JBoss jBPM 4

Prozessautomatisierung, BPM und SOA sind nicht tot, wie mancherorts gebloggt wird, sondern putzmunter. Zudem erhärtet sich der Eindruck, das BPEL doch nicht für alle Probleme die Lösung darstellt. Genau in dieser Stimmung bringt JBoss die schon länger erwartete Version 4 der leichtgewichtigen, quelloffenen Java Process Engine jBPM heraus. Das schauen wir uns doch einmal an.

von Bernd Rücker und Falko Menge

Starten wir mit einem kleinen Beispiel: Ein Designershop verkauft selbstgeschneiderte Taschen. Da immer mehr Bestellungen über Internet getätigt werden und steigende Bestellzahlen Chaos verursachen, benötigt man zusätzliche Softwareunterstützung. Genau genommen muss die Steuerung des Geschäftsprozesses *Bestellung* automatisiert werden, auch wenn man das im Designershop noch nicht weiß.

Üblicherweise könnte man nun die notwendigen Anforderungen in Java-Code gießen. Das hätte jedoch entscheidende Nachteile. Zwei Beispiele: Bis eine Tasche genäht und verschickt ist, vergeht Zeit. Also läuft der Prozess über mehrere Tage oder Wochen. Wir müssen daher Zwischenzustände persistent in einer Datenbank speichern. Des Weiteren möchten wir bei zu langer Wartezeit eine automatische Eskalation einbauen und brauchen dafür Timer-Funktionen. Prinzipiell kann ein Java-Entwickler diese Anforderungen natürlich umsetzen. Das Problem ist jedoch: Es dauert einfach zu lange und ist viel zu umständlich. Denn genau für diesen Einsatzweck, der so genannten *Process Execution*, gibt es bereits

fertige Frameworks, *Process Engines* genannt, häufig auch synonym als *Workflow Engines* bezeichnet. Diese unterstützen auch eine grafische Darstellung des Geschäftsprozesses, sodass er also nicht in Java-Code versteckt, sondern an die Oberfläche geholt und sichtbar gemacht wird. Dieser Transparenzgedanke ist u. a. ein wichtiger Treiber der Prozessmaschinen.

Wir sprechen hier nun bewusst von Frameworks. Prinzipiell gibt es bei den Process Engines verschiedene Architekturen. Sehr bekannt sind Prozessmaschinen die den Standard Business Process Execution Language (BPEL) umsetzen. Das ist jedoch nicht die einzige Variante. Durchaus verbreitet und für den skizzierten Anwendungsfall sehr viel geeigneter sind Java-basierte Tools, die einerseits ohne den Overhead von WSDL und Web Services auskommen und andererseits perfekt in eine Java-Landschaft oder die eigene Anwendung integriert werden können.

Ein Vertreter dieser Zunft ist eben auch JBoss jBPM [1]. Vergleichbare Open-Source-Produkte sind z. B. Nova Bonita [2] oder Enhydra Shark [3], wobei jBPM unserer Meinung nach aktuell bei Downloadzahlen, Community-Größe und Aktivität sowie dem verfügbaren Support die Nase vorn hat. Alle diese Engines sind in Java geschrieben, verfügen über eine Java-API und können an definierten Stellen im Prozessablauf Java-Code ausführen. Typischerweise erfolgt die Persistenz von Prozessmodellen oder Instanzen über Standardwege, im Fall von jBPM mit Hibernate. Somit kann ein solches Tool in die eigene Java-EE-Architektur integriert werden und z. B. die Transaktionssteuerung des Containers (JTA) mitverwenden.

Diese Nähe zu Java macht die Tools wesentlich einfacher beherrschbar und reduziert viel Komplexität, die beispielsweise BPEL-Maschinen mitbringen. Typische Java-Entwickler können sich daher auch relativ einfach in diese Technologie einarbeiten.

Beispielprozess in jBPM 4

Abbildung 1 zeigt den Beispielbestellprozess unserer Designertaschen. Die Notation von jBPM 4 ist dabei an den Standard Business Process Modeling Notation (BPMN) [4] angelehnt, auch wenn noch nicht alle Möglichkeiten unterstützt werden und einige Auslegungen leider noch nicht hundertprozentig korrekt sind [5]. Dennoch sind die Prozessdiagramme ein sehr wichtiges Kommunikationsmittel und eine große Hilfe in der Diskussion mit der Fachabteilung oder dem Anwender, in unserem Fall eben dem Betreiber des Designershops.

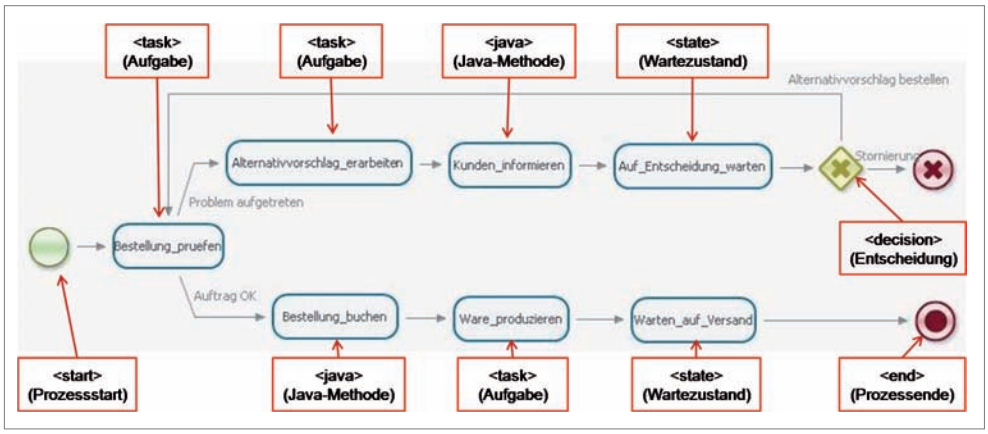

Abb. 1: Beispielprozess mit jBPM 4

Hinter der grafischen Darstellung steckt eigentlich ein XML-Dokument, das die Prozessbeschreibung inklusive notwendiger technischer Details enthält. Im Gegensatz zu jBPM 3 sind auch die grafischen Informationen darin enthalten. Ein Ausschnitt aus der Prozessdefinition ist in Listing 1 dargestellt, der gesamte Prozess findet sich in unserem Blog [6].

Ein Prozess ist in jBPM ein gerichteter Graf, in dem es verschiedene Knotentypen, auch Activities genannt, gibt. Diese verschiedenen Typen werden übrigens bald auch noch durch kleine Icons in der grafischen Prozessdarstellung unterstützt. Von Haus aus bringt jBPM neben Start- und Endknoten bereits einige Standardtypen mit. So gibt es *task*s, die menschliche Interaktion kennzeichnen, so genannte Benutzeraktivitäten oder auch Human Tasks. An dieser Stelle bekommt ein Anwender eine Aufgabe in seine elektronische Aufgabenliste und die Process Engine wartet im Prozess auf die Erledigung dieser Aufgabe. Die Aufgabenliste ist mit einem E-Mail-Posteingang vergleichbar, auch dort wartet der Absender typischerweise auf Erledigung der Aufgabe, was in diesem Fall das Schreiben einer Antwort bedeutet. Nach Bearbeitung der Aufgabe durch den Menschen wird der Prozess informiert und läuft entsprechend weiter.

Listing 1: Ausschnitt des Prozessdefinitions-XML

```
<process name="Bestellung" xmlns="http://jbpm.org/4.0/jpdl">
 <start name="start" g="18,150,48,48">
  <transition to="Bestellung pruefen"/>
 </start>

 <task candidate-groups="Lager" name="Bestellung_pruefen"
    g="191,149,109,52">
  <transition name="Problem aufgetreten"
       to="Alternativvorschlag_erarbeiten"
       g="267,103:-3,27"/>
  <transition name="Auftrag OK" to="Bestellung_buchen"
       g="268,236:0,-25"/>
 </task>

 <java name="Kunden informieren"
    class="com.camunda.training.jbpm.service.MockService"
    method="informCustomer" g="493,75,119,52">
     <arg><object expr="#{order.customer}"/></arg>
     <arg><object expr="#{problem}"/></arg>
  <transition to="Auf Entscheidung warten"/>
 </java>

 <decision name="Stornierung?" g="804,84,48,48">
  <transition name="Alternativvorschlag bestellen"
       to="Bestellung_pruefen"
       g="828,43;245,43:247,-19">
    <condition expr="#{orderChanged}"/>
  </transition>
  <transition name="Stornierung" to="cancel" g="-26,-18"/>
 </decision>

 <state name="Auf Entscheidung warten" g="695,72,144,52" >
  <transition to="Stornierung?"/>
 </state>
 ...
 <end name="end" g="906,212,48,48" />
 <end-cancel name="cancel" g="905,152,48,48"/>
</process>
```

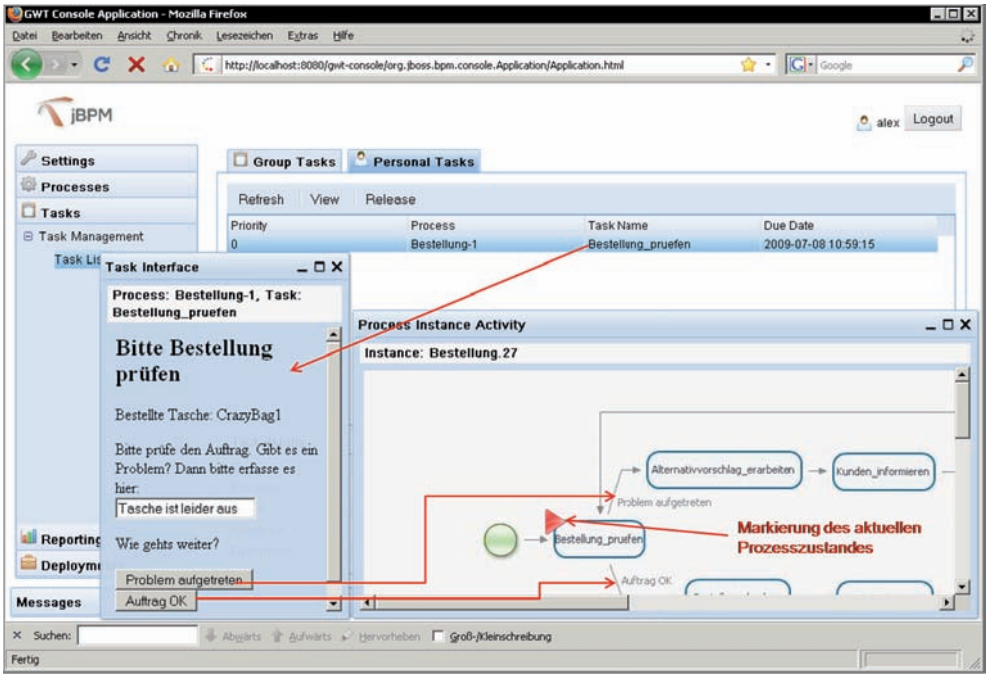

Abb. 2: Webkonsole mit Aufgabenliste, Aufgabenformular und Anzeige des aktuellen Prozesszustands

Einfache Wartezustände werden durch *state*s repräsentiert. In einem solchen Zustand wartet die Engine auf ein eintretendes Ereignis, beispielsweise das Eintreffen einer Nachricht per E-Mail oder JMS oder auch das Eintreffen eines Zahlungseingangs oder physikalischen Pakets. Durch das Eintreffen des Ereignisses wird die Prozessinstanz wieder angestoßen und läuft entsprechend weiter. Wichtig: Das Auffangen des Events und das Triggern von jBPM ist Aufgabe der Anwendung und muss programmiert werden. Von der Möglichkeit, dass das ein ESB für uns erledigen könnte, sehen wir an dieser Stelle ab.

Ein weiteres Konstrukt ist die *decision*, in der ein Prozess genau einen Pfad aus mehreren Möglichkeiten auswählen muss. Diese Auswahl erfolgt vollautomatisch nach angegebenen Kriterien. Im Beispiel in Listing 1 wird dazu geprüft, ob es eine Prozessvariable *orderChanged* gibt und ob diese den Wert *true* hat, dann wird die Transition *Alternativvorschlag bestellen* gewählt, ansonsten die andere.

Als Letztes möchten wir an dieser Stelle den *java*-Knoten beschreiben. Dieser ruft Java-Code auf, wobei Klasse und Methode angegeben werden müssen. Parameter werden per Expression Language aus Prozessvariablen gesetzt und Rückgabewerte können zurückgeschrieben werden. Für Java-Code gibt es allerdings auch die Möglichkeit, einen EventListener zu schreiben und diesen an diverse Events im Prozessablauf zu hängen, z. B. an den *start*-Event einer Aktivität, der dem *node-enter* von jBPM 3 entspricht.

Natürlich gibt es noch weitere Sprachkonstrukte, z. B. Subprozesse, Skripte, Timeouts oder auch Parallelisierung über Fork und Join. Näheres dazu kann der recht guten Dokumentation entnommen werden [7]. Spannend ist noch, dass die Sprache auch um Konstrukte erweitert werden kann, um eigene Anforderungen oder auch eine domänenspezifische Sprache zu realisieren.

Webkonsole

jBPM 4 bringt auch eine Webanwendung mit, die, wie bei JBoss aktuell üblich, mit dem Google Web Toolkit (GWT) realisiert wurde. Abbildung 2 zeigt einen Screenshot der so genannten Webkonsole im Einsatz der Demoanwendung. Prinzipiell wird die Webanwendung hauptsächlich zu Administrations- oder Prototyping-Zwecken genutzt. Als tatsächliche Anwendung für den Endbenutzer ist sie jedoch nicht ausgelegt. Auch zeigt die Projekterfahrung mit verschiedensten Tools, dass die Endbenutzeroberfläche häufig in die eigene Umgebung integriert wird.

Grundsätzlich kann die Anwendung Prozessdefinitionen, laufende Prozessinstanzen und Aufgabenlisten anzeigen, wobei jeweils in unterschiedliche Details gesprungen werden kann. Öffnet man eine Aufgabe, wird ein Formular zur Bearbeitung der Aufgabe und Anzeige der Prozessinformationen dargestellt. Dieses Formular wird als Freemarker Template realisiert und muss bei der Aufgabe in der Prozessdefinition explizit konfiguriert werden.

Interessant ist, dass in der Webkonsole bereits prototypisch Reporting in Form von Eclipse Birt realisiert ist. Wenn die angebotenen Reports nicht ausreichen, können sie einfach als Vorlage verwendet werden, um angepasste Auswertungen zu erstellen.

Architektur

Version 4 der Prozessmaschine ist eine komplette Neuentwicklung. Code aus jBPM 3 wurde kaum wiederverwendet. Das mutet zuerst eigenartig an, denn wirklich schlecht ist der jBPM-3-Code nicht. Der Schritt wird jedoch nachvollziehbar, wenn man sich die grundlegenden Änderungen in der Architektur ansieht.

Die wichtigste Änderung ist die Einführung der Process Virtual Machine (PVM) [8]. Die PVM ist aus einer einfachen Idee heraus entstanden: Es wird immer verschiedene Prozesssprachen geben, da eben nicht eine Lösung auf alle Probleme passt. So ist BPEL bei Integrationsprojekten mit XML und Web Services vielleicht eine gute Wahl, in einer Umgebung mit Java-Services aber nicht. Daher wurde eine Abstraktionsschicht eingeführt, dazu wurden die Grundkonzepte eines Zustandsautomaten, auf den alle gängigen Sprachen abgebildet werden können, sowie eine stabile API zur Verwendung der Prozessmaschine in der PVM abstrahiert. jBPM 4 selbst bzw. deren Sprache jPDL (jBPM Process Definition Language), ist dann „nur“ eine Sprache auf dieser virtuellen Maschine. Mit Nova Bonita steht übrigens auch eine XPDL-Engine auf der JBoss PVM zur Verfügung, auch wenn Bonita leider auf einer recht alten Version aufsetzt.

Anzeige

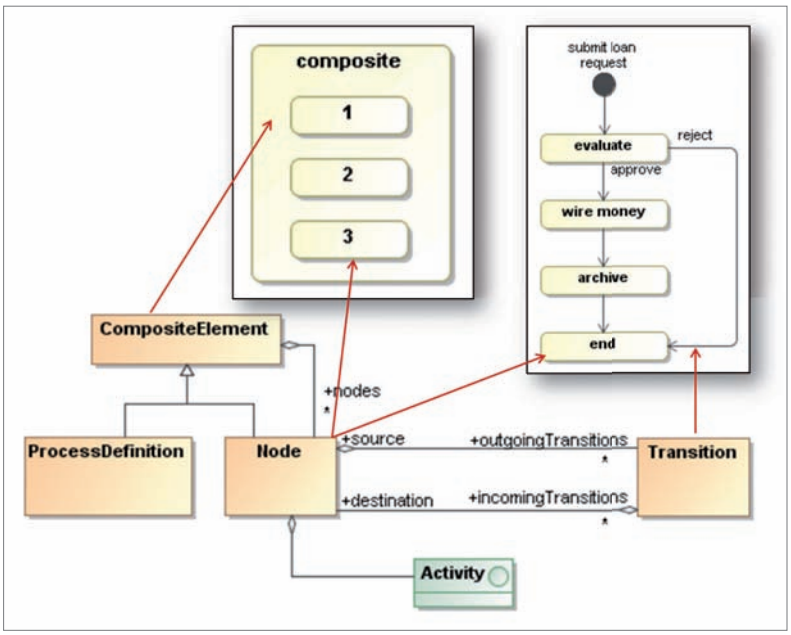

Abb. 3: Grundabstraktionen der Process Virtual Machine

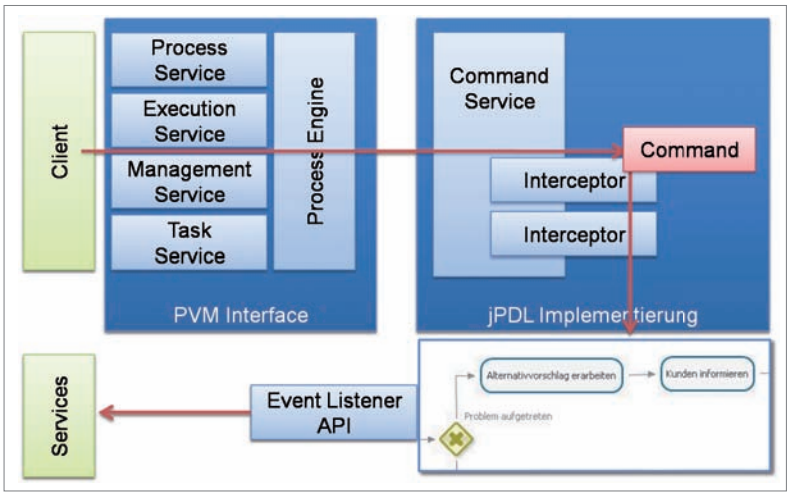

Abb. 4: Interner Aufbau von jBPM

Abbildung 3 zeigt die grundlegendsten Abstraktionen des in der PVM umgesetzten Zustandsautomaten: *Node*, *Transition* und *CompositeElement*. Die Sprache wird dann über das Verhalten bestimmt, das an gewisse Nodes angehängt wird. Beispielsweise wird der *task*-Aktivität angehängt, sich wie ein Wartezustand zu verhalten sowie eine Aufgabe zu erzeugen. Dank der Composites können neben Grafen übrigens auch Blockstrukturen, z. B. Sequenzen, wie in BPEL oft üblich, abgebildet werden.

Mit der PVM kommt auch eine deutlich stabilere API ins Spiel. In jBPM 3 hat man noch sehr objektorientiert entwickelt, z. B. kann dort eine Aufgabe folgendermaßen gesucht und beendet werden:

```
List<TaskInstance> tasks = jbpmContext.get-
TaskMgmtSession().findPooledTaskInstances
                                    ("Vertrieb");
TaskInstance ti = tasks.get(0);
ti.assign("bernd"); // claim task
ti.end();
```

Die internen Klassen waren somit gleichzeitig auch die API. Zusätzlich gab es den *jBPMcontext* und diverse Sessions als Wrapper um die Konfiguration und Hibernate. jBPM 4 führt hier eine echte API in Form von Serviceinterfaces ein. Die gleiche Funktionalität wird in jBPM 4 folgendermaßen umgesetzt:

```
TaskService taskService =
                    processEngine.getTaskService();
List<Task> tasks = taskService.
findGroupTasks("Vertrieb");
Task t = tasks.get(0);
taskService.takeTask(t.getId(), "bernd"); //claim
task
taskService.completeTask(t.getId());
```

Dabei versteckt die so genannte *processEngine* – ein Interface der PVM – die eigentliche Umgebung. Was hinter diesen Schnittstellen steckt, ist in der API nicht sichtbar. Hier könnte prinzipiell auch ein BPEL- oder XPDL-Prozess angesprochen werden. In jBPM 4 steckt hinter diesen Schnittstellen ein *CommandService*. Ein Aufruf der *completeTask*-Methode schickt intern ein *CompleteTaskCmd* an diesen. Der Clou an dieser Architektur ist nun, dass an diesem *CommandService* einerseits sehr einfach Interceptors angebracht werden können, etwa für Transaktionssteuerung oder Logging, andererseits können diese Commands auch remote per RMI oder asynchron per JMS über die Leitung geschickt werden. Die Umstellung auf Remote-Kommunikation per EJB-*SessionBean* ist dann durch eine einfache Konfigurationsoption möglich. Der Code des Clients muss hierfür nicht geändert werden. Die verschiedenen Services sowie eine Übersicht dieser Architektur ist in Abbildung 4 gezeigt.

jBPM 3 vs. 4

Aktuell stellt sich natürlich die Frage, welche Version von jBPM in einem neuen Projekt eingesetzt werden soll. Da die Version 4 eine Neuentwicklung darstellt und gewisse Releasetermine eingehalten werden mussten, ist die Funktionalität noch deutlich geringer als bei jBPM 3. Auch die Webkonsole hat noch Defizite. jBPM 3 dagegen ist extrem stabil und auch in großen Projekten bewährt, befindet sich jetzt allerdings im Wartungsmodus. Daher sind neue Features nicht mehr zu erwarten. Die neu eingeführte API sowie die Nutzung der BPMN als Modellierungsnotation sind gute Gründe für die neue Version, die bis Ende des Jahres auch wieder mit fast allen bekannten Features aufwarten sollte, denn jBPM unterliegt inzwischen einem zweimonatigen Releasezyklus, sodass zum 01.11.2009 die Version 4.2 vorliegen wird.

Nicht aussparen möchten wir an dieser Stelle die Frage der Migration. Es

wird aktuell an einem Migrationstool gearbeitet, dass XML-Prozessdefinitionen in die neue Version konvertieren kann. Trotzdem gestaltet sich die Umstellung aufwendig, da sich die API verändert hat und auch laufende Prozessinstanzen nicht mit Board-Mitteln migriert werden können. Immerhin ist ein Parallelbetrieb von jBPM 3 und 4 möglich und auch für Version 3 ist noch einige Zeit Support verfügbar.

jBPM und „das Business"

Regelmäßig sehen wir uns mit der Frage konfrontiert, wie die Fachabteilungen selbst modellieren können und dieses Modell auf jBPM transformiert wird. Die schlechte Nachricht ist, dass das in der Praxis nicht funktioniert, weil ausführbare Modelle viele technische Details benötigen, was sie für den Fachbereich schnell zu komplex und unübersichtlich werden lässt. Auch sind fachliche Modelle häufig für einen speziellen Kommunikationszweck abgestimmt, sodass bewusst Details weggelassen werden sollen.

Die gute Nachricht ist: Durch Nutzung der gleichen Notation – hier BPMN – sowie strukturell ähnlicher Modelle können sich Fachbereich und IT annähern. Dabei ist es auch bereits ein großer Fortschritt, wenn die jeweils anderen Modelle verstanden werden, sodass die Kommunikation zwischen den Welten möglich ist.

In jBPM wurde dieses Ziel bisher nur insofern verfolgt, dass technische Details des Prozesses im XML „versteckt" sind und somit die grafische Repräsentation nicht aufgebläht wird und fachlich verständlich bleibt. Das ist natürlich auch weiterhin der Fall und wird durch die Nutzung der BPMN noch fachbereichstauglicher. Trotzdem glauben wir, dass es weiterhin getrennt fachliche und technische Modelle geben wird, auch wenn diese näher zusammenrücken. Ein interessanter Ansatz ist aktuell das Projekt, den quelloffenen Webmodeller Oryx des HPI der Uni Potsdam [9] in jBPM zu integrieren. Ziel ist es, dort eine fachliche Modellierung zu ermöglichen und diese fachlichen Modelle als Ausgangspunkt für die technischen zu „kopieren". Andererseits kann der Fachanwender dort sehen, was sich im technischen Modell geändert hat, um eventuell „sein" Modell anzupassen. Dazu soll ein Link zwischen den Modellen erhalten bleiben und beide Fraktionen sollen auf dem gleichen Prozess-Repository arbeiten können. Natürlich ist das kein automatischer Roundtrip, aber eine gute Vision.

Eine zweite Baustelle ist aktuell die Entwicklung einer BPMN-2.0-Ausführungs-Engine. Diese kann das standardisierte BPMN-2.0-XML-Format einlesen und ausführen [10]. Die Implementierung steckt noch in den Kinderschuhen, allerdings wird bis zur Verabschiedung der Spezifikation noch einige Zeit vergehen. Auf jeden Fall zeigt diese Entwicklung, dass auf Basis der PVM einfach weitere Sprachen umgesetzt werden können und dass Standardunterstützung bei jBPM in greifbare Nähe rückt.

Fazit und Empfehlung

jBPM 4 macht einen sehr guten Eindruck. Es zeigt sich, dass viele Jahre Erfahrung eingeflossen sind und im Vorfeld Aspekte mehrfach diskutiert wurden. Leider ist der Funktionsumfang des 4.0-Releases noch eingeschränkt, was zumindest jBPM-3-Benutzer verwirren dürfte. Auch ist die Migration auf die neue Version keine leichte Aufgabe. Allerdings kann sich die Mühe lohnen, da die nun stabile API ein Projekt für die Zukunft rüsten sollte. Der Einsatz der Standardnotation BPMN, die sich mehr und mehr verbreitet, zeigt, dass jBPM Standards unterstützt.

Da wir bereits mit jBPM 3 viele Projekte begleitet haben, können wir guten Gewissens behaupten, dass die Engine ausgereift und stabil ist, und das in großen Projekten unter Last oder in Cluster-Umgebungen. Die in Java eingebettete, erweiterbare Process Engine ist für viele Anwendungsszenarien sicherlich eine gute Wahl. Dabei soll sie nicht BPEL ablösen, sondern adressiert ein anderes Problem. Sollten Sie also eine Prozessmaschine suchen, die in einer Java-Umgebung zu Hause ist oder in die eigene Anwendung eingebettet werden kann, werfen Sie doch einen Blick auf jBPM 4. ■

Bernd Rücker (bernd.ruecker@camunda.com) ist Berater, Trainer und Geschäftsführer bei der camunda services GmbH. Sein besonderes Interesse liegt im Bereich BPM & SOA sowie deren praktische Umsetzung. Er ist Autor eines EJB3-Buchs, zahlreicher Fachartikel, Sprecher auf Konferenzen aber auch Committer im JBoss jBPM-Projekt und Softwareentwickler. Zurzeit schreibt er an einem Buch zur BPMN und bloggt regelmäßig auf www.bpm-guide.de.

Falko Menge (falko.menge@camunda.com) ist Berater und Trainer bei der camunda services GmbH. Er ist auf SOA-, EAI- und BPM-Lösungen spezialisiert und ist Mitgründer/Committer einiger Open-Source-Projekte in diesem Bereich. Dazu gehören u. a. WSDL2XForms, InstantSVC und der im Artikel erwähnte Oryx Editor.

Links & Literatur

[1] JBoss jBPM: http://www.jboss.org/jbossjbpm/
[2] Nova Bonita: http://labs.jboss.com/jbossjbpm/
[3] Enhydra Shark: http://labs.jboss.com/jbossjbpm/
[4] BPMN: http://www.bpm-guide.de/bpmn/
[5] http://www.bpm-guide.de/2009/01/26/ein-erster-blick-auf-jbpm-4/
[6] http://www.bpm-guide.de/2009/07/08/ein-kleiner-jbpm-4-showcase/
[7] jBPM-Dokumentation: http://docs.jboss.com/jbpm/v4.0/userguide/html_single/
[8] JBoss PVM: http://jboss.org/jbossjbpm/pvm
[9] Oryx-Editor: http://www.oryx-editor.org/
[10] http://www.bpm-guide.de/2009/08/02/bauen-wir-uns-eine-bpmn-20-engine/