Prozesse außer Kontrolle?

Mithilfe der quelloffenen Prozessverwaltung JBoss jBPM lassen sich Abläufe im Unternehmen modellieren und deren Steuerung automatisieren. In ernsthaften Projekten im größeren Unternehmenskontext ist jedoch auch die Verwaltung laufender Prozessinstanzen eine wichtige Funktionalität, die in jBPM aktuell leider kaum adressiert wird. PIS (Prozessinformationssystem) ist daher eine eigene Prozessverwaltung, mit der die jBPM-Prozesse auch zur Laufzeit im Zaum gehalten werden können.

von Bernd Rücker und Dr. Jörg Moldenhauer

Spricht man heute von einer SOA, so werden oft im gleichen Atemzug die Begriffe „Enterprise Service Bus“ (ESB) und „Business Process Management“ (BPM) genannt. In der Tat sind beide wichtige Komponenten zum Aufbau einer SOA in einem Unternehmen. Über den ESB können auf bestimmte Aufgaben zugeschnittene Services flexibel zur Verfügung gestellt und über BPM orchestriert werden, d. h. entsprechend der Geschäftsprozesse zusammengeschaltet und angesteuert werden. Auch innerhalb der frei verfügbaren Produktfamilie von JBoss werden diese zwei Komponenten einer SOA in Form von JBossESB [1] und jBPM [2] angeboten, eine Einführung dazu ist unter [3] zu finden.

Generell kann man JBossESB und jBPM als ein Gespann betrachten, mit dem sich eine SOA im Unternehmen etablieren lässt. Aus diesem Grund hat sich auch die 1&1 Internet AG für deren Einsatz entschieden. Beim täglichen Betrieb wird man sich aber schnell bewusst, dass allein die Entwicklung von Services und deren Verschaltung über Prozesse nicht ausreichend ist, um Abläufe im Unternehmen wie Vertragsanlage, Einrichtung von Diensten oder Tarifumstellung damit realisieren zu können. Viel mehr braucht es Unterstützung für Monitoring, Betrieb und Governance. In diesem Artikel wenden wir uns daher jBPM zu, zeigen die Möglichkeiten aber auch die Grenzen auf und stellen mit PIS eine Erweiterung vor, die benötigte Features für die tägliche Arbeit nachliefert.

Grundlagen zu JBoss jBPM

JBoss jBPM ist eine leichtgewichtige Prozessmaschine, die zum Zeitpunkt des Verfassens dieses Artikels in der Version 3.2.6 vorliegt (Version 3.3.x ist dabei irreführenderweise älter als 3.2.6). Zum Zeitpunkt des Drucks wird bereits Version 4 veröffentlicht sein. Die neue Version ist eine deutlich verbesserte Neuimplementierung der Engine, die im Bereich der Modellierung die standardisierte Notation BPMN nutzt. Ein detaillierter Artikel zu jBPM 4 folgt in einer der folgenden Ausgaben des Java Magazins.

jBPM ist rein POJO-basiert implementiert, Persistenz wird durch Hibernate umgesetzt. Das ermöglicht den Einsatz ohne Persistenz oder mit allen von Hibernate unterstützen Datenbanken, mit oder ohne Application-Server. In den meisten größeren Anwendungen jedoch, so auch bei 1&1, wird jBPM im JBoss-Application-Server betrieben und lebt in einer transaktionalen Umgebung.

Die Hauptaufgabe der Engine ist die Abbildung eines Zustandsautomaten mit Wartezuständen, in denen die Engine den jeweils aktuellen Prozesszustand persistiert. Dabei gibt es, wie in Abbildung 1 gezeigt, Nodes und Transitionen, wobei letztere zwei Nodes miteinander verbinden. Dieses Grundprinzip ermöglicht es bereits, Prozesse als Grafen zu beschreiben. Es gibt unterschiedliche Arten von Nodes, die das Prozessverhalten beeinflussen können. Die Prozessausführung wird durch ein Token-Objekt gesteuert, das durch den Prozessgrafen „wandert“ – es entspricht also einer Prozessinstanz (Hinweis: In jBPM 4 wird das Token durch den Begriff „Execution“ ersetzt). Während des Prozessdurchlaufs kann die Engine Aktionen anstoßen, z. B. das Versenden einer E-Mail oder den Aufruf eines Web Service. Manche Knotentypen fungieren dabei als Wartezustände, in denen das Token stehen bleibt, bevor der Zustand in die Datenbank geschrieben wird.

Wozu eine Prozessverwaltung?

jBPM eignet sich gut zur Orchestrierung von Prozessen, auch bei Verwendung des JBossESB. Allerdings bietet die Open Source Process Engine zurzeit nur ein sehr rudimentäres Tooling in Form einer Webkonsole an, um Prozess-Monitoring oder das aktive Eingreifen in den Prozessablauf durch den Betrieb zu gewährleisten. Für einen ernsthaften Betrieb im unternehmensweiten Kontext mit großen Mengen von Prozessinstanzen ist das bei Weitem nicht ausreichend. Bei 1&1 war

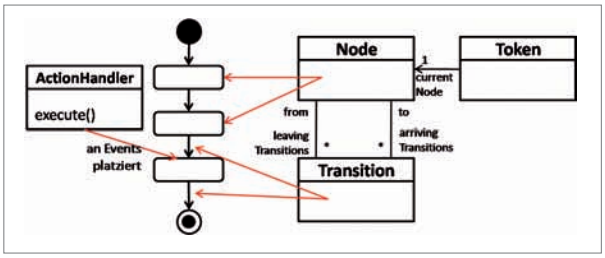

Abb. 1: Interner Aufbau von jBPM

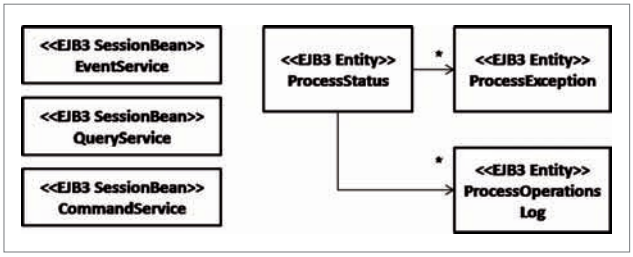

Abb. 2: PIS-Entitäten und Session Beans

dabei auch zu berücksichtigen, dass in der SOA-Landschaft verschiedene jBPM-Instanzen betrieben werden (also jBPM mehrfach „installiert" ist) und eine zentrale Sicht auf die Prozesse aller Engines essenziell ist. Dabei ist es perspektivisch auch möglich, dass verschiedene Process Engines zum Einsatz kommen, ein absehbares Beispiel ist der Parallelbetrieb von jBPM in der Version 3 und 4. Ebenfalls denkbar wäre, dass auch Altsysteme ohne Process Engine Zustände an die Prozessverwaltung liefern, um an zentraler Stelle alle Prozessinformationen zusammenlaufen zu lassen.

Was aktuell ebenfalls in jBPM fehlt, ist der einfache Einstieg über fachliche Informationen. Beispielsweise möchte man auf einen Blick alle laufenden Prozessinstanzen eines Kunden und deren jeweiligen Zustand einsehen können. Aus diesem Grund ist die Entscheidung gefallen, PIS als eigene Anwendung zu entwickeln und im Rahmen der 1&1-SOA-Plattform einzusetzen.

Implementierung der Prozessverwaltung

Die Grundlage der zentralen Prozessverwaltung PIS ist eine eigens entwickelte EJB-3-Komponente, die den Zustand aller laufenden Prozessinstanzen speichert. Den Kern bilden drei Entitäten, die Informationen zum aktuellen Prozesszustand, den Prozessvariablen (beides in *ProcessStatus*), aufgetretenen Fehlern *(ProcessException)* sowie Eingriffe des Betriebs auf Prozessinstanzen *(ProcessOperationsLog)* speichern. Das Design wurde bewusst einfach gehalten, um unkompliziert Abfragen auf diese Entitäten zu ermöglichen. Zum Zugriff auf die Komponente werden EJB-3-Session-Beans bereitgestellt, wobei drei Services unterschieden werden: das Melden von Prozessinformationen *(EventService)*, Abfragen von Prozessstatus *(QueryService)* sowie Ausführen von Operationen zur Beeinflussung des Prozessablaufs *(CommandService)*. Das ist in Abbildung 2 dargestellt.

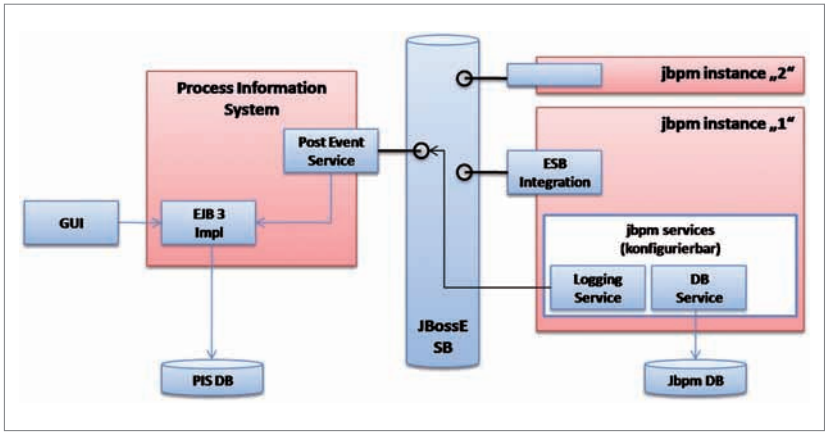

Abb. 3: Funktionsweise von PIS in der Übersicht

Damit die zentrale Prozessverwaltung alle notwendigen Informationen über Prozesszustände kennt, muss sie von den laufenden jBPM-Instanzen stetig über Veränderungen benachrichtigt werden. Das ist über Events realisiert, die jede jBPM-Instanz erzeugen und über den ESB an einen eigenen Service schicken. Dieser nimmt die Events als XML an, transformiert sie in das notwendige Java-Objekt und übergibt dieses an die Session Bean. Der gesamte Ablauf ist in Abbildung 3 visualisiert.

Um die Lösung praxistauglich zu halten, war eine wichtige Anforderung, dass die Event-Erzeugung in jBPM generisch erfolgt. Anpassungen an den einzelnen Prozessdefinitionen werden somit vermieden, was nicht nur Arbeit erspart, sondern vor allem Fehlerquellen eliminiert. PIS erweitert zu diesem Zweck den in jBPM existierenden Logging-Service. Dieser ist dafür zuständig, Audit Logs während der Prozessausführung in der Datenbank abzulegen, z. B. wann welcher Prozessknoten durchlaufen wurde. Entsprechend können diese Informationen verwendet werden, um Events zu erzeugen und an PIS zu versenden.

Leider ist dieser Mechanismus nicht ganz ausreichend. Wird z. B. eine Exception in einem Prozess ausgelöst, so wird der *LoggingService* nicht angesprochen, da eine Exception direkt zu einem Zurückrollen der aktuellen Transaktion führt. Aus diesem Grund muss eine weitere Erweiterung am „Einstiegspunkt" zu jBPM vorgenommen werden. In unserem Fall ist das der jBPM-*CommandService*. Auftretende Exceptions werden dadurch ebenfalls an PIS gemeldet. Das passiert übrigens in einer eigenen Transaktion, sodass PIS auch im Fehlerfall erfolgreich über den Fehler informiert wird. Ein Beispiel wird in Abbildung 4 gezeigt. In diesem Fall ist es ein Prozess mit parallelen Handlungssträngen, im jBPM-Vokabular ein *fork*. Da die einzelnen Prozesspfade unabhängig voneinander ausgeführt werden, muss die Prozessverwaltung über alle gleichzeitig erreichten Wartezustände Bescheid wissen. Tabelle 1 veranschaulicht die Daten von PIS zu dem in der Abbildung illustrierten Zustand. Der Prozess hat das Fork durchlaufen, das linke Token wurde bereits signalisiert und ist im Join angekommen. Der rechte Pfad ist sowieso ohne Stopp zum Join gelaufen. Wir warten also aktuell auf den mittleren Pfad im Wartezustand *in-fork-2*.

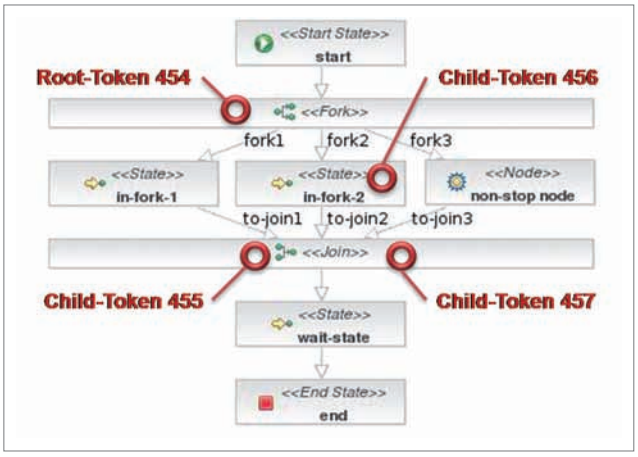

Abb. 4: Beispielprozess

An die Prozessverwaltung werden übrigens nur Informationen über erreichte Wartezustände gesendet. Zwischenzustände sind nicht von Belang, da die Prozessmaschine dort nicht anhält. PIS speichert ebenfalls keine Historie, sondern immer nur den aktuellen Zustand der Prozessinstanzen.

Aktives Eingreifen in Prozesse

Neben den reinen Statusinformationen bietet PIS auch Möglichkeiten zum Eingreifen in den Prozessablauf. Dabei wurde vor allem auf ein massentaugliches Verfahren Wert gelegt, sodass problemlos auf große Mengen an Prozessinstanzen Einfluss genommen werden kann.

Zum Zugriff auf Funktionalitäten der Process Engine unterstützt jBPM selbst bereits das Command-Pattern. Damit können Commands selbst implementiert und über den mitgebrachten *CommandService* ausgeführt werden. Glücklicherweise werden dabei zahlreiche Commands bereits mitgeliefert, z. B. zum Abbrechen einer Prozessinstanz (*CancelProcessInstanceCommand*), sodass nur spezifische Funktionalität als eigenes Command implementiert werden muss. Ein *CommandService* als EJB2.1-Session-Bean liefert jBPM ebenfalls bereits mit.

Die Prozessverwaltung erweitert dieses Verhalten insofern, dass sie Commands zentral entgegennimmt und dann an die jeweiligen jBPM-Instanzen weiter verteilt. Der Aufrufer braucht sich also nicht darum zu kümmern, auf welcher Instanz der angesprochene Prozess läuft. In diesem Zusammenhang können mehrere Aktionen in eine Transaktion verkettet werden, so genannte *CommandChains*.

Des Weiteren gibt es die Möglichkeit, Batch Commands auszuführen, also z. B. das Verändern einer Prozessvariablen in allen laufenden Prozessinstanzen

eines bestimmten Prozesses, zu einem bestimmten Kunden oder zu einem Produkt. Das Batch kann dank der zentralen Prozessverwaltung auch Prozessinstanzen unterschiedlicher jBPM-Instanzen umfassen. Beim Ausführen des Batch kann spezifiziert werden, ob alle Aktionen in einer Transaktion synchron ausgeführt werden oder eine asynchrone Ausführung via JMS erfolgt. Letzteres führt nicht zum Zurückrollen der gesamten Transaktion, wenn nur ein Command fehlschlägt, und verhindert bei großen Batches Timeouts der Transaktion. Darüber hinaus können Fehler über vorhandene JMS-Tools erkannt und Commands neu ausgeführt werden. Neben der Möglichkeit, eigene Commands in Java zu implementieren, was ein Redeploy der Prozessverwaltung erfordert, wurde noch „ultimative" Flexibilität eingebaut: Es gibt ein Command, das Groovy-Skripte ausführt. Die Skripte können dann zur Laufzeit mitgegeben werden.

Grau ist alle Theorie, weswegen wir an dieser Stelle ein Beispiel für ein Szenario zum steuernden Eingreifen in den Prozessablauf geben möchten: Eine Prozessdefinition ist für einen aufgetretenen Problemfall unzureichend. Ein übliches Szenario bei der Prozessautomatisierung ist, dass Spezialfälle nicht modelliert wurden oder vorher nicht relevant waren. Nun wird ein verbesserter Prozess deployt. Je nachdem, wie komplex die Veränderung der Prozessdefinition aussieht, könnte man eventuell die laufende Prozessinstanz auf die neue Prozessversion updaten, wofür ein eigenes Command zur Verfügung steht. Ist die Veränderung zu komplex, so ist es aber vielleicht geeigneter, die bestehende Prozessinstanz abzubrechen, eine neue Prozessinstanz zu starten und sie in einen bestimmten Wartezustand zu versetzen. Abbildung 5 visualisiert das Vorgehen. Die notwendige Funktionalität kann anhand von mehreren Commands in einer *CommandChain* umgesetzt werden. Den notwendigen Code für das Beispiel zeigt Listing 1. Eigentlich gibt es für diesen Anwendungsfall sogar ein eigenes *ReplaceProcessInstanceCommand*, jedoch ließe sich damit die *CommandChain* nicht klar verdeutlichen.

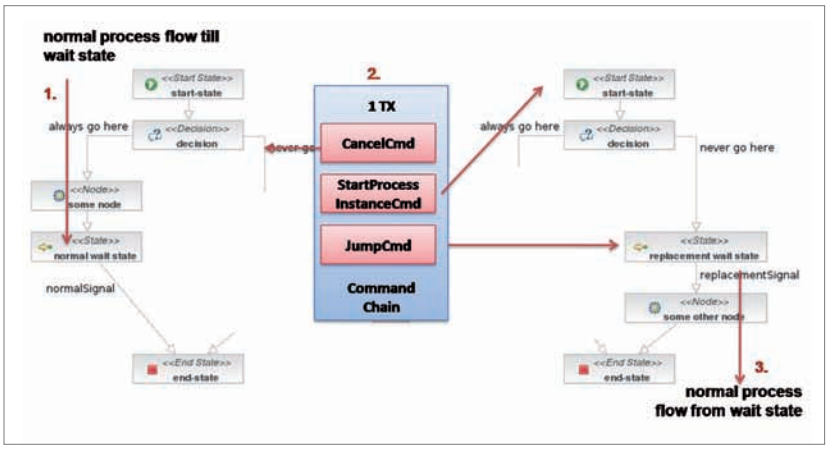

Abb. 5: Beispiel einer CommandChain: Eine Prozessinstanz wird durch eine neue ersetzt

Gibt es eine größere Menge von Prozessinstanzen mit dem gleichen Schicksal, kann diese Aktion auch als Batch auf einer beliebig großen Menge von Prozessinstanzen ausgeführt werden. Es sei darauf hingewiesen, dass wenn eine dieser Prozessinstanzen in der Zwischenzeit weiterläuft, die *CommandChain* für diese Instanz einen Fehler verursachen wird, da der erwartete Zustand nicht dem im Command übergebenen entspricht. Damit landet das Command in einer Fehler-Queue, in der es ein Administrator gesondert bearbeiten kann. Somit ist dieses Vorgehen auch für Systeme mit großer Last und einer Vielzahl paralleler Prozessinstanzen geeignet.

tokenId	processInstanceId	parentTokenId	Status	State	Timestamp
454	232	0	WAITING	the fork	2009-06-15 10:15:31
455	232	454	ENDED	the join	2009-06-15 10:16:55
456	232	454	RUNNING	in-fork-2	2009-06-15 10:15:31
457	232	454	ENDED	the join	2009-06-15 10:15:31

Tabelle 1: Daten in PIS

Wie bereits angedeutet, wollen wir an dieser Stelle noch auf die Zukunftssicherheit und Erweiterbarkeit von PIS eingehen, da aktuell bereits jBPM in der Version 4 vor der Tür steht. Der Mechanismus, den aktuellen Zustand an PIS zu melden, kann natürlich auch in der neuen Version wieder eingebaut werden. Prinzipiell könnte das auch in ganz anderen Prozessmaschinen integriert werden. Auch das Eingreifen im laufenden Betrieb kann einfach auf die neue Version umgestellt werden, da jBPM 4 wieder mit Commands arbeitet. Das geschieht hier sogar noch deutlich konsequenter als bei jBPM 3. Es ist daher auch denkbar, in der Prozessverwaltung eine Übersetzung zwischen den Welten zu

Listing 1: Beispiel einer CommandChain

```
PisCommandChainBuilder builser =
                    PisCommandChainBuilder.create(jbpmInstanceId)
PisCommand cmd = builder
  .expectedStateForToken(tokenId, "normal wait state")
  .addCommand(new CancelProcessInstanceCommand()
    .processInstanceId(processInstanceId))
  .addCommand(new StartProcessInstanceCommand()
    .processDefinitionName("OpsCancelWithReplacementProcess"))
  .addCommand(new JumpCommand()
    .jumpToNode("replacement wait state")
    .replacementProcessDefinitionName
                              ("OpsCancelWithReplacementProcess"))
  .addResultToParameterMappingForLastCommand
                                 (-1, "result.id", "processInstanceId")
  .addCommand(new SignalCommand()
    .transitionName("replacementSignal"))
  .addResultToParameterMappingForLastCommand
                                 (-1, "result.id", "processInstanceId")
  .build();
getOperationsFacade().executeCommandSync(cmd);
```

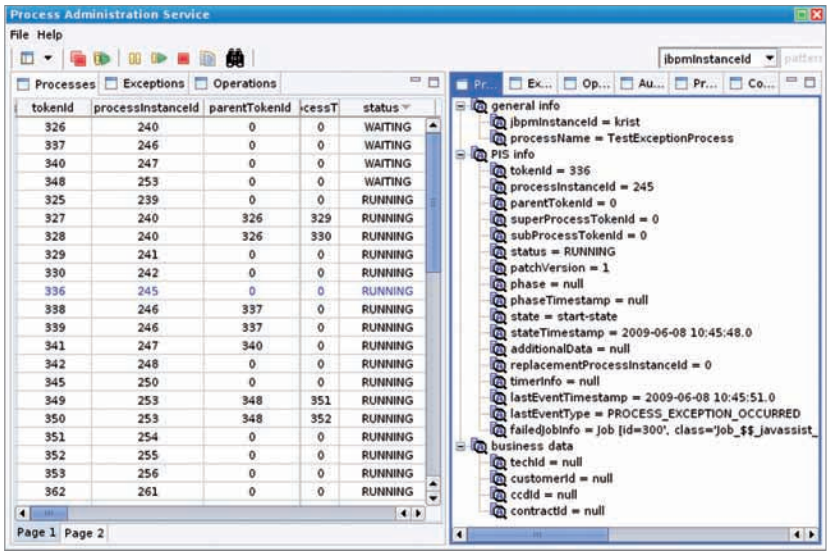

Abb. 6: Der Blick auf PIS nach dem Starten als RAP-Anwendung

implementieren, also falls Commands an Instanzen von jBPM 3 und jBPM 4 geschickt werden müssen. Da die Parameter größtenteils gleich sind, müssten sie im besten Fall lediglich in ein anderes Command-Objekt verpackt werden.

Die grafische Benutzerschnittstelle

Aus der praktischen Erfahrung beim Umgang mit Geschäftsprozessen heraus hat sich sehr früh als Anforderung ergeben, dass die Prozessverwaltung nicht nur von Entwicklern, sondern auch von operativen Einheiten im Unternehmen eingesetzt werden können muss. Ohne ein geeignetes Frontend für PIS wäre kein Zugewinn erreicht worden. Zielsetzung war, über das Frontend Informationen zu laufenden Prozessinstanzen abrufen zu können, eine Möglichkeit zur Suche zu haben und über die beschriebenen Commands aktiv in die Prozesse eingreifen zu können.

Als Grundlage für die Implementierung des Frontends dient RCP bzw. RAP. Basierend auf diesen Technologien lassen sich elegant Standalone-Anwendung und Eclipse-Plug-ins, aber auch Webanwendungen über nahezu identischen Code bauen. Weitere Informationen zu RAP und das so genannte Single Sourcing sind unter [4] zu finden. Durch die verschiedenen Laufzeitumgebungen des Frontends werden wir den unterschiedlichen Anforderungen der Benutzer gerecht. So wird vielleicht der Entwickler ein in seine IDE eingebettetes Plug-in bevorzugen, der Mitarbeiter im Operationsteam die eigenständige Anwendung für den häufigen Gebrauch und ein Administrator, der im Fehlerfall schnell eine Übersicht über die Prozesse braucht, den einfachen Blick auf die Anwendung im Browser.

In Abbildung 6 ist ein Screenshot der Webanwendung im Browser zu sehen. Die RCP-Anwendung sieht analog aus. Das Fenster von PIS besteht im Wesentlichen aus zwei Views, um einerseits eine Übersicht über alle registrierten Prozesse und deren Instanzen (linke View) und eine Detailansicht für eine ausgewählte Prozessinstanz (rechte View) zu zeigen. Um das Navigieren in der langen Liste von Prozessinstanzen zu erleichtern, kann über ein Paging-Mechanismus darin geblättert werden. Von der tabellarischen Sicht kann zudem auch auf eine Baumansicht gewechselt werden, die z. B. eine Gruppierung nach Prozesszuständen oder Versionen von Prozessinstanzen erlaubt. Neben der Prozessliste kann auch mithilfe der Tabs oberhalb der Tabelle auf eine Übersicht mit allen aufgetretenen Fehlermeldungen und ein Protokoll mit allen durchgeführten Commands umgeschaltet werden. In der Detailansicht sind nochmals alle Informationen zu einer ausgewählten Prozessinstanz nach allgemeinen Informationen aus jBPM, technischen Informationen aus der Prozessverwaltung, aber auch ausgewählten fachlichen Informationen wie Kunden- und Vertragsnummer gegliedert. Darüber hinaus bieten die Tabs oberhalb der Detailansicht die Möglichkeit, auf Log-Informationen (Exceptions, durchgeführte Commands, Audit) für die jeweilige Prozessinstanz zuzugreifen oder die aktuelle Position des Token im Prozessbild, wie man das von der jBPM Console kennt, zu visualisieren. Alternativ zum Prozessbild lässt sich auch die zugrunde liegende XML-Beschreibung anzeigen.

Für operative Eingriffe kann eine Menge von Prozessinstanzen in der Liste selektiert werden und danach Commands, z. B. Pause, Suspend oder Resume, ausgeführt werden. Die Commands werden über die Toolbar oder das Menü aufgerufen. Ein Suspend auf allen Instanzen zu einem bestimmten Prozess („Not-Aus“) wird ebenfalls als Command unterstützt. Dieses kann auch über ein entsprechendes, gesammeltes Resume wieder zurückgenommen werden. In der Toolbar sind noch weitere Buttons zu finden. Unter anderem wird dadurch z. B. die Suche nach Prozessinstanzen ermöglicht. Hierzu wird ein eigener Suchdialog geöffnet, in dem komplexe Suchanfragen über logische Verknüpfungen erzeugt und durchgeführt werden können. Eine schnelle Suche lässt sich aber auch über das Suchfeld in der Toolbar erzielen, das eine Filterung der Prozessliste anhand eines Suchmusters für einen einzelnen Parameter bewirkt.

Besonderes Augenmerk soll an dieser Stelle noch auf die Anwendung von Batch Commands und Command Chains gerichtet werden, da erst durch sie komplexe Eingriffe an den Prozessen durchgeführt werden können. Mithilfe des *Command*-Buttons in der Toolbar kann eine erweiterte Detailansicht aktiviert werden, in der auch ein Command-Editor zur Verfügung steht. Im Command-Editor lassen sich nochmals alle existierenden Commands einzeln auswählen oder diese zu einer *CommandChain* kombinieren. Als besonderes Command ist das Groovy Command enthalten, über das ganze Groovy-Skripte ausgeführt werden können. Spätestens hiermit stehen dem erfahrenen Operator alle Möglichkeiten zum Eingriff in aktuelle Abläufe zur Verfügung. Im gezeigten Beispiel wird zur Demonstration nach dem Suspend der

jeweiligen Prozessinstanz das einfache Groovy-Skript ausgeführt, das einige Informationen zu der jeweiligen Prozessinstanz ausliest und sie zur Ausgabe in Form einer Nachricht verwendet. Danach wird die Prozessinstanz über ein Resume wieder fortgesetzt:

```
println "Hi, this is a message from token " + token.
    getId() + " in node "+ token.getNode().getName()
return jbpmContext.getProcessInstance
                        (processInstance.getId())
```

Der Einsatz und die Zukunft von PIS

PIS wurde bisher nur in kleinem Rahmen mit einigen exemplarischen jBPM-Prozessen verwendet. In den kommenden Monaten wird der Einsatz aber deutlich zunehmen, da PIS lediglich eine von mehreren Governance-Komponenten einer vollständigen SOA-Suite darstellt, die bei der 1&1 Internet AG gerade aufgebaut und etabliert wird. Bekanntermaßen geht mit der konsequenten Einführung einer SOA im Unternehmen nicht nur die Umstellung von technologischen Komponenten einher, sondern auch ein Umdenken beim täglichen Arbeiten. Das wird auch beim Einsatz von PIS der Fall sein. Bisher wurden operative Eingriffe in die Geschäftsprozesse im Unternehmen direkt mit SQL auf Datenbanktabellen, die die Prozessdaten enthalten, vorgenommen. Jetzt erfolgt dies über ein Werkzeug, das einerseits Erleichterung und Unterstützung bei der täglichen Arbeit mit jBPM offeriert, aber auch die gleiche Mächtigkeit und Flexibilität bieten muss, die man für die bisherigen Systeme gehabt hat. Aus diesem Grund wurde bei der Konzeptionierung von PIS sehr viel Wert auf die Gestaltung der Benutzeroberfläche und die Bereitstellung von Commands gelegt, sodass die Anforderungen der Nutzer möglichst gut abgedeckt werden. Die tatsächliche Akzeptanz wird sich im langfristigen Betrieb zeigen.

Was die Entwicklung von PIS anbelangt, so wird diese sicherlich fortgeführt. Ein Arbeitspaket für die unmittelbare Zukunft ist die Anpassung an jBPM 4. Bestimmt werden aber auch noch weitere Features integriert. Auch besteht die Idee, das System einem größeren Kreis von Entwicklern und Nutzern als wertvolle Ergänzung zu jBPM über ein Open-Source-Projekt zugänglich zu machen. Sollte bei Ihnen Interesse bestehen, so setzen Sie sich bitte mit den Autoren in Verbindung. Es gibt schließlich kein Open-Source-Projekt ohne Community. Erste Schritte dazu sind bereits getan, denn einige Ideen aus diesem Projekt sind in jBPM 4 eingeflossen – die Reise mit jBPM geht also weiter. ■

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. Zurzeit schreibt er u. a. an einem Buch zur BPMN und hilft beim Aufbau der SOA-Plattform der 1&1 Internet AG .

Dr. Jörg Moldenhauer ist Diplom-Informatiker und leitet das Team Technology & Infrastructure bei der 1&1 Internet AG, das für die Bereitstellung der eigenen SOA-Plattform verantwortlich ist. Er hat an der Universität Karlsruhe (TH) u. a. Softwaretechnik und Übersetzerbau studiert und dort anschließend im Bereich Anthropomatik promoviert.

Links & Literatur

[1] http://www.jboss.org/jbossesb/
[2] http://www.jboss.org/jbossjbpm/
[3] Demolsky, Rücker: „jBPM meets ESB“, in Java Magazin 09.2008
[4] http://www.eclipse.org/rap/