

Java™ magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

CD-INHALT



NI02

Erweiterungen der
I/O-Funktionalität ▶ 27

Product Line Engineering mit OSGi

Ein sinnvolles Paar? ▶ 105



Erste Sessions ab Seite ▶ 43

DEVOPS-KEYNOTE

von Matthias Marschall

Video von der W-JAX 2011

HIGHLIGHT

Cucumber-chef v1.0.4-0

ChameRIA 1.5.0

jBPM 5.1.0.Final

WEITERE INHALTE

- H-Ubu
- Apache POI 3.8 beta 4
- Activiti 5.8

Alle CD-Infos ab Seite 3

DevOps

Entwicklung und
Betrieb zusammen-
bringen ▶▶ 32, 41

Infrastructure
as Code
mit Chef ▶▶ 53



Datenträger enthält
Info- und
Lehrprogramme
gemäß § 14 JuSchG

GWT & HTML5

Das große Web-
Tutorial ▶ S. 60

Google Dart

Die neue Programmier-
sprache ▶ S. 75

Spring ohne XML

Java-basierte
Konfiguration ▶ S. 96

Ein Überblick über aktuelle Tools und Strömungen

Prozesse in Bewegung



Business Process Management ist auch in der IT angekommen, vor allem BPMN 2.0 als weltweiter Standard zur Prozessmodellierung und die Verbreitung von leichtgewichtigen, quelloffenen Java-Tools katalysieren die Verbreitung. Der Platzhirsch im Bereich Open Source Process Engines, JBoss jBPM 3, ist inzwischen in Version 5 verfügbar, der eigentliche Nachfolger von jBPM 3 ist aber Activiti. Bonita als Open Source BPM Suite sammelt gerade mächtig Risikokapital ein, um die etablierten Herstellern das Fürchten zu lehren. Kurzum, es ist ein spannendes Feld, aber leider auch ein unübersichtliches. Daher möchten wir uns den interessanten Fragen widmen, wie BPM in Java überhaupt Sinn macht, warum Philosophie eine Rolle spielt und wie Sie das richtige Tool für Ihr Projekt finden.

von Bernd Rücker

Business Process Management (BPM) ist ein großer Sammelbegriff für viele Disziplinen rund um Geschäftsprozesse. Viele davon haben mit IT oder Java nichts zu tun. Immer häufiger dreht es sich aber heute darum, wie Prozesse automatisiert werden können, sei es um effizienter zu werden, Kosten zu senken, Fehlerquoten zu reduzieren, Qualität zu steigern oder neue Geschäftsmodelle zu ermöglichen. Prozessautomatisierung geht uns als Java-Entwickler daher auch etwas an, schließlich wird ein nicht unerheblicher Teil in typischen „Businessanwendungen“ in Java entwickelt. Als Notation setzt sich aktuell international wie auch im deutschsprachigen Raum die BPMN durch, auch die aktuellen Open-Source-Tools setzen auf die BPMN. Einen beispielhaften Geschäftsprozess in BPMN sehen wir in **Abbildung 1**.

Anstatt die Prozesse nun aber durch hoffentlich ausgefeilte Methoden fachlich „nur“ zu modellieren, um sie als Anforderungen zu verwenden und dann manuell in Java-Code zu übersetzen, geht der Trend ganz

klar dahin, die Prozessmodelle auch direkt ausführbar zu machen. Dabei wird ein technisch angereichertes Prozessmodell auf einer so genannten Process Engine ausgeführt. Der Clou ist, dass sowohl die Fachabteilung als auch die Softwareentwicklung mit BPMN die gleiche Notation verwenden. BPMN in der Version 2.0 kann nämlich auch mit notwendigen Zusatzattributen versehen und als XML abgespeichert werden, was die Engine dann direkt lesen kann. Hierdurch entfällt das lästige, zeitraubende und fehleranfällige Übersetzen der Prozessmodelle in Quellcode, was ein großer Vorteil ist: Prozessmodelle sind auf einmal aktuell und verstauben nicht als veraltete Anforderungsdokumentation.

Wir sprechen hier vom „Business IT Alignment“, das verschiedenartig ausgeprägt sein kann. Ich möchte diesem Thema noch etwas Platz einräumen, da es meiner Ansicht nach sowohl ein großer Treiber für BPM-Projekte ist als auch ausschlaggebend für den Erfolg von BPM-Projekten sein kann, nicht zuletzt, da es meist auch eine große Sichtbarkeit im Management hat.

Business IT Alignment und das Versprechen des Zero Coding

Die Nutzung des „gleichen“ Modells auf Seiten des Fachbereichs und der IT kann unterschiedlichste Formen annehmen. Die wohl bekannteste ist die Vision des Zero Coding. Dahinter steckt der Traum, nicht mehr von den „unzuverlässigen“, „langsamen“ oder auch „begriffsstutzigen“ IT-Abteilungen abhängig zu sein und als Fachbereich sein Prozessmodell kurzerhand selbst „zusammenklicken“ zu können. Der BPM-Herstellermarkt bedient diese Vision teilweise sehr gezielt, da sich hier beeindruckende Demos und damit einhergehende einfache Verkaufsabschlüsse erzielen lassen.

Das Problem ist, dass die Komplexität halbwegs komplexer Lösungen irgendwo abgebildet werden muss. Der Fachbereich ist hierfür nicht ausgebildet, sodass es bei der IT hängen bleibt. Diese lehnt Zero-Coding-Tools aber gerne ab, da es nicht unbedingt produktivitätsfördernd ist, Java-Code in kleine Textboxen von Webanwendungen einzutragen. Ein großes Problem tritt meist auch dann ein, wenn man mit solchen Tools etwas umsetzen möchte, das der Hersteller nicht vorausgesehen und vorgesehen hat. Dies ist dann oft nur über Umwege oder gar nicht realisierbar, was dann oft zur Aufwandsexplosion führt.

Auf der Herstellerseite ist es ebenfalls nicht einfach, eine solche Lösung bereitzustellen, da die Oberfläche sehr ausgereift und intuitiv sein muss. Die Lösung muss viele Aspekte außerhalb der Prozesslogik abdecken können, beispielsweise Daten, Oberflächen und Formulare. Anders als es das Marketing vermittelt, sind Zero-Coding-Tools meiner Ansicht nach eher eine Nische mit begrenzten Anwendungsmöglichkeiten.

Business IT Alignment und Prozessmodelle

Hat man den Traum des Zero Coding verworfen, liegt folgender Ansatz nahe: Der Fachbereich liefert ein Prozessmodell als Vorlage, die IT setzt es entsprechend um und ändert es wo nötig. Meist ist die Erwartungshaltung hierbei, entweder nur technische Details hinzuzufügen oder einzelne Aktivitäten im Prozess zu verfeinern, beispielsweise durch Subprozesse.

Typischerweise möchten verschiedene Rollen in diesem Spiel unterschiedliche Tools verwenden. So ist ein Fachbereich meist wenig erpicht über Eclipse und der Java-Entwickler über Weboberflächen. Hier kann nun aber der BPMN-2.0-Standard helfen, da er es ermöglicht, Prozessmodelle auch mit unterschiedlichen Modellierungstools zu bearbeiten. Dies funktioniert zwar noch nicht wirklich „flächendeckend“, aber es wird

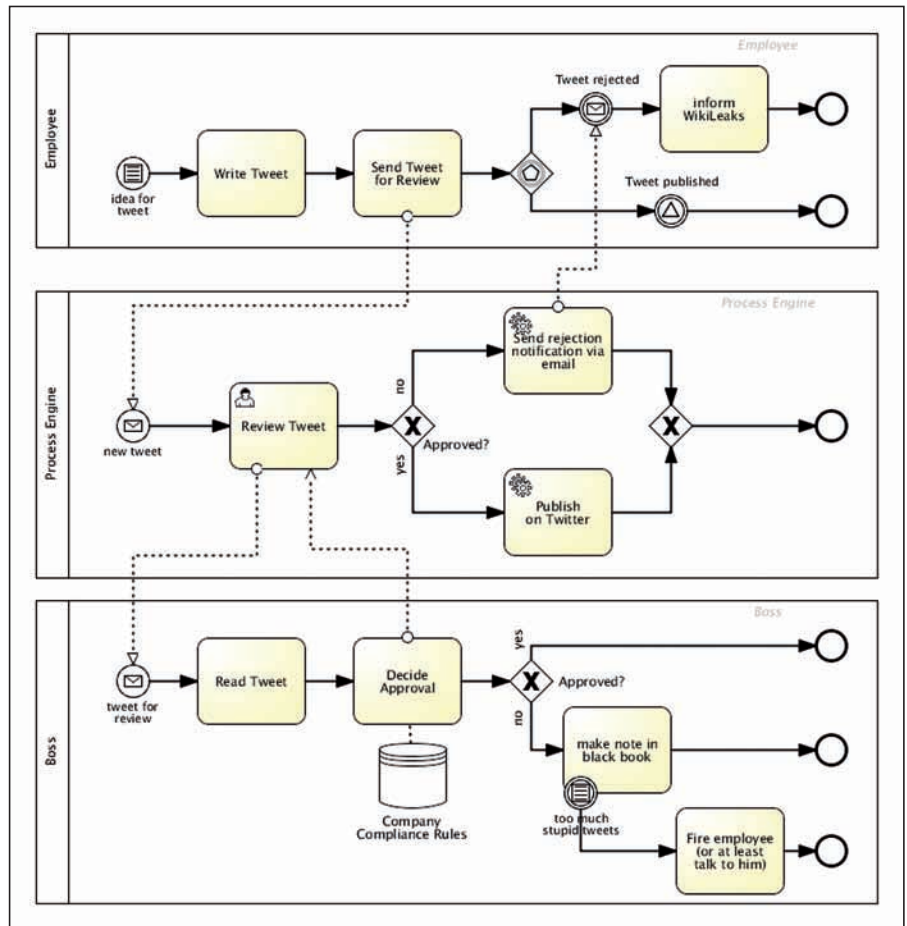


Abb. 1: Der Twitter-Beispielprozess als BPMN-Modell

zunehmend Realität. In der Beratungspraxis trifft uns aber ein anderes Problem: Rein fachliche Aspekte, zum Beispiel was ein Nutzer alles macht, was nicht in der Software abgebildet wird, oder rein technische Aspekte, zum Beispiel die Generierung von technischen Schlüsseln oder anderen fachlich irrelevanten aber notwendigen Aktionen, können nur extrem schwierig in einem einzigen Prozessmodell untergebracht werden. Außerdem ist es meist nicht hilfreich, wenn das Prozessmodell von beiden Fraktionen nicht ganz akzeptiert wird. Das führt dazu, dass sich eine Partei durchsetzt, natürlich meist die IT, da das Prozessmodell ja „Code“ ist.

Daher hat sich methodisch inzwischen ein Vorgehen etabliert, das einen expliziten technischen Prozess vorsieht, der allerdings mit seiner rein fachlichen Umwelt passend interagiert. **Abbildung 1** zeigt ein Beispiel, für weitere Details sei auf [1] verwiesen. Vorteil dieser Vorgehensweise ist, dass jede Fraktion ihren eigenen Prozess hat und dort auch nur die jeweiligen Aspekte abgebildet sind, sich aber trotzdem eine Gesamtsicht ableiten lässt.

Embeddable vs. BPM-Server

Neben der Umsetzung des Business IT Alignments gibt es noch einen weiteren zentralen Punkt für die Wahl einer Process Engine: die Einbettbarkeit (Embeddability). Betrachtet man JBoss jBPM 3, was meiner Einschätzung nach tatsächlich der Platzhirsch war, so war

dies vor allem aus einem Grund erfolgreich: Es ist ein Java-Framework, eine Library, etwas das ich in meine Anwendung einbinden kann, um den Prozessaspekt zu lösen. Es ist kein Server, macht kaum Annahmen über die Umgebung und kann sogar mit der Datenbank und den Transaktionen der eigenen Anwendung mitspielen, egal ob Java EE oder Spring. Man kann es aber auch völlig ohne Persistenz zum Beispiel im Rahmen eines JUnit Test Case hochfahren.

Ein weiterer großer Vorteil ist, dass Serviceaufrufe nun auch einfache Java-Methodenaufrufe sein können. Auch hier sind Web Services oder REST zwar möglich, aber nicht zwingend. Es können auch EJBs direkt transaktionsal aufgerufen oder vorhandene Softwarefunktionen in Java direkt integriert werden. Wir sprechen in diesem Zusammenhang auch gerne von Prozessanwendungen oder Prozesslösungen. Diese beinhalten neben der Prozesslogik auch noch viele weitere Bestandteile wie Daten, Geschäftslogik und Oberflächen. In diesem Szenario fügt sich die Process Engine nahtlos als Library in die Architektur ein.

Technische Auswahlkriterien für Process Engines

Im Folgenden will ich kurz die aus unserer Sicht wichtigsten Aspekte bei der Evaluierung einer Process Engine aufführen, ohne eine formale Bewertung für die vorgestellten Projekte vorzunehmen. Wir gehen auf ausgewählte Besonderheiten der Engines jeweils im Haupttext ein:

- API: Ist das API intuitiv, angemessen und gut dokumentiert? In welcher Technologie steht das API zur Verfügung (Java, EJB, REST, Web Service)?
- Embeddability: Kann die Engine in die eigene Umgebung (Spring, Java EE, OSGi) eingebettet werden? Wie viele Abhängigkeiten gibt es zu anderen Libraries oder Technologien?
- Persistenz: Wann, wie häufig und wie effizient werden Prozessinstanzen persistiert? Ist das Konzept auch für Massendaten tragfähig? Welche Datenbanken werden dabei unterstützt? Ist der Persistenzmechanismus austauschbar (Stichwort: Cloud)?
- Transaktionssteuerung: Wie ist die Transaktionssteuerung der Engine? Kann dies in die eigene Anwendung integriert werden (Stichwort: JTA)? Können Serviceaufrufe in die Transaktion einbezogen werden?
- Serviceaufrufe und Delegation: Wie können Services technologisch aufgerufen werden (Java, REST, Web Services) und wie einfach ist dies vom Programmiermodell her (Expressions mit Spring oder CDI Beans, XPath). Welche Scripting-Sprachen werden unterstützt? Wird JSR 223 unterstützt?
- Testbarkeit: Können Prozesse einfach als Test Cases umgesetzt werden? Wie kompliziert ist dies? Gibt es sonstige Unterstützung?
- Komplexität und Lernkurve: Wie aufwändig ist das Aufsetzen der Umgebung? Wie schnell kann der erste Prozess umgesetzt werden? Wie umständlich ist das Einbinden in die eigene Umgebung?
- Notation: Wird BPMN 2.0 unterstützt oder wird noch eine proprietäre Sprache (wie jPDL) oder ein überholter Standard (wie XPDL) eingesetzt?
- Konfiguration und Erweiterbarkeit: Wie leicht können Erweiterungen konfiguriert sein (z. B. Benutzer aus dem LDAP)? Können flexibel eigene Implementierungen für bestimmte Aspekte (zum Beispiel Aufgabenzuweisung) eingehängt werden?

Trotzdem lässt sich mit einem solchen Framework auch ein Standalone-Server aufbauen, beispielsweise als zentraler BPM-Server eines Unternehmens. Man kann JMS verwenden und die Process Engine nach jeder Aktivität den Prozess im aktuellen Zustand in die Datenbank schreiben lassen, zumindest bei den meisten Tools. Und spätestens mit der Kombination von anderen Frameworks oder ESBs wie Apache Camel, ServiceMix oder Mule lässt sich eine ernstzunehmende technische SOA-Infrastruktur aufbauen.

Open Source Process Engines im Überblick

Schauen wir uns genauer im Open-Source-Lager um. Dabei stellt man fest, dass es nur drei Projekte gibt, die als BPMN 2.0 Engine eine maßgebliche Rolle spielen: Bonita Open Solution [2], JBoss jBPM 5 [3] und Activiti [3]. Andere OS Process Engines in Java (z. B. Enhydra Shark, Apache ODE, OfBiz WorkflowEngine) wollen wir außen vor lassen, da sie BPMN nicht unterstützen und aktuell in der Breite wenig erfolgreich sind.

Interessant ist als Erstes ein kleiner Blick in die Geschichte, die immer wieder am eingangs erwähnten jBPM 3 ankommt. So ist Activiti technologisch gesehen der Nachfolger von jBPM 4, was wiederum der Nachfolger von jBPM 3 ist. jBPM 4 wurde allerdings seitens JBoss nie ganz fertig entwickelt, da in der Zwischenzeit das Projektteam von Alfresco angeheuert wurde, um Activiti zu entwickeln. Die Motivation war hier übrigens nicht technologisch, sondern lizenzrechtlich: Alfresco wollte eine Process Engine in das ECM-Produkt integrieren, die unter der Apache-Lizenz steht.

JBoss jBPM 5 ist dann allerdings entgegen der spontanen Erwartung technisch überhaupt nicht mehr mit den vorherigen Versionen verwandt, sondern basiert auf einem anderen Projekt: Drools Flow. Hier wurde die Regelmachine aus dem Hause JBoss um Workflow-funktionalitäten erweitert. Und auch Bonita hat mit dem jBPM-4-Projektteam zusammengearbeitet, sodass der Bonita-Kern technologisch auf jBPM-4-Konzepten beruht, wenn er sich dann auch unabhängig weiterentwickelt hat. Alles klar, oder?

Ein Beispielprozess

Werden wir konkret und schauen uns die drei Engines einmal genauer an. Wir tun dies anhand unseres Twitter-Prozesses, den wir gerne in Demos verwenden. Dieser wird durch einen Mitarbeiter gestartet, der twittern möchte. Daraufhin enthält der Prozess einen so genannten UserTask, da sein Vorgesetzter den Tweet freigeben soll. Je nach dessen Entscheidung wird über einen so genannten ServiceTask der Tweet veröffentlicht oder der Mitarbeiter per E-Mail über dessen Ablehnung in Kenntnis gesetzt. **Abbildung 1** zeigt den Prozess als BPMN-Diagramm, der mittlere so genannte Pool (Process Engine) soll dabei automatisiert werden, die anderen Pools dokumentieren den Arbeitsablauf der beteiligten Mitarbeiter.

Anzeige

Bonita Open Solution 5

Bonita hat jüngst etwas Aufsehen in der Branche erregt, als es sein Risikokapital um ganze elf Millionen US-Dollar aufstockte und gleichzeitig den etablierten proprietären BPM-Herstellern den Kampf ansagte [5]. Bonita zielt strategisch klar auf Zero Coding und eine

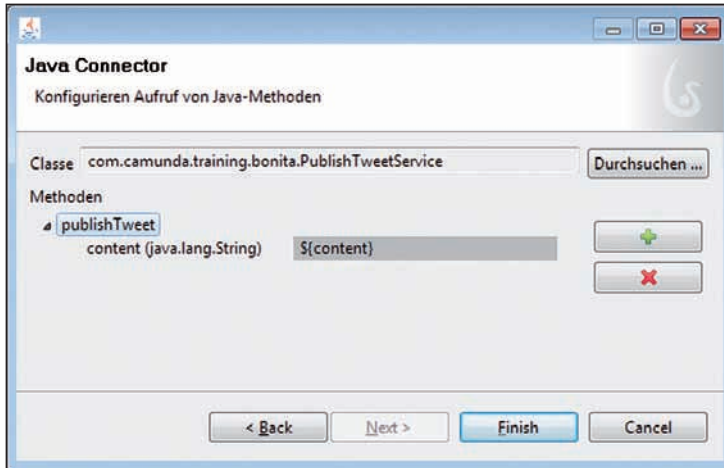


Abb. 2: Anbindung von Java-Code im Bonita-Studio

Listing 1: Anmeldung eines ServiceTasks im Designer

```
import org.drools....StringDataType;
[
[
"name" : "publish-tweet",
"parameters" : [
"Tweet" : new StringDataType(),
"Email" : new StringDataType(),
],
"displayName" : "Publish Tweet",
"icon" : "icons/twitter-icon.png"
]
]
```

Listing 2: WorkItemHandler zur Implementierung der Logik des ServiceTasks

```
import org.drools.runtime.process.*;
public class PublishTweetHandler implements WorkItemHandler {

public void executeWorkItem(WorkItem workItem, WorkItemManager manager) {
String tweet = (String) workItem.getParameter("content");
// add logic ...
manager.completeWorkItem(workItem.getId(), null);
}

public void abortWorkItem(WorkItem arg0, WorkItemManager arg1) {
// do nothing, a service task will not be aborted
}
}
```

einfache Benutzbarkeit ab. Dies zeigt sich auch darin, dass es relativ schnell installiert ist und man sofort loslegen kann, wobei sich alles im so genannten „Studio“ abspielt, dem Eclipse zugrunde liegt. Im Unterschied zu den beiden anderen hier vorgestellten Projekten ist Bonita GPL-lizenziert (Teile auch LGPL), das bedeutet dass es beispielsweise für die Einbettung in die eigene (Nicht-GPL-)Software lizenzrechtlich ungeeignet ist. Auch ist die GPL-Lizenz in großen Unternehmen aus Risikogesichtspunkten oft verboten, was allerdings durch den Kauf einer Enterprise-Lizenz umschifft werden kann.

Der Twitter-Prozess kann recht schnell modelliert werden. Spannend wird es aber beim Einbinden von Services im Prozess. Bonita bietet hierbei eine ganze Liste von Konnektoren an, die teilweise aus der Community kommen und teilweise selbst entwickelt werden. Diese können im Prozess über die IDE konfiguriert werden, was sich aber meist auf recht einfache Formulare beschränkt und nicht den Komfort bietet, den man von den kommerziellen Tools kennt. Trotzdem kann auch Java-Code, den ich für diese Demo in einem eigenen Entwicklungsprojekt geschrieben habe, angebunden werden, indem die erzeugte *jar*-Datei als Abhängigkeit deklariert wird. **Abbildung 2** zeigt die Konfiguration des so genannten Java-Konnektors zum Aufruf unseres eigenen Java-Codes. Unschön ist übrigens dabei, dass Bonita das Prozessmodell in einem eigenen Repository ablegt, was dann in der Community-Edition nicht mit anderen Nutzern geteilt werden kann (dies ist anders als bei jBPM und Activiti, die das normale Dateisystem in Eclipse verwenden und damit vor allem auch normale Versionskontrollsysteme wie SVN oder GIT erlauben).

Technisch gesehen kommt Bonita als Java-Anwendung daher, die Persistenz über Hibernate erledigt und über ein XML-File konfiguriert werden kann. Es gibt ein Java Client API sowie ein REST API. Es wird eine Expression Language für Entscheidungen verwendet und man kann sehr einfach Groovy-Skripte einbetten. Soweit ist das Ganze also für uns als Java-Entwickler ganz brauchbar. Leider merkt man jedoch, dass Bonita nicht unbedingt den Java-Entwickler als Zielgruppe im Sinn hat. So wird standardmäßig sehr viel von der darunter liegenden Technik versteckt und man muss ein bisschen suchen, um die relevanten Informationen zu finden. Das Java API wirkt ein bisschen, als ob es früher entstanden ist, als geplant war. Für die Einbettung in eine eigene Anwendung und beispielsweise den Abruf einer gefilterten Task-Liste oder die Suche nach Prozessinstanzen ist das API schnell nicht mehr ausreichend. Java-Klassen können auch nicht im Bonita-Studio entwickelt werden, sondern man muss Eclipse dafür bemühen. Dies demonstriert sehr gut die Denkweise von Bonita: Der Java-Entwickler ist nicht unbedingt derjenige, der dann den Prozess „zusammenklickt“ und man baut auch keine integrierte Prozesslösung mit anderen Bestandteilen als dem, was ich im Bonita-Studio modellieren kann.

Als Zero-Coding-Tool macht Bonita dann doch einen nicht ganz ausgereiften Eindruck, da vieles noch nicht intuitiv bedienbar ist, Wizards fehlen und es auch noch einige sehr störende Bugs gibt. Bei der deutschen Übersetzung rollen sich einem übrigens auch die Zehennägel auf. Der eingebaute Formulareditor ist für einfache Fälle zwar ausreichend, wirkt aber schnell beengend. Die mitgelieferte Weboberfläche für Aufgabenliste und Formulare wirkt sehr unaufgeräumt, auch wenn die Grundkonzepte durchaus gut sind.

BPMN 2.0 nimmt Bonita leider nicht zu 100 % ernst. Zwar ist die Darstellung korrektes BPMN, es werden vergleichsweise viele Symbole unterstützt und Prozesse können in BPMN 2.0 exportiert und importiert werden. Dabei gehen allerdings sehr viele Informationen verloren, zum Beispiel welche Konnektoren wie eingebunden sind. Nach einem Re-Import des eigenen Exports ist das Prozessmodell also nicht mehr lauffähig. Intern verwendet Bonita auch ein anderes XML-Format um Prozesse auszuführen. Dies verhindert, dass im Projekt auch ein anderes Tool für die Modellierung verwendet wird, beispielsweise durch die Businessanalysten. Eine eigene Vision für das Business IT Alignment bringt Bonita dann aktuell leider auch nicht mit, sodass wir hier in der oben beschriebenen „Ein Modell für alles“-Falle stecken.

Wirtschaftliche Auswahlkriterien für Process Engines

Folgende Kriterien können eine Rolle bei der Auswahl der Process Engine spielen:

- **Lizenz:** Wie liberal ist die Open-Source-Lizenz und erlaubt sie beispielsweise den Einbau in das eigene Closed-Source-Produkt oder Framework? Gilt die gleiche Freizügigkeit für alle Third Party Libraries, die verwendet werden? Wenn nicht, ist dies ein Problem?
- **Community:** Wie groß und aktiv ist die Community? Welche Rollen tummeln sich dort hauptsächlich und werden dort die Art Fragen beantwortet, die mich betreffen?
- **Zukunftssicherheit:** Hat das Projekt eine ausreichend große Kundenbasis? Gibt es ein kommerzielles Produkt, was gewisse Wartungszeiträume garantiert? Wer treibt das Projekt und mit welchem Interesse? Ist das auch in fünf Jahren noch gewahrt?
- **Kommerzieller Support:** Gibt es kommerziellen Support auch für den Betrieb? Spätestens in großen Unternehmen kann dies einen Show Stopper darstellen.
- **Verfügbare Services:** Gibt es Trainings und Beratungspakete? Können auf dem Markt genügend Mitarbeiter, Freiberufler oder Berater mit entsprechendem Know-how gefunden werden?

Anzeige

Die jBPM-5-Engine ist schwer verständlich, was aus Performancegesichtspunkten kritisch ist.

Zusammenfassend kann man sagen, dass Bonita vor allem durch das schnelle Setup punktet, das Einstiegschürden schnell überwindet. Für kleinere Prozesse oder Teams ohne Java Know-how kann es dann auch eine durchaus brauchbare Lösung sein. Für größere Projekte, ernsthafte Entwicklung von Java-Prozessanwendungen oder auch Business-IT-Alignment-Bemühungen mit BPMN ist es im aktuellen Stand kritisch zu sehen.

JBoss jBPM 5

Mit großen Hoffnungen wurde jBPM 5 als BPM-Flaggschiff aus dem Hause JBoss erwartet. Wie bereits erwähnt basiert es technologisch auf DroolsFlow aus dem JBoss-Drools-4-Projekt und ist dabei durch Apache lizenziert. Damit bleibt eigentlich aus „den alten Zeiten“ nur die Marke jBPM. Dementsprechend ist auch eine Migration und Umgewöhnung von alten jBPM-Versionen schwierig, wobei es ein Projekt zur Migration von Prozessdefinitionen gibt [6]. Die Vision des Projekts ist die Bereitstellung nicht nur einer Process Engine, sondern eines Tool-Stacks, der sowohl Entwickler als auch Businessanalysten anspricht. Dies resultiert in einem Eclipse-Plug-in mit BPMN Designer sowie einer Web-Oberfläche (Drools Guvnor) zur Verwaltung und Modellierung von BPMN-Prozessen.

Modellieren wir den Prozess im Eclipse-Designer, stoßen wir schnell auf ein erstes Problem beim Umset-

Listing 3: Clientcode zum Starten einer Prozessinstanz

```
StatefulKnowledgeSession ksession = createKnowledgeSession
    ("ReviewTweetProcessBpmn.bpmn");

ksession.getWorkItemManager().registerWorkItemHandler
    ("Human Task", new TestWorkItemHandler());
ksession.getWorkItemManager().registerWorkItemHandler
    ("Notification", new NotificationHandler());
ksession.getWorkItemManager().registerWorkItemHandler
    ("publish-tweet", new PublishTweetHandler());

Map<String, Object> variables = new HashMap<String, Object>();
variables.put("content", "Testing jBPM 5");
variables.put("email", "info@camunda.com");

ProcessInstance processInstance = ksession.startProcess
    ("com.camunda.training.reviewtweet", variables);

//...
```

zen des ServiceTasks zur Veröffentlichung des Tweets: Die BPMN-Palette enthält keinen ServiceTask. Hier beschreibt jBPM 5 einen eher ungewöhnlichen Weg: Der ServiceTask muss zuerst wie in Listing 1 gezeigt konfiguriert (im Designer „angemeldet“) werden und steht dann als eigener TaskTyp sogar mit eigenem Icon zur Verfügung. Um die Logik des ServiceTasks umzusetzen, muss dann ein so genannter *WorkItemHandler* implementiert werden (Listing 2). Zur Laufzeit muss eine Instanz des *WorkItemHandlers* der Prozessmaschine übergeben werden. Listing 3 zeigt hierzu den Ausschnitt aus dem Clientcode, der benötigt wird, um eine Prozessinstanz zu starten. Die Registrierung der Handler ist für Test Cases zwar äußerst hilfreich, im produktiven Einsatz muss man dadurch aber auch das Problem der Instanziierung der Handler im jBPM 5 umgebenden Code selbst lösen. Dies führt dann vor allem dazu, dass dieser Code wissen muss, welche Handler später im Prozess gebraucht werden, was deutlich mehr ist als er wissen sollte. Zwar könnte man sich nun einen generischen „JavaTask“ bauen, muss dies aber erst selbst tun. Das funktionsfähige Twitter-Beispiel kann übrigens online abgerufen werden [7].

In Listing 3 sehen wir noch etwas Interessantes: Das API für jBPM 5 kann seine Drools-Wurzeln nicht verbergen. Es wird eine *KnowledgeSession* erstellt, mit der dann ein Prozess gestartet wird. Intern arbeitet jBPM 5 übrigens tatsächlich auf Basis der Drools-Regelmaschine, was zwar einige Vorteile mit sich bringt, z. B. das Einbinden von Geschäftsregeln im Prozess, aber auch einen großen Nachteil hat: Die Engine ist schwerer verständlich (was wir ignorieren könnten) und braucht immer komplette Prozessinstanzobjekte im Speicher (sonst kann eine Regelmaschine nicht arbeiten). Letzteres ist nun aber aus Performancegesichtspunkten sehr kritisch, vor allem da es dazu geführt hat, dass Prozessinstanzen gleich als Blobs in der Datenbank abgelegt werden, was einen großen Show Stopper für die Skalierung darstellt. Man stelle sich hier nur einmal vor, eine Prozessinstanz mit bestimmten Eigenschaften zu suchen, was die Datenbank dann nicht direkt übernehmen kann.

Ein weiterer interessanter Punkt ist, dass jBPM 5 für die Aufgabenverwaltung einen eigenen Task-Server entwickelt hat. Dieser basiert auf dem WS-HumanTask-Standard, muss allerdings auch als eigener Server betrieben werden. Dabei treten auch noch Kinderkrankheiten zu Tage, der Server muss z. B. immer unter 127.0.0.1 erreichbar sein, was zumindest die Frage aufwirft, warum er dann nicht gleich im JBoss Application Server von jBPM 5 integriert ist.

Ansonsten ist jBPM 5 aber durchaus in verschiedenen Umgebungen lebensfähig, sofern man ein bisschen Konfigurationsaufwand auf sich nimmt. Dabei könnte man auch seine eigene Aufgabenverwaltung einhängen, wenn einem WS-HumanTask zu kompliziert ist. Spätestens bei der Persistenz müsste man aber viel an Aufwand investieren, um eine massendatenfähige Engine zu erhalten.

BPMN 2.0 wird in jBPM 5 ernst genommen, die Abdeckung ist in der Dokumentation aufgeführt [8]. Der Prozess wird als BPMN 2.0 XML mit Erweiterungen für jBPM 5 abgespeichert und in die Engine eingelesen. Der Designer kommt auch damit klar, wenn man mit anderen Tools Änderungen vornimmt, gute Voraussetzungen also. Fast schon störend wirkt aber, dass jBPM 5 die BPMN 2.0 zu konsequent umgesetzt hat, so muss man Daten im Prozess ebenfalls definieren und vor allem dann ein Mapping dieser Daten in jedem Schritt vornehmen, der Daten benötigt oder bereitstellt. Was soweit sinnvoll klingt ist in der Praxis sehr aufwändig, auch weil es durch das Eclipse-Plug-in kaum unterstützt wird. Für Java-Entwickler ist es auf jeden Fall umständlich, was wieder die Lern- und Akzeptanzkurve erhöht.

Als letzten Punkt möchte ich einen kurzen Blick auf das Web-Tooling werfen, das aus der jBPM Console, Guvnor und einem webbasierten Modeler für den Businessanalysten besteht. Leider sind hier alle Tools wenig durchdacht was die Benutzerführung und den genauen Verwendungszweck angeht, sodass die Bedienung unintuitiv und mühsam ist. Spätestens für Nichttechniker scheint Guvnor also ungeeignet, obwohl es doch mit Governance und Geschäftsregeleditoren eine fachliche Zielgruppe ansprechen will. Im aktuellen Stand haben Kunden, die es produktiv einsetzen wollten, auch viele Bugs oder gar Probleme im Classloading selbst fixen müssen, wodurch der Einsatz von Guvnor an sich leider kritisch zu sehen ist. Auch wird Business IT Alignment zwar viel diskutiert und auch gewürdigt, eine durchgehende erkennbare Vision fehlt dem jBPM-5-Projekt aber leider noch.

Zusammenfassend kann man sagen, dass sich jBPM 5 als Engine klar für Java-Entwickler positioniert, hierbei aber eine relativ hohe Lernkurve mitbringt und die Prozessumsetzung vergleichsweise aufwändig ist. In der aktuellen Version gibt es auch noch einige Designentscheidungen, durch die ich den Einsatz in Praxisprojekten mehr als kritisch sehe. Ein Trostpflaster bleibt die Unterstützung durch JBoss. Sie möchten jBPM 5 im Jahr 2012 produktisieren und damit vermutlich auch in die SOA Plattform aufnehmen und supporten.

Activiti 5

Activiti, als neues Open-Source-Projekt von Alfresco initiiert, hat von Anfang an relativ große Aufmerksamkeit auf sich gezogen. Technisch tritt es in die Fußstapfen von jBPM 4 und startete deshalb auch selbstbewusst als Activiti 5. Es ist durch Apache lizenziert und damit sehr vielseitig einsetzbar. Anders als jBPM 5 bei JBoss wird Activiti nicht als eigenes Produkt bei Alfresco verkauft und supportet, auf der anderen Seite gibt es bereits kommerzielle Enterprise-Versionen mit Support für Activiti [9].

Der Tweet-Prozess kann nun wie in den anderen Tools mit einem Eclipse-Designer umgesetzt werden. Auch bei Activiti hat der Designer Kinderkrankheiten. Im Gegensatz zu jBPM 5 werden in Activiti generische

ServiceTasks im Prozess eingefügt, die dann jeweils auf eine Java-Klasse verweisen, die die Logik implementiert. Trotzdem ist es möglich, eigene Task-Typen zu definieren und auch als Plug-in dem Designer bekannt zu machen, sogar mit eigenem Konfigurations-Panel. Listing 4 zeigt beispielhaft das BPMN 2.0 XML (was übrigens in jBPM 5 durchaus ähnlich aussieht) und Listing 5 die benötigte *JavaDelegate*-Klasse. Der gesamte Quellcode kann online abgerufen werden [10].

Im Gegensatz zu jBPM 5 übernimmt Activiti auch die Instanziierung der *JavaDelegate*-Klassen, wobei hier sogar das Classloading beeinflusst werden kann. So gibt es in camunda fox beispielsweise einen JBoss Deployer [11], der Prozesse als Teil eines normalen Deployments akzeptiert und es damit erlaubt, Prozesse zusammen mit Klassen und Ressourcen als *jar*, *war* oder *ear* auf eine zentrale Engine zu deployen. Trotzdem bekommt man die richtigen Klassen instanziiert, was sogar versioniert funktioniert.

Listing 4: Der Prozess als BPMN 2.0 XML mit Activiti Extensions

```
<definitions ... xmlns:activiti="http://activiti.org/bpmn">
<process id="TwitterDemoProcess" name="TwitterDemoProcess">
  <startEvent id="startevent1" name="Start" />
  <userTask id="usertask1" name="Review Tweet" activiti:candidateGroups=
                                                                    "management"/>
  <exclusiveGateway id="exclusivegateway1" name="Approved?" />
  <serviceTask id="servicetask1" name="Publish on Twitter" activiti:class="com.camunda.
                                                                    fox.demo.twitter.TweetContentDelegate" />
  <serviceTask id="mailtask1" name="Send rejection notification via email"
                                                                    activiti:type="mail">
    <extensionElements>
      <activiti:field name="to" expression="\${email}" />
      ...
    </extensionElements>
  </serviceTask>
  <exclusiveGateway id="exclusivegateway2" name="Exclusive_Databased_Gateway"/>
  <endEvent id="endevent1" name="End"/>
  <sequenceFlow id="flow1" sourceRef="startevent1" targetRef="usertask1"/>
  <sequenceFlow id="flow2" sourceRef="usertask1" targetRef="exclusivegateway1"/>
  <sequenceFlow id="flow3" sourceRef="servicetask1" targetRef="exclusivegateway2"/>
  <sequenceFlow id="flow4" sourceRef="mailtask1" targetRef="exclusivegateway2"/>
  <sequenceFlow id="flow5" sourceRef="exclusivegateway2" targetRef="endevent1"/>
  <sequenceFlow id="flow6" sourceRef="exclusivegateway1" targetRef="mailtask1">
    <conditionExpression><![CDATA[#{!approved}]]></conditionExpression>
  </sequenceFlow>
  <sequenceFlow id="flow7" sourceRef="exclusivegateway1" targetRef="servicetask1">
    <conditionExpression><![CDATA[#{approved}]]></conditionExpression>
  </sequenceFlow>
</process>
<bpmndi:BPMNDiagram id="BPMNDiagram_TwitterDemoProcess">
  ...
</bpmndi:BPMNDiagram>
</definitions>
```

Oft ist es zielführend, durch ein Pilotprojekt die präferierte Engine abzusichern

An dieser Stelle können in Activiti übrigens nicht nur *JavaDelegates* verwendet werden, sondern auch direkt Expressions (in der Unified Expression Language). Bei geeigneter Konfiguration können diese direkt auf Spring oder CDI Beans zugreifen, was ein komfortables Programmieren ermöglicht, zumal auch die JBoss-Tools eingesetzt werden können, um hier sogar im XML des Prozesses Code-Completion zu erreichen.

Persistenz erledigt Activiti über Apache MyBatis, wodurch dann auch das ganze Projekt ohne LGPL-Abhängigkeiten auskommt (jBPM 5 verwendet Hibernate, das LGPL-lizenziert ist und somit für die Einbettung zumindest ein Risiko darstellen kann). Dadurch ist die Anbindung neuer Datenbanken etwas umständlicher, da eventuell SQL-Skripte „übersetzt“ werden müssen. Auf der anderen Seite ist Activiti damit auch für die entsprechenden Datenbanken optimiert und kann mit Masendaten gut umgehen.

Viel gelobt wird das saubere API der Engine, das durch die Evolution seit jBPM 2 (an jBPM 1 möchten wir uns lieber nicht erinnern) inzwischen sehr ausgereift ist. Dort ist auch ein relativ mächtiges Query API enthalten, um flexible Abfragen rein über das API zu programmieren. Listing 6 zeigt beispielhaften Clientcode für den Start einer neuen Prozessinstanz.

Listing 5: JavaDelegate zum Aufruf von Logik im Prozess

```
import org.activiti.engine.delegate.*;
public class TweetContentDelegate implements JavaDelegate {

    public void execute(DelegateExecution execution) throws Exception {
        String content = (String) execution.getVariable("content");
        // do what you need to do
    }
}
```

Listing 6: Starten einer Prozessinstanz in Activiti

```
// build default Process Engine
ProcessEngine engine = ProcessEngineConfiguration.buildProcessEngine();
RuntimeService service = processEngine.getRuntimeService();

HashMap<String, Object> variables = new HashMap<String, Object>();
variables.put("content", "Testing Activiti");
variables.put("email", "info@camunda.com");
ProcessInstance processInstance = runtimeService.startProcessInstanceByKey(
    ("TwitterDemoProcess", variables);
```

BPMN 2.0 wird in Activiti auch durchgehend unterstützt, wobei allerdings Daten nicht wie in jBPM 5 deklariert und gemappt werden. Viel mehr gibt es für jede Prozessinstanz auch eine Java Map für beliebige Prozessvariablen, die ebenfalls persistiert wird. Dies hat zwar den Nachteil, dass der Datenfluss manchmal etwas schwerer nachzuvollziehen ist, macht aber die Entwicklung deutlich einfacher und erlaubt mehr Freiheiten. Dies kann man z. B. im CDI Binding sehen, das einen eigenen Scope für Prozesse und ihre Daten einführt [12]. Ein weiteres Beispiel ist die Unterstützung von JPA-Entitäten als Prozessvariablen, wobei Activiti nur den Primärschlüssel in der eigenen Datenbank speichert. Somit erscheint es für die Entwicklung von Java-Lösungen die bessere Wahl zu sein. Von der BPMN-Abdeckung werden inzwischen die wichtigsten Symbole unterstützt, wobei aber auch der eine oder andere weiße Fleck existiert [13].

Activiti positioniert sich als Projekt klar als embeddable Java Engine und schafft es durch diese Fokussierung auch, Java-Entwickler gut abzuholen. Die Lernkurve ist flacher als bei jBPM 5, auch wenn die Installation nicht ganz so einfach ist wie bei Bonita. Allerdings sollte es Entwickler nicht vor Probleme stellen, einen Ant-Task auszuführen. Für verschiedene Laufzeitumgebungen (Spring, Java EE, JBoss, OSGi) bietet Activiti dann Erweiterungen beispielsweise durch Partnerschaften an. So sind die Spring-, die Java-EE/CDI-, aber auch die OSGI/ServiceMix/Camel-Integration bereits sehr tragfähig und produktiv einsetzbar. Trotzdem kann Activiti auch einfach in Unit Tests betrieben werden und stellt eigentlich gar keine hohen Anforderungen an die Umgebung. Im einfachsten Fall kann Activiti mit einer Hand voll Abhängigkeiten betrieben werden, hauptsächlich der JDBC-Treiber und MyBatis. Dies ist ein wesentlicher Vorteil spätestens bei der Einbindung in die eigene Anwendung.

Im Bereich Business IT Alignment verfolgt Activiti eine ähnliche Strategie. Dieser Punkt ist „ausgelagert“ und wird aktuell durch zwei Tools abgedeckt: camunda fox cycle und Signavio Process Editor [14]. Ersteres ist Open Source verfügbar und sozusagen der Glue Layer zwischen den verschiedenen Rollen und Tools, die aus unserer Projekterfahrung in BPM-Automatisierungsprojekten verwendet werden. Letzteres ist ein kommerzieller webbasierter BPMN Modeler, von dem es allerdings auch eine abgespeckte Open-Source-Variante gibt, die aktuell nur mit etwas Fummelei zum Laufen zu bewegen ist. Insgesamt setzt dieser Stack aus mehreren Tools eine gute Vision zum Business IT Alignment um, wobei ich selbst nicht unbedingt objektiv bin, da wir selbst diese Vision nach unserer Projekterfahrung treiben und weiterentwickeln.

Zusammenfassend kann man sagen, dass mit Activiti die technisch ausgereifteste Engine zur Verfügung steht, was sicherlich auch durch die Konzentration auf die Kernfunktionalitäten gefördert wird. Eine Lernkurve ist auch hier zu überwinden und ohne Java-Entwickler ist

Activiti sicherlich keine gute Wahl. Auch muss die Engine meist erst durch eigene Leistung im Projekt in die Zielarchitektur eingebettet werden, auch wenn es schon heute Produkte gibt, die diese Leistung für bestimmte Stacks übernehmen. Kinderkrankheiten gibt es in den „umgebenden“ Tools wie dem Designer, der webbasierten Task-Liste oder auch der REST-Schnittstelle.

Bewertung

In diesem Artikel kann ich mit Sicherheit keine abschließende Bewertung geben oder einen umfassenden objektiven Vergleich anstellen, was sowieso nicht pauschalisiert werden kann. Ich wollte Ihnen einen Eindruck von den aktuell spannenden OS-BPM-Projekten geben, wobei dies eher bedeutete, ein paar Schlaglichter auf einzelne Bereiche zu werfen. Viele Aspekte, wie das Monitoring, haben wir hier nicht weiter beleuchtet, auch wenn sie in der Praxis eine wichtige Rolle spielen. Hier sind aber alle Kandidaten schwach auf der Brust, was teilweise durch kommerzielle Add-ons wieder wettgemacht werden kann.

Man kann jedoch im aktuellen Markt zwei Dinge klar beobachten: Der Trend geht zu BPMN 2.0, und Open Source Process Engines werden auch in großen Projekten ernst genommen. Auch der BPM-Markt an sich verändert sich aktuell, so gibt es wenig Nachfrage nach Standalone-BPM-Tools und vor allem im Kontext von Java-Projekten die Forderung nach „embeddable Engines“. Da Bonita strategisch einen anderen Weg geht, ist es in der Java-Welt einfach weniger geeignet und kann eher für kleine und einfache Projekte eingesetzt werden. Da kommt einem dann auch die geringe Lernkurve zu Gute und man kann ganz glücklich werden, sofern man nur die 80 % tut, die Bonita vorgesehen hat.

Für ausgewachsene Prozessanwendungen mit Java sollte auf eine Engine mit Fokus Java gesetzt werden. Durch einige zumindest in der aktuell vorliegenden Version 5.1 vorhandene Mängel in jBPM würde ich in einem ernsthaften Projekt eher davon abraten, auch wenn man die JBoss-SOA-Plattform einsetzen oder die Regelmaschine im Prozess verwenden möchte. Dadurch landet man dann bei Activiti, was aus meiner Sicht eine gute Wahl darstellt.

Auch die Betrachtung der wirtschaftlichen Faktoren wie Roadmap, Community und Zukunftssicherheit sind bei Activiti aktuell positiver, so hat sich JBoss leider in jüngster Vergangenheit nicht unbedingt einen Namen damit gemacht, eine verlässliche Roadmap herauszugeben und Communityprojekte (wie jBPM 5) auch wirklich zu produktisieren. Bonita ist hier unproblematischer, da das Unternehmen dahinter rein vom Erfolg des Produkts abhängt. Allerdings sind sowohl Community als auch Referenzprojekte vor allem im deutschsprachigen Raum etwas mager.

Und jetzt?

Ich hoffe der Artikel konnte einen ersten Eindruck geben und den Einstieg in die Evaluierung erleichtern, aber

auch das Interesse an Open Source Process Engines steigern. Natürlich können und müssen alle Projekte noch tiefer betrachtet und objektivere Vergleiche angestellt werden, soll eine gut abgesicherte Entscheidung getroffen werden. Oft ist es dann zielführend, durch ein Pilotprojekt die präferierte Wahl abzusichern. Dabei sollten technische und wirtschaftliche Überlegungen einbezogen werden, aber auch das Business IT Alignment und die Umsetzung methodischer Ideen nicht vernachlässigt werden. Denn eigentlich sind Process Engines sowie die BPMN genau für die Unterstützung der Brücke zwischen Fachbereich und IT angetreten.



Bernd Rücker (bernd.ruecker@camunda.com) verfügt über umfangreiche Projekterfahrung im Bereich BPM sowie Java Enterprise. Sein Schwerpunkt liegt in den Bereichen Process Execution, Business Rules, SOA und Business IT Alignment. Bernd ist langjähriger Committer im jBPM- und im Activiti-Projekt.

Links & Literatur

- [1] Freund, Jakob; Rücker, Bernd: Praxishandbuch BPMN 2.0
- [2] <http://www.bonitasoft.com/>
- [3] <http://www.jboss.org/jbpm>
- [4] <http://www.activiti.org/>
- [5] <http://www.bonitasoft.com/company/blog/bonitasoft-closes-11-million-series-b-funding-fuel-continued-worldwide-growth-and>
- [6] <http://www.schabell.org/2011/10/jbpm-migration-tooling-available-in.html>
- [7] <https://svn.camunda.com/fox/demo/jbpm5/twitter/trunk/>
- [8] <http://docs.jboss.org/jbpm/v5.1/userguide/ch06.html#d0e1992>
- [9] <http://www.camunda.com/fox/>
- [10] <https://svn.camunda.com/fox/demo/activiti/twitter/trunk/>
- [11] <http://www.bpm-guide.de/2011/05/09/going-java-enterprise-with-activiti-and-jboss-as/>
- [12] <http://www.bpm-guide.de/2011/09/28/less-code-bpm-with-camunda-fox-server-activiti-and-jboss-as-7/>
- [13] <http://www.activiti.org/userguide/index.html#bpmnConstructs>
- [14] <http://www.signavio.com/>