Geschäft(ige) Prozesse

jBPM – Ein Erfahrungsbericht

Bernd Rücker

Die Betriebswirtschaftslehre hat es schon seit Jahren erkannt: Geschäftliche Abläufe in einem Unternehmen folgen zu einem großen Prozentsatz beschreibbaren Prozessen, den so genannten Geschäftsprozessen. Diese Erkenntnis führte zum Business Process Management (BPM). jBPM ist eine kleine flexible Opensource-BPM-Engine, die auch kleinen Unternehmen effizientes BPM ermöglicht.

In den 90er Jahren fand auf Grund der Erkenntnis, dass sich geschäftliche Abläufe mit Geschäftsprozessen beschreiben lassen, eine Masse an Qualitätsmanagement-Projekten statt, welche als Ziel auch eine ausführliche Beschreibung der Prozesse hatten. Inzwischen reift jedoch überall die Erkenntnis, dass dokumentierte Geschäftsprozesse nur dann von wirklichem Nutzen sind, wenn sie auch so wie beschrieben IT-unterstützt ablaufen. Idealerweise stammen Dokumentation und Implementierung dabei aus dem gleichen Prozessmodell. Vielen Unternehmen fehlt heute aber genau hier die entscheidende Umsetzung in ihrer Software.

Dabei ist festzustellen, dass vor allem kleinen und mittleren Unternehmen Lösungen fehlen, die dieses Problem angehen. Dies liegt wohl nicht zuletzt an den hohen Preisen der entsprechenden Produkte der in diesem Bereich etablierten Hersteller. Aber das muss nicht sein, denn es gibt auch schon heute entsprechend gute Opensource-Unterstützung zur Umsetzung von BPM-Projekten.

Kurzüberblick BPM

Im BPM-Bereich gibt es zwei wichtige Standards:

- BPEL (Business Process Execution Language), eine Prozessbeschreibungssprache, die komplett auf Web-Services aufbaut, und
- XPDL (XML Process Definition Language).

Daneben existieren zwar noch weitere „Standards“, die in Zukunft aber vermutlich eine untergeordnete Rolle spielen werden.

Im Markt der Opensource-Workflow-Engines gibt es bisher nur wenige Produkte, die diese Standards beherrschen, dagegen sind proprietäre Lösungen hier sehr häufig. Dies hat uns dazu motiviert, in einer Evaluierung auch Produkte zu betrachten, welche BPEL erst auf der Roadmap stehen haben. Eine Übersicht über Opensource-BPM-Engines findet sich z. B. unter [Manageability].

In diesem Artikel wollen wir das Projekt jBPM [JBPM] betrachten, welches inzwischen in die JBoss Group [JBoss] aufgenommen wurde und sich einer großen und wachsenden Community im Internet erfreut. Unserer Einschätzung nach hat jBPM daher gute Chancen, im Java-Bereich zum De-facto-Standard zu werden (ähnlich wie der JBoss Application Server). Da jBPM auch eine sehr gute Roadmap [JBMPlan] vorlegt, sehen wir es als gute Wahl für ein neues Java-BPM-Projekt mit Open Source an, auch wenn es in der aktuellen Version BPEL noch nicht unterstützt.

Das Opensource-Projekt

Die hier beschriebenen Ideen rund um jBPM haben zu einem kleinen Opensource-Projekt geführt, welches es erleichtert, jBPM in typische Geschäftsanwendungen zu integrieren [CamundaTK]. Daher ist auch das vorgestellte Beispiel entsprechend dokumentiert und in unserem Wiki [CamundaWiki] zu finden.

Technische Grundlagen von jBPM

Was ist nun jBPM? jBPM ist eigentlich als klassische Workflow-Engine zu verstehen: Es wird ein Zustandsautomat beschrieben, der dann zur Laufzeit in der jBPM-Engine abläuft. Die Prozessbeschreibung muss dabei als XML vorliegen.

Da das Grundprinzip am einfachsten an einem Beispiel verständlich wird, ist in Abbildung 1 ein einfacher Beschaffungsprozess aus der in einem Kundenprojekt entwickelten Warenwirtschaft [CCS] grafisch dargestellt (für die betriebswirtschaftliche Seite der Prozessfindung sei auf [SchmSe04] verwiesen). Listing 1 zeigt Ausschnitte aus diesem Prozess als XML-Beschreibung, allerdings in einem eigenen, um Metainformationen angereicherten Format.

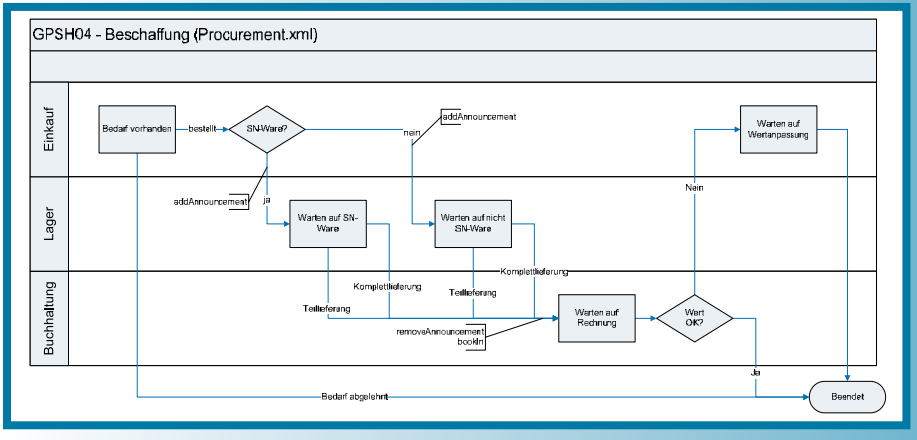

Abb. 1: Der Beispielprozess

```
<process-definition name="Procurement">
  <swimlane name="procurement" delegation="Einkauf" />

  <start-state name="ProcurementStart"
               swingForm="CreateProcurementProcessComponent">
    <assignment swimlane="procurement" />
    <transition name="createDemand" to="GoodsNeeded" />
  </start-state>

  <task-node name="GoodsNeeded" swingForm="GoodsOrderedComponent">
    <assignment swimlane="procurement" />
    <transition name="goodsOrdered" to="decideOrderComplete">
      <action>
        <businessService facade="StockJBpmFacade"
                         method="addAnnouncement">
          <param name="announcement">announcementValue</param>
          <result>article</result>
          <resultType>key</resultType>
        </businessService>
      </action>
    </transition>
    <transition name="demandDeclined" to="end" />
  </task-node>

  <decision name="decideOrderComplete">
    <delegation class="OrderCompleteDecisionHandler" />
    <transition name="complete"    to="decideIdentifiedGoods" />
    <transition name="incomplete"  to="splitProcurementProcess">
      <action>
        <delegation class="StartSecondGoodsNeededProcessAction" />
      </action>
    </transition>
  </decision>
[...]
</process-definition>
```

Listing 1: Unsere Prozessbeschreibung mit jBMP 3

Dieses XML kann sehr einfach per XSLT in das von jBPM benötigte jPDL [JPDL] transformiert werden. jPDL ist nun ebenfalls ein XML-Format, das unserer Prozessbeschreibung sehr ähnlich ist. Das jPDL kann, wie das gesamte Beispiel, in unserem Wiki [CamundaWikiBsp] nachgeschlagen werden. Momentan arbeiten wir daran, die XML-Dateien direkt aus dem vorliegenden grafischen Modell zu generieren.

Die wichtigen Prozesskonstrukte sind nun selbsterklärend: state (Zustand), descicion (Entscheidung) und transition (Transition zwischen Zuständen). Allein mit diesen Mitteln kann schon ein Großteil aller Prozesse beschrieben werden. Sind die Anforderungen jedoch komplizierter, kann jBPM auch mit fork, join sowie milestones umgehen.

Was kann nun die Engine mit diesen Prozessen anfangen? Zunächst einmal kann sie einzelne Prozess-Instanzen starten und wacht dann über deren Zustände und Zustandsübergänge (Transitionen). Dies ist komplizierter als es sich anhört, ein einfaches Codebeispiel soll dies verdeutlichen:

```
JbpmSession session  = JbpmSessionFactory.buildJbpmSessionFactory().
  openJbpmSession();

// Variablen für Prozess
Map variables = new HashMap();

// Prozess-Definition laden
ProcessDefinition pd = session.getGraphSession().
  findLatestProcessDefinition(„Procurement“);

// Prozess-Instanz „Einkauf“ starten
ProcessInstance pi = pd.createProcessInstance();

// Variablen zum Prozess hinzufügen
pi.getContextInstance().addVariables(variables);
```



```
// Transition „Bedarf abgelehnt“ ausführen
pi.getRootToken().signal(„demandDeclined“);

// Alle Aufgaben für „Bernd Rücker“
TaskInstance[] tasks = (TaskInstance[])session.getTaskMgmtSession().
findTaskInstances(„Bernd Rücker“).toArray(new TaskInstance[0]);
```

Ein weiteres Konstrukt der Prozessbeschreibungen sind die so genannten „swimlanes“. Sie regeln, welchem Geschäftsbereich (oder Mitarbeiter) ein Zustand zugeordnet wird. Dies ermöglicht es nun To-Do-Listen für einzelne Mitarbeiter zu erstellen. Die Pflege der Gruppen sowie der Zuordnung von Mitarbeitern zu Gruppen wird dabei von jBPM selbst nicht implementiert. Die Engine arbeitet vielmehr intern mit reinen Strings, die ganz allgemein einen „Actor“ beschreiben (dies kann dann sowohl eine Gruppe, eine Einzelperson oder auch ein Fremdsystem sein). In unserem Projekt haben wir dann einfach eine Komponente implementiert, die die To-Do-Liste für den einzelnen Anwender auflöst, indem sie alle Tasks für ihn und alle seine Gruppen von der Engine lädt. Die Komponente abstrahiert auch die konkrete Implementierung der Workflow-Engine (und wandelt beispielsweise die Tasks nach dem DataTransferObject-Pattern in ein technikneutrales „WorkItemDTO“).

Eine weitere wichtige Eigenschaft eines Prozesses ist, dass eine Prozess-Instanz beliebige Variablen beinhalten kann. Dabei kann in jBPM jedes Java-Objekt verwendet werden. Um in Java eine Abstraktion von jBPM zu erreichen und vor allem auch schneller entwickeln zu können, generieren wir aus der Prozessbeschreibung Java-Klassen, die ein Prozessmodell bilden und unter anderem mit JGoodies DataBinding [JGoodies] an die Prozessvariablen gebundene Oberflächenfelder bereitstellen.

Wie kann nun der Prozess auf Geschäftslogik zugreifen? Hier haben die jBPM-Entwickler auf Flexibilität gesetzt und lassen dem Anwender alle Freiheiten, einzige Bedingung: Aktionen müssen als Java-Code implementiert werden. Dann können an verschiedene Prozesskonstrukte Actions „angehängt“ werden, die bei entsprechenden Zustandsübergängen ausgeführt werden.

Dies klingt zwar nicht nach Flexibilität, schreibt man sich jedoch sein eigenes kleines Framework, kann man wirklich so gut wie alles machen. In unserem Projekt haben wir beispielsweise den Zugriff auf die als Services implementierte Geschäftslogik so gekapselt, dass wir sie in der Prozessbeschreibung einfach aufrufen konnten (siehe auch Listing 1 und unsere Wiki). Ein ähnliches Vorgehen ist dann auch für Web-Services, andere Prozesse, Fremdsysteme oder ähnliches möglich.

Ablauf-Umgebung

jBPM kann sowohl in jedem J2EE-konformen Applikationsserver als auch in einem stand-alone Programm betrieben werden. Der Betrieb in einem Applikationsserver bietet dabei den großen Vorteil, dass sowohl die BPM-Engine als auch die J2EE-Services evtl. in einem Transaktionskontext ausgeführt werden können, was ein Rollback bei Fehlern vereinfacht.

Alternativ können hierzu natürlich auch Undo-Mechanismen als Teil der Prozesse eingesetzt werden. Dieses Konzept, wie es vor allem in BPEL verfolgt wird, ist zwar auch in jBPM möglich, jedoch etwas komplexer.

Als Persistenzmechanismus setzt jBPM auf Hibernate [Hibernate], was es einfach ermöglicht auch einmal selbst Querys auf das Objektmodell von jBPM loszulassen, wenn die Bordmittel nicht mehr genügen. In unserem Fall haben wir z. B. Querys implementiert, die Prozess-Instanzen mit speziellen Variablenwerten heraussuchen (z. B. alle Beschaffungsprozesse für einen bestimmten Artikel).

Insgesamt bietet jBPM so die Möglichkeit, in jeglicher Architektur und somit auch in vielen denkbaren Java-Projekten eingesetzt zu werden, mit oder ohne Applikationsserver. Eine Flexibilität, die schwergewichtige kommerzielle Geschwister häufig vermissen lassen.

Vorteile der BPM-Engine

Arbeitet man mit einer BPM-Engine, kann man schnell erkennen, wo der Vorteil dieser prozessorientierten Entwicklung liegt. Da der Prozess die zu Grunde liegenden Services steuert, ist die Struktur der Anwendung einfach verständlich und der Kontrollfluss leicht nachvollziehbar. Außerdem können Geschäftsprozesse sehr leicht angepasst und Änderungen sogar im laufenden System deployed werden. Eine Versionierung verhindert dabei Chaos mit bereits laufenden Prozessen. Dadurch bleibt die Anwendung an sich wartbar, denn schließlich ändern sich an die Prozesse gestellte Anforderungen meist deutlich rascher als die Anforderungen an die Services.

Ein weiterer dicker Pluspunkt ist die Wiederverwendbarkeit der Geschäfts-Services, denn diese können für sich sehr unabhängig implementiert werden, da das Zusammenspiel lediglich durch den Prozess bestimmt wird. So kann man aus verschiedenen zur Verfügung stehenden Services die verschiedensten Prozesse „zusammenbauen“ (man spricht hierbei von „Orchestrierung“).

BPM im Kontext von SOA

Das eben beschriebene Vorgehen hängt eng mit service-orientierter Architektur (SOA) zusammen, wobei immer klar sein sollte, dass eine SOA nicht unbedingt etwas mit Web-Services zu tun hat! Ich möchte hier auf die meiner Ansicht nach gute Definition der Karlsruher Versicherung zurückgreifen: „Eine Service-Orientierte Architektur ist eine Software-Architektur, die idealerweise für alle Anwendungssysteme eines Unternehmens gilt und auf den Prinzipen der Service-Orientierung aufbaut.“ [SOADef] Diese Prinzipen sind

- Kapselung von Business-Funktionalität,
- Entkopplung der Schnittstelle,
- Technologie-Neutralität,
- Services sind Bausteine für Prozesse und Anwendungen.

Somit kann man in diesem Kontext sagen, dass die BPM-Engine als zentrale Instanz einzelne Services zu einem funktionierenden Geschäftsprozess verbindet. Diese Prozesse können dann natürlich auch als ganzes wieder als Services bereitgestellt werden.

Oberfläche für Geschäftsprozesse

Wie integriere ich die BPM-Engine nun in die Oberfläche der Anwendung? In unserem Framework haben wir hauptsächlich eine Worklist, also eine To-Do-Liste für die einzelnen Mitarbeiter, welche mit Daten aus jBPM gefüllt wird, implementiert.

Diese Worklist kann sich dann die Java-Prozessmodelle für laufende oder neue Prozesse über einen Resolver besorgen. Das Java-Prozessmodell stellt neben den Eingabe-Feldern auch Actions bereit, die dann die entsprechenden Transitionen aufrufen. Dabei sind immer nur die in dem momentanen Status des Prozesses ausführbaren Actions aktiv.

Die Worklist kann sich neben dem Modell auch eine in der Prozessbeschreibung konfigurierte Oberflächenkomponente für den aktuellen Zustand eines Prozesses holen und mit der Prozessmodell-Instanz initialisieren. Die Oberflächenkomponente kann dann völlig flexibel implementiert werden und die vom Prozessmodell bereitgestellten Felder, Variablen oder Actions benutzen. Einzige Einschränkung ist bei uns die Implementierung einer abstrakten Komponente des eingesetzten GUI-Frameworks (als GUI-Framework wird eine an Genuine [Genuine] angelehnte Implementierung verwendet). Durch eine leichte Anpassung des Resolvers kann die Implementierung jedoch leicht an die eigene Umgebung angepasst werden.

Aus Platzgründen sei für eine genauere Betrachtung der Oberfläche auf unser Wiki verwiesen.

Zukunft

Zu der Lösung ist zu sagen, dass sie in unserem Projekt hervorragend funktioniert hat. Zwar ist immer noch viel Verbesserungspotential vorhanden, die Generatoren und Framework-Teile werden aber auch ständig weiterentwickelt. Und letztendlich profitiert jBPM hierbei von den weitgehend bekannten Vorteilen eines aktiven Opensource-Projektes.

Einige Punkte haben wir hier noch gar nicht angesprochen, so planen wir momentan z. B. eine Konsole (evtl. auch JMX), um Prozessinstanzen zu überwachen bzw. manuell in deren Ablauf eingreifen zu können. Auch sind Auswertungen von Prozesslaufzeiten, Hot-Spots u. ä. im Sinne der Geschäftsprozessorientierung notwendig.

Aber auch jBPM selbst ist nicht untätig, so steht bereits eine Integration der standardisierten Prozess-Beschreibungs-Sprache BPEL auf der nahen Roadmap und auch sonst ist von diesem Projekt unserer Ansicht nach noch viel zu erwarten.

Java & BPM

Betrachtet man das Thema „BPM mit Java“ so kann man feststellen, dass es momentan einen relativ gesunden Aufschwung erlebt. Vielleicht brauchte man erst die viel zitierten service-orientierten Architekturen, um das große Potential von BPM zu erkennen oder aber es fehlte einfach an guten freien Produkten in diesem Bereich. Wir wissen auch nicht, warum sich diese Technik erst jetzt durchsetzt, sind uns aber ziemlich sicher, dass sie über kurz oder lang die Softwareentwicklung zusammen mit SOA und MDA deutlich voranbringen wird.

Es ist auch eine Marktbereinigung der jetzt doch unzähligen Opensource-Workflow-Engines zu erwarten, weshalb jBPM mit seiner Ehe mit JBoss einen guten Zug gemacht hat. Auch im kommerziellen Bereich, wo zugegebenermaßen sehr ausgereifte Produkte verfügbar sind, ist Bewegung zu erwarten. Denn durch die neuen Mitstreiter müssen sich auch die bisher recht konkurrenzlosen großen Hersteller wärmer anziehen.

Zusammenfassung

Der Einsatz einer BPM-Engine lohnt sich in einer typischen Geschäftsanwendung auf jeden Fall. Idealerweise orchestriert der Prozess dann lediglich vorhandene Services zu unterschiedlichen Geschäftsprozessen. Im Prozess können eigene Services, Fremdkomponenten oder auch Web-Services benutzt werden. Dabei können bei geeignetem Vorgehen betriebswirtschaftliche Prozessdokumentation und ablauffähige Prozessbeschreibung aus einer Quelle kommen.

Mit jBPM bekommt man eine kleine flexible Opensource-BPM-Engine an die Hand, die auch für große Projekte einsetzbar ist. Dabei muss man eben im Vergleich zu guten kommerziellen Produkten etwas mehr Handarbeit leisten und eigene kleine Erweiterungen oder Frameworks schreiben, was wir mit unserem Opensource-Projekt einmal angeregt haben. Insgesamt ist hier in Zukunft sicher immer mehr Unterstützung durch die Open Source Community zu erwarten.

Literatur und Links

[CamundaTK] Camunda Toolkit for developing jBPM applications, `tk-jbpm.camunda.org`

[CamundaWiki] `http://www.camunda.com/wiki/index.php/Camunda_Toolkit_for_jBPM`

[CamundaWikiBsp] `http://www.camunda.com/wiki/index.php/Beispiel_Einkaufsprozess`

[CCS] Computation Combined Services, `www.camunda.com/ccs/`

[Genuine] `genuine.sourceforge.net/`

[Hibernate] `www.hibernate.org`

[JBoss] JBoss-Startseite, `www.jboss.org`

[JBPM] JBoss jBPM, `www.jbpm.org`

[JBPMPlan] JBoss, Inc. JIRA, `jira.jboss.com/jira/browse/JBPM`

[JGoodies] JGoodies Data-Binding-Framework, `binding.dev.java.net/`

[JPDL] jBPM Process Definition Language, `http://www.jbpm.org/jpdl.html`

[Manageability] Blog zu Opensource-Workflow-Engines in Java, `www.manageability.org/blog/stuff/workflow_in_java/view`

[SchmSe04] H. J. Schmelzer, W. Sesselmann, Geschäftsprozeßmanagement in der Praxis, Hanser Wirtschaft 2004

[SOADef] Definition der Karlsruher Versicherung des Begriffs service-orientierte Architektur (SOA), `www.eec2005.de/speaker/EB 04 Muench_Karlsruher.pdf`

Bernd Rücker ist Softwarearchitekt und Geschäftsführer bei der camunda GmbH, einem Unternehmen, das seinen Schwerpunkt auf die Entwicklung und den Betrieb von Geschäftssoftware in J2EE legt und dabei vornehmlich BPM und Business Rule Engines in Verbindung mit service-orientierten Architekturen einsetzt. E-Mail: bernd.ruecker@camunda.com.