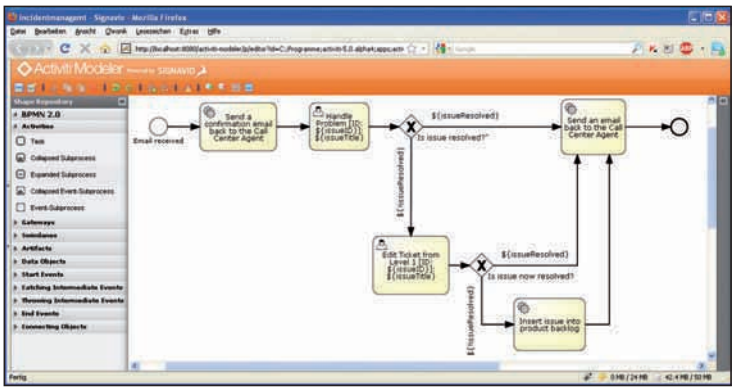

Abb. 3: Der zu automatisierende Prozess im Activiti Modeler

Listing 1

```
<definitions>
  <process id="incidentManagement" name="Incident Management">
    <startEvent id="emailReceived" name="Email received"/>
    <sequenceFlow sourceRef="emailReceived" targetRef="sendConfirmationEmail" />
    <serviceTask id="sendConfirmationEmail"  name="Send a confirmation email back to
                                        the Call Center Agent"activiti:class="com.camunda...
                                          ToConfirmEmailCallCenterAgentActivityBehavior" />
    <sequenceFlow sourceRef="sendConfirmationEmail" targetRef="editTicketLevel1" />
    <userTask id="editTicketLevel1" name="Handle Problem [ID: ${issueID}]: ${issueTitle}"
      activiti:form="com/camunda/.../level_1_support.form">
      <humanPerformer>
        <resourceAssignmentExpression>
          <formalExpression>kermit</formalExpression>
        </resourceAssignmentExpression>
      </humanPerformer>
    </userTask>
    <sequenceFlow sourceRef="editTicketLevel1" targetRef="issueResolvedGateway" />
    <exclusiveGateway id="issueResolvedGateway" name="Was issue resolved?" />
    <sequenceFlow sourceRef="issueResolvedGateway" targetRef=
                                        "informCallCenterAgent" name="Problem solved" >
      <conditionExpression xsi:type="tFormalExpression">${issueResolved}
                                                          </conditionExpression>
    </sequenceFlow>
    <sequenceFlow sourceRef="issueResolvedGateway" targetRef=
                                        "editTicketLevel2" name="Problem not solved" >
      <conditionExpression xsi:type="tFormalExpression">${!issueResolved}</
conditionExpression>
    </sequenceFlow>
```

Listing 2

```
public class ToConfirmEmailCallCenterAgentActivityBehavior implements ActivityBehavior {
  public void execute(ActivityExecution execution)  {
    String issueID = execution.getProcessInstance().getId();
    String issueTitle = (String) execution.getVariable("issueTitle");
    ...
    performDefaultOutgoingBehavior(execution);
  }
}
```

die Engine naturgemäß ähnlich an. Für nähere Details zu den Innereien müssen wir leider aus Platzgründen auf die Onlinedokumentation von Activiti verweisen [3].

Vom BPMN zur Ausführung

Aber zurück zu unserem Beispielprozess. **Abbildung 3** zeigt den Ausschnitt des umzusetzenden Modells als Screenshot im Activiti Modeler, der grafischen Modellierungskomponente von Activiti. Der browserbasierte Activiti Modeler ist eine abgespeckte Open-Source-Version des kommerziellen Modellierungstools Signavio. Beim Speichern des Modells wird im Hintergrund auch immer das XML mit dem BPMN-2.0-Quellcode abgelegt. Listing 1 zeigt einen Ausschnitt aus diesem Quellcode, das komplette lauffähige Beispiel kann von unserem Blog heruntergeladen werden [4].

Die grafischen Informationen des Prozesses sind übrigens im gleichen XML weiter unten enthalten, aber für die Ausführung erst einmal nicht von Bedeutung, weswegen wir sie hier ausgespart haben. Wir möchten in diesem Beispiel nicht auf alle verfügbaren BPMN-Elemente eingehen, dennoch die verwendeten Knotentypen kurz erläutern [5].

Das Startevent ist sozusagen der Startknoten des Prozesses, an dem die Prozessausführung beginnt. Sequence Flows sind die Verbindung zwischen Knoten, zur Vorstellung zieht man am besten das Token-Konzept zu Rate. Jede Prozessinstanz – also genau ein Incident in unserem Fall – kennt zu Anfang einen Wurzel-Token. Dieser wandert an den Sequence Flows entlang durch das Prozessmodell. Kommt er an einem so genannten Exclusive Gateway (XOR) an, muss er sich für einen Weg entscheiden. Im XML ist zu sehen, dass dies in unserem Beispiel mithilfe der Java Unified Expression Language geschieht, wobei eine Prozessvariable ausgewertet wird. Eine Prozessvariable ist im einfachsten Fall ein Java-Objekt, das mit der aktuellen Prozessinstanz verbunden ist. Wir möchten uns an dieser Stelle damit begnügen zu erwähnen, dass die Variablenpersistenz sehr flexibel konfiguriert werden und prinzipiell auch auf die Nutzung von XML umgestellt werden kann. Damit könnte man auch XPath für Gateways verwenden. Dies ist übrigens auch eine große Neuerung von BPMN gegenüber BPEL: Das Binding an eine bestimmte Technologie wie Java oder auch Web Services wird offen gelassen.

Bleiben zwei weitere, sehr typische Knoten: Service und User Tasks. Service Tasks sind für Serviceaufrufe gedacht. Im einfachsten Fall wird dabei Java-Code aufgerufen, wie in unserem Beispiel zu sehen. Listing 2 zeigt die im Prozess verwendete Java-Klasse. Dort können Sie sehen, dass ein Kontext hereingereicht wird, über den die Klasse mit der Engine kommunizieren kann. Dass dies als „Verhalten“ implementiert ist, ist aktuell noch ein Workaround mit der Beta 1, da noch keine so genannten EventListener zur Verfügung stehen. Aktuell arbeiten wir übrigens auch an der direkten Web-Service-Unterstützung in Activiti.

User Tasks sind, wie der Name vermuten lässt, für Human Task Management gedacht. In diesem Fall wird

einem Benutzer des Systems eine Aufgabe in die Aufgabenliste gelegt. Im Beispiel wird jede Aufgabe dem Activiti-Standarduser „kermit“ zugewiesen, in einer echten Anwendung könnten wir hier auch auf Daten des Prozesses zugreifen, um zum Beispiel den Bearbeiter je nach Komponente zuzuordnen oder auch LDAP oder gar eine Regelmaschine dazu befragen. Der BPMN-Standard definiert übrigens nicht, wie Formulare für Aufgaben auszusehen haben. Letztendlich ist das auch in Ordnung, da es einfach zu viele verschiedene Oberflächentechnologien gibt, denen man unmöglich allen gerecht werden kann. Activiti definiert an dieser Stelle eine Erweiterung, um ein spezielles HTML-Formular zur Verfügung zu stellen, das dann durch die Weboberfläche ausgewertet werden kann.

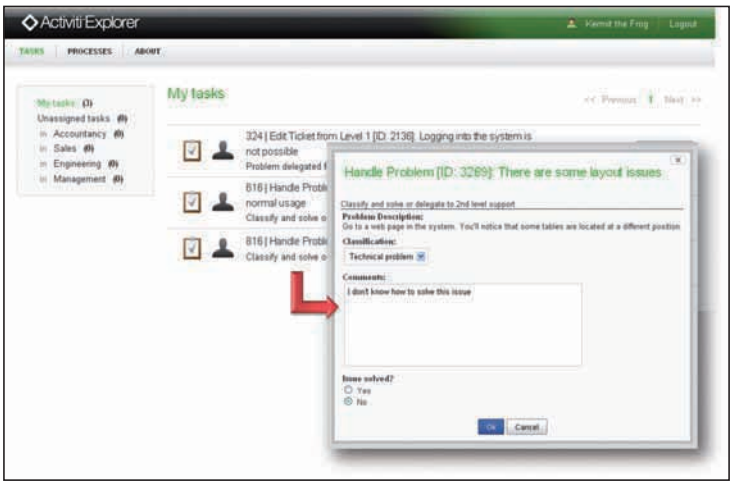

Abb. 4: Taskliste und Formular

Die Benutzerperspektive

Für den Endnutzer bietet die Activiti-Plattform den Activiti Explorer. Hier sind laufende Prozessinstanzen einsehbar, und wir können beispielsweise überprüfen, wo ein einzelner Incident gerade steht. Dies ist vergleichbar mit der Situation, wenn wir bei einem Onlineshop wissen möchten, wie es aktuell um unsere Bestellung steht. Ein weiteres Feature der Komponente ist Reporting. Auch wenn im aktuellen Stand (zum Zeitpunkt des Schreibens dieses Artikels ist dies die Beta 1) noch keine Reports verfügbar sind, so sollen diese bald folgen. Sie können Fragen nach durchschnittlichen Laufzeiten oder auch Wartezeiten vor Knotenbearbeitungen beantworten. Schlussendlich enthält der Explorer auch eine Oberfläche zur Einbeziehung der „normalen“ Mitarbeiter in den Prozess, nämlich eine Aufgabenliste und die oben bereits erwähnten Formulare. **Abbildung 4** zeigt anhand eines Screenshots, wie ein Benutzer dabei von der Aufgabenliste in das verknüpfte Formular springen kann. Hat er die Aufgabe beendet und schließt das Formular, so läuft im Hintergrund der Prozess weiter.

API und Testing

Ein großer Vorteil der leichtgewichtigen Java Engines ist, dass wir einfache Prozesstests mit JUnit schreiben können. Dabei kann man direkt auf das Client-API von Activiti zugreifen. In JUnit-Tests muss dann kein Server laufen, da die Activiti Engine direkt im Test ohne Persistenz hochgefahren wird. Dies ermöglicht einfache, schnelle Tests ohne notwendige Aufräumarbeiten. Listing 3 zeigt in einem Unit Test die Verwendung des API.

Will man den Client übrigens unabhängig von Activiti halten, wird sehr häufig eine Abstraktionsschicht eingebaut. Wir haben daher inzwischen mit Kunden zusammen den Process Engine Abstraction Layer PEAL entwickelt, aktuell mit Binding gegen jBPM und Activiti, der ebenfalls Open Source verfügbar ist [6].

Kollaboration in BPM-Projekten

Sagt man BPM, so denken viele leider immer noch an Prozessautomatisierung per Knopfdruck, idealerweise durch die Fachabteilung. Dahinter steckt der Traum, endlich die IT loszuwerden. Leider hat sich gezeigt, dass dies nicht oder nur in bestimmten eingeschränkten Anwendungsfällen funktioniert. In allen anderen Projekten wird nach wie vor Softwareentwicklung betrieben, und der Prozess ist nur ein Teil davon. Nach unserer Erfahrung sind in diesen Projekten Tools, die den Knopfdruck im Hinterkopf haben, oft ungeeignet, da sie dem Entwickler das Leben oft unnötig schwer machen, wie wir es hier bereits einmal beschrieben haben [8]. Des Weiteren sind agile Werte mehr und mehr auf dem Vormarsch und haben sich in Praxisprojekten bewährt. Agilität bedeutet aber auch Kommunikation über Tools. Und deshalb geht Activiti mit der Komponente Cycle in Bezug auf Softwareentwicklungsprojekte einen anderen Weg als viele andere Werkzeuge.

Cycle versucht, die Brücke zwischen Business und IT zu schlagen. Technisch betrachtet manifestiert sich dies zuallererst darin, dass in Cycle verschiedene Repositories zusammengeführt werden können. Man kann dort prinzipiell sowohl die fachlichen Prozessmodel-

Listing 3

```
public class IncidentManagementTest extends ProcessEngineTestCase {

@Deployment
public void testHappyPath() {
  Map<String,Object> variablesTable = new HashMap<String,Object>();
  variablesTable.put("issueTitle", "TestIssue");
  // ...

  ProcessInstance pi = runtimeService.startProcessInstanceByKey
                                          ("incidentManagement", variablesTable);

  Task task = taskService.createTaskQuery().singleResult();
  assertEquals("Handle Problem [ID: 1]: TestIssue", task.getName());
  taskService.complete(task.getId());
  // check next state...
}
```

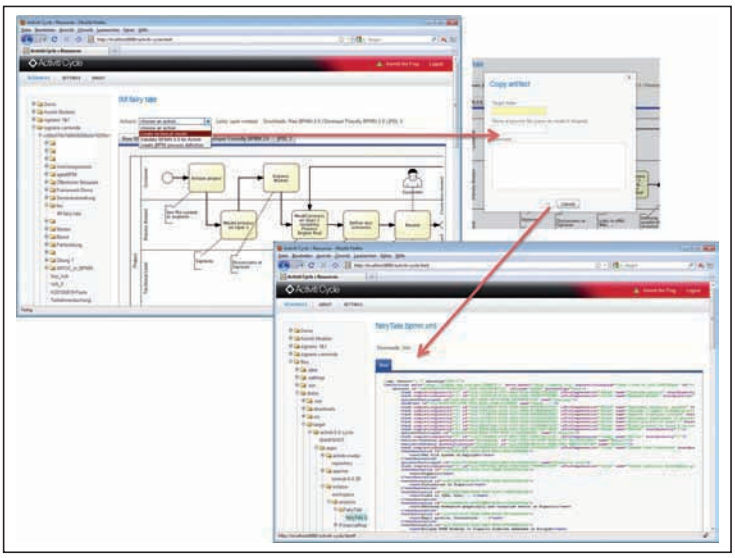

Abb. 5: Kopieren des BPMN 2.0 XML in ein eigenes Repository mit Activiti Cycle

le aus Signavio, die Entwicklungsartefakte aus SVN, Besprechungsprotokolle aus dem Dateisystem oder E-Mail-Diskussionen in einem Archivsystem zusammenführen. Dies ermöglicht es beispielweise, Änderungen im technischen Modell zu erkennen und den Analysten darüber zu informieren, dass das fachliche Modell aktualisiert werden muss. Cycle setzt dabei die Philosophie um, dass Tools auf Erweiterbarkeit ausgelegt sein sollten und sich einfach an das Vorgehen im eigenen Projekt anpassen lassen, statt Entwicklern ein Korsett aufzuzwingen. So haben wir beispielsweise in einem Pilotprojekt bereits erfolgreich ein Roundtrip-Vorgehen mit BPMN 2.0 und jBPM 3 in Cycle umgesetzt, auch wenn jBPM praktisch eine Konkurrenz-Engine ist. Die erste Vorschau auf Cycle ist bereits im Beta-1-Release enthalten [9], dies ist aber erst der Anfang. **Abbildung 5** zeigt beispielweise die Activiti-Cycle-Oberfläche beim Kopieren eines Prozessmodells in den Entwickler-Workspace.

Status und Fazit

Activiti, das ganz ambitioniert direkt mit der Version 5.0 startet, wird im November das Betastadium verlassen. In diesem Release wird sicherlich noch einiges fehlen, auch wird BPMN 2.0 wohl noch nicht vollständig unterstützt werden. Derzeit wird auch noch viel Energie in die Stabilisierung der Core Engine gesteckt, wofür derzeit größere Refactorings stattfinden. Aus unserer Sicht ist dies sicherlich ein zielführenderes Vorgehen, als mit möglichst vielen instabilen Features aufzuwarten.

Die grafische Oberfläche kann sich sehen lassen und der Modeler ist gut, auch wenn es aktuell noch kleine Inkompatibilitäten dank der Änderungen in der finalen BPMN-2.0-Spezifikation gibt, die aber bis November auch beseitigt sein werden und aktuell schon durch Activiti Cycle im Entwicklungsprozess ausgebügelt werden. Wermutstropfen für Entwickler ist aktuell, dass es kein Eclipse Modeler gibt, aber hier rumort es bereits in einigen Open-Source-Ecken. Und dank Standardisierung wäre hier jeder BPMN 2.0 Modeler direkt nutzbar. Bis November soll auch der Activiti Modeler direkt im Eclipse-Browser funktionieren.

Monitoring und Überwachung ist sicherlich bis dahin noch eine offene Flanke, steht aber auf der Roadmap. Dank ordentlicher Teamgröße bei Activiti ist dies auch nicht unrealistisch. Und last, but not least steht mit Activiti Cycle eine Komponente in den Startlöchern, den Entwicklungsprozess rund um Prozesse und Regeln bedeutend verbessern kann. Im Vergleich zu jBPM-Zeiten ist dies ein deutlicher Schritt vom reinen Entwicklerlager heraus in die große weite Welt der vollwertigen BPM-Plattformen.

Ran an die Tastatur!

Die Installation der Umgebung ist relativ einfach. Voraussetzung ist lediglich Java und ein installiertes Ant. Dazu laden wir uns zunächst Activiti 5.0 Beta 1 herunter [7]. Nach dem Entpacken des Archivs können in der *build.properties* ein paar Einstellungen verändert werden, danach kann man direkt mit *ant demo.setup* eine Standardkonfiguration mit Tomcat und H2-Datenbank installieren. Dann noch fix das vorgestellte Beispiel herunterladen [4], ebenfalls entpacken, Properties anpassen und per *ant* in das installierte Activiti deployen. Das Deployment der Prozesse in die Datenbank der Engine erfolgt dabei über die Ant Task direkt.
Viel Vergnügen!

Nils Preusker (nils.preusker@camunda.com) ist Berater, Trainer und Softwareentwickler bei der camunda services GmbH. Er setzt jBPM und Activiti in verschiedenen Kundenprojekten ein und ist Committer im Activiti-Projekt.

Bernd Rücker (bernd.ruecker@camunda.com) ist Geschäftsführer der camunda services GmbH. Er arbeitet als Berater, Trainer und Coach in verschiedenen Kundenprojekten und ist Committer bei JBoss jBPM, Activiti und Project Lead von Activiti Cycle. Er ist Autor mehrerer Fachbücher, zahlreicher Fachartikel und regelmäßiger Sprecher auf Konferenzen.

Links & Literatur

[1] http://www.omg.org/cgi-bin/doc?dtc/10-06-02

[2] http://www.bpm-guide.de/2010/07/15/bpmn-2-0-am-beispiel-incident-management/

[3] http://www.activiti.org/userguide/

[4] http://www.bpm-guide.de/activiti

[5] http://www.bpmb.de/index.php/BPMNPoster

[6] http://www.bpm-guide.de/2010/04/17/abstracting-the-process-engine/

[7] http://activiti.org/download.html

[8] http://www.infoq.com/articles/collaborativeBPM

[9] http://www.bpm-guide.de/2010/08/27/activiti-cycle-explained/