8.07

Deutschland €6,50 Niederlande €7,80 Österreich €7,50 Schweiz sFr 12,70

S&S

Javamagazin

Javamagazin™

Internet & Enterprise Technology

www.javamagazin.de

Auf CD

Alle Infos auf S. 3

Testversionen & more

- JAX-TV: Interview mit Dierk König
- TeamCity 2.1
- Groovy 1.0
- Ivy 1.4.1
- Apache Forrest 0.8
- DbUnit 2.2

Enterprise ARCHITEKTUR MAGAZIN

→ Lesen Sie im Heft ab Seite 67!

OSGi Die Zukunft Javas?

→ Komponenten in Java, jetzt aber richtig!

→ Mit Eclipse OSGi-Bundles entwerfen

SOA mit Open Source

Was leisten die freien Tools?

JSF im Härtetest

Praxis in großen Projekten

Reporting unternehmensweit

Ein Bericht sagt mehr als 1.000 Zeilen

Zahlen & Fakten

JSR 275: Internationale wissenschaftliche Formate

Spring-Framework

Avalon-Spring-Migration – ein Praxisbericht

Crossmedia mit XML

Apache Forrest bedient viele Formate

Datenträger enthält nur Lehr- oder Infoprogramme

D 45867

4 194586 706503 08

Realisierung einer serviceorientierten Architektur auf der Basis von Open Source

SOA auf die leichte Art

VON WOLFGANG PLEUS

Derzeit positionieren sich die Großen der Branche als Anbieter von Plattformen für serviceorientierte Architekturen (SOA). Dies lässt erahnen, wie bestimmend das SOA-Paradigma für die nähere Zukunft sein wird. Auch jenseits der etablierten Anbieter entstehen anspruchvolle Produkte im Open-Source-Bereich, die jedoch mangels Marketing oft kaum wahrgenommen werden. Dabei haben auch diese Produkte eine Menge zu bieten.

Getrieben vom eigenen Produktportfolio hat jeder Hersteller eine eigene Vorstellung von der technischen Ausgestaltung einer serviceorientierten Architektur. Unabhängig von den Unterschieden finden sich aber immer wieder gleiche Funktionalitäten, sodass eine serviceorientierte Referenzarchitektur in der Regel einen oder mehrere Bausteine aus Tabelle 1 umfasst.

Der Enterprise Service Bus sowie das Business Process Management stellen die Kernfunktionalitäten einer SOA bereit, da sie es ermöglichen, Prozesse zu definieren und auszuführen. Eine Rule Engine erlaubt es, fachliche Regeln in die Prozesse einzubeziehen und dynamisch anzupassen. Mit zunehmender Komplexität einer SOA wird die Verwaltung von Services, Regeln oder Richtlinien erschwert. Eine Registry/Repository hilft dabei, die Komplexität zu beherrschen. Sicherheitsaspekte wie Authentifizierung und Autorisierung werden durch ein Identity-Management-System abgebildet. Und schließlich bedarf es der Möglichkeit, eine SOA zu überwachen. Da die Interaktion häufig asynchron und verteilt abläuft, stellt dies eine echte Herausforderung dar. Durch Health Tracking lassen sich Fehlersituationen aufspüren und beseitigen, während Business Activity Monitoring zur statistischen Aufbereitung der erfassten Daten dient. Daduch lassen sich beispielsweise Nutzungsprofile für einzelne Services erstellen.

Quellcode auf CD

Open-Source-Bausteine

Mit Ausnahme der Tracking- und Monitoring-Funktionalität gibt es für die oben genannten Bausteine Open-Source-basierte Implementierungen mit einem zufriedenstellenden Reifegrad (Tabelle 2). Da es den Umfang des Artikels sprengen würde, alle Bausteine im Detail zu betrachten, konzentrieren wir uns zunächst auf die Kernbausteine Enterprise Service Bus und Business Process Management.

Mit Apache ServiceMix steht ein Java Business Integration-(JBI-)konformer ESB zur Verfügung, während Apache Ode die Laufzeitumgebung für Business Process Execution Language (BPEL) bereitstellt. JBI [6] ermöglicht die herstellerunabhängige Wiederverwendung von JBI-Komponenten und erhöht somit die Investitionssicherheit. Dieses Versprechen ist nicht neu, wurde ein ähnliches Ziel doch auch mit EJB als Komponentenmodell verfolgt. Dies war aber nur bedingt erfolgreich, da wichtige Details

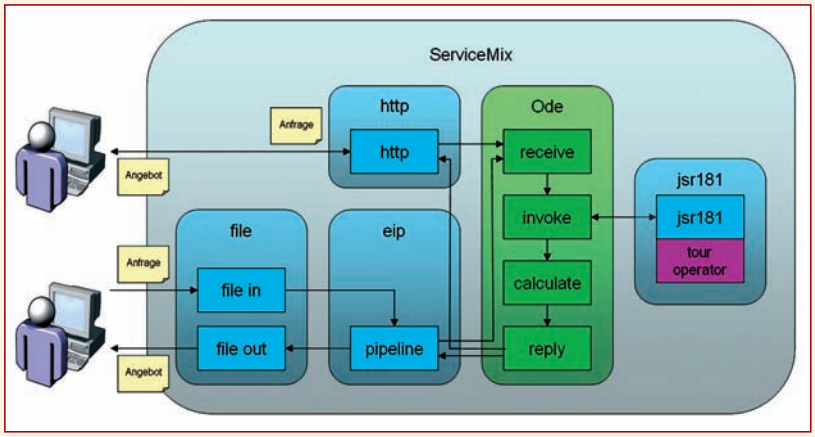

Abb. 1: Integriertes Szenario

wie Verteilung und Verwaltung nicht im Standard geregelt waren.

Wie es scheint, hat man dazugelernt und neben der Interaktion und Packetierung der Komponenten auch spezifiziert, wie eine Komponente installiert, gestartet und gestoppt wird. Zudem sind die Verantwortlichkeiten von JBI-Komponenten klar umrissen. Es wird zwischen Binding Components und Service Engines unterschieden. Während erstgenannte die Kommunikation zur Außenwelt regeln, also beispielsweise Protokolle wie FTP, FILE oder HTTP implementieren, sind letztere für die interne Verarbeitung bestimmt. Dazu gehören beispielsweise XSLT-Transformationen oder die Implentierung von Interaktionsmustern wie Splitter oder Pipeline. Eine vollständige Liste der verfügbaren Komponenten steht auf der ServiceMix-Website zur Verfügung [1].

Apache Ode [2] stellt eine Laufzeitumgebung für BPEL4WS 1.1- und WS-BPEL 2.0-konforme Prozesse zur Verfügung. Die Integration von Ode und ServiceMix erfolgt über eine JBI Service Engine.

Prozesse

Je nachdem, in welchem Kontext eine serviceorientierte Architektur entsteht, liegt der Schwerpunkt entweder auf der Integration bestehender Systeme (EAI) oder der Abbildung von Geschäftsprozessen (BPM). Dies führt regelmäßig zu Missverständnissen. Während die BPM-Fraktion meist reflexartig mit BPEL oder BPMN als Mittel der Wahl antwortet, führen die EAI-Verfechter meist die Möglichkeiten des ESB ins Feld. Dabei führt die naive Anwendung der einen oder anderen Technologie selten zum Erfolg. In der Praxis hat es sich bewährt, zwischen Mediations- und Geschäftsprozessen zu unterscheiden. Mediationsprozesse dienen der Entkopplung von Services, indem sie zwischen dem Konsumenten und dem Anbieter vermitteln. Mediation umfasst:

- Protokollumsetzung, zum Beispiel von FTP nach HTTP
- Formatkonversion, zum Beispiel Flat-File nach XML
- Schematransformation, zum Beispiel Kunde1.xsd nach Kunde2.xsd
- Routing, zum Beispiel Auftragswert > 500 – Lieferant A sonst Lieferant B

Geschäftsprozesse dagegen modellieren Prozesse mit Relevanz zur jeweiligen fachlichen Domäne. Beispielsweise könnte ein komplexer, lang laufender Buchungsprozess über BPEL abgebildet werden. Einer der Vorteile ist die grafische Repräsentation des Prozesses und die damit verbundene einfachere Kommunikation mit den Fachbereichen. Mediationsprozesse und Geschäftsprozesse ergänzen sich somit.

Szenario

Soweit zur Theorie. Im Folgenden soll gezeigt werden, wie sich Prozesse mit Open-Source-Technologien implementieren lassen. Als Beispiel dient das fiktive Szenario einer Hotelbuchung (Abb. 1).

Im Kern steht dabei ein Geschäftsprozess, der in BPEL implementiert und in Apache Ode ausgeführt wird. Der Geschäftsprozess nutzt einen Service *tour operator*, der als POJO implementiert ist und durch die Komponente *servicemix-jsr181* an den ESB angebunden wird. Dies stellt die einfachste Methode dar, um einen Service bereitzustellen. Hier wäre es auch möglich, externe Services beispielsweise über HTTP anzubinden. Der Service *tour operator* liefert eine Liste möglicher Hotels in unterschiedlichen Preislagen. Der Geschäftsprozess ermittelt das günstigste Hotel und sendet das Ergebnis an den Aufrufer. Der Geschäftsprozess kann entweder synchron über SOAP/HTTP oder asynchron über XML/Filedrop aufgerufen werden. Die Vermittlung und Umsetzung übernimmt der Mediationsprozess im ESB.

Die Einbindung einer Rule Engine durch die Komponente *servicemix-drools* wäre ebenfalls möglich. Diese würde sich analog zu *servicemix-jsr181* als Service

Baustein	Verantwortlichkeit
Enterprise Service Bus (ESB)	Entkopplung von Services, Vermittlung
Business Process Management (BPM)	Abbildung von Geschäftsprozessen
Rule Engine	Dynamische Änderung von Regeln
Registry/Repository	Verwaltung von Artefakten
Identity Management	Authentifizierung, Autorisierung, Single Sign-On
Health Tracking / Activity Monitoring (BAM)	Überwachung

Tabelle 1: Bausteine einer SOA

Baustein	Verantwortlichkeit
Enterprise Service Bus	Apache ServiceMix [1], Codehaus Mule [15], Sun OpenESB [16]
Business Process Management	Apache Ode, JBoss jBPM [17], Intalio BPMS [14]
Rule Engine	JBoss Drools [11]
Registry/Repository	Apache jUDDI [10]
Identity Management	Internet 2 Shibboleth [9]

Tabelle 2: Open-Source-Bausteine

in den ESB integrieren. Prinzipiell ist es sekundär, welche Technologie für die Entwicklung eines Service verwendet wird. BPEL, Drools, Groovy oder POJOs sind gleichberechtigt und für den Aufrufer transparent. Der Aufruf erfolgt immer über klar definierte Schnittstellen. Somit stellt der ESB zusätzlich zur Mediationsfunktionalität eine Laufzeitumgebung für Serviceimplementierungen bereit.

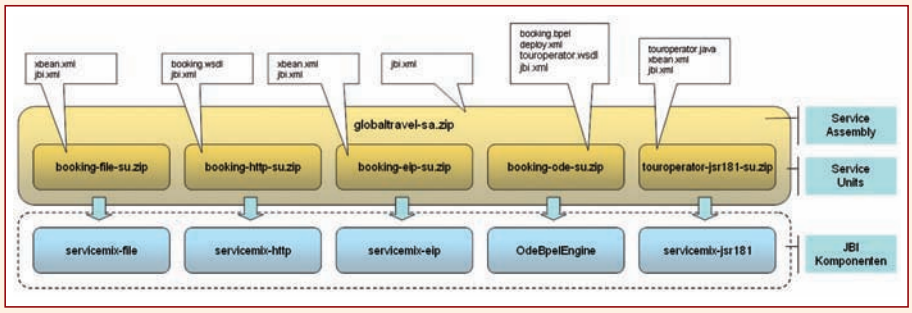

Abb. 2: JBI-Artefakte

Artefakte

Die Entwicklung von prozessbasierten Systemen erfolgt zum großen Teil deklarativ. Einzig der *tour operator*-Service ist in Java realisiert. Bei BPEL verschwimmen die Grenzen zwischen deklarativer oder imperativer Programmierung. Spätestens wenn komplexe Zuweisungen oder Transformationen entwickelt werden sollen, sind handfeste Entwicklungskenntnisse erforderlich.

Die Hauptaufgabe bei der Entwicklung besteht darin, die JBI-konformen Artefakte zu erstellen. JBI unterscheidet zwischen Komponenten, Service Units und Service Assemblies (Abb. 2).

Komponenten werden einmalig in der JBI-Laufzeitumgebung installiert. Dabei kann es sich um bereits verfügbare oder selbst entwickelte Komponenten handeln. Um eine Komponente zu nutzen, muss sie parametrisiert werden. Dies erfolgt durch eine Service Unit. Eine Service Unit ist eine ZIP-Datei mit den für eine Komponente erforderlichen Konfigurationsdateien. Diese Dateien können für jede Komponente unterschiedlich sein. Beispielsweise enthält die Service Unit *booking-ode-su.zip* unter anderem eine BPEL-Datei für den Buchungsprozess, während *booking-file-su.zip* nur die Datei *xbean.xml* enthält. Die ServiceMix-Komponenten werden über XBean [3] konfiguriert, sodass die Datei *xbean.xml* in den meisten Service Units zu finden ist. Listing 1 zeigt dies am Beispiel der Konfiguration für die Komponente *servicemix-http*.

Die Konfiguration der Endpunkte ist abhängig von der jeweiligen Komponente. Ein Endpunkt beschreibt den Einstiegspunkt für den Aufrufer innerhalb des ESB. Die Angaben *service* und *endpoint* adressieren einen Endpunkt eindeutig. Über die in Listing 1 gezeigt Konfiguration könnte beispielsweise der *touroperator service* als Web Service über HTTP bereitgestellt werden.

In der Regel bilden mehrere Service Units einen Verbund, der beispielsweise um einen Mediationsprozess abzubilden. Dieser Verbund wird in Form einer Service Assembly realisiert. Dabei handelt es sich ebenfalls um eine ZIP-Datei, die alle erforderlichen Service Units enthält. Eine Service Assembly ist die Einheit, die in den JBI-Container deployt wird. Jedes JBI-Artefakt enthält einen Deskriptor mit Namen *jbi.xml*, der zusätzliche Informationen gemäß der JBI-Spezifikation enthält. Diese Deskriptoren werden, wie später noch zu sehen sein wird, vollständig generiert, sodass der Inhalt an dieser Stelle nicht relevant ist.

Entwicklungsprozess

Als Entwicklungsumgebung bietet sich die Kombination von Maven2 für den Build-Prozess, eine Java-Entwicklungsumgebung für die Bearbeitung von Java und XML sowie ein BPEL-Designer an. Einen wirklich vollständigen, freien BPEL-Designer gibt es momentan noch nicht, vielversprechend sind aber NetBeans 5.5 inklusive Enterprise Pack [4] und das Eclipse BPEL-Designer-Plug-in. Insbesondere NetBeans verfügt über eine anspruchsvolle, BPMN-ähnliche Visualisierung, unterstützt aber noch nicht all BPEL-Merkmale, wie beispielsweise Kompensation. Dafür verfügt es über einen BPEL Mapper, der bei der Erstellung von BPEL Assigns hilfreich ist (Abb. 3).

Listing 1

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns:http="http://servicemix.apache.org/
                                        http/1.0"
   xmlns:to="http://www.pleus.net/globaltravel/
                                    touroperator">

 <http:endpoint service="to:touroperator"
        endpoint="touroperatorSOA"
        role="consumer"
        locationURI="http://0.0.0.0:8191/
                                    Touroperator/"
        defaultMep="http://www.w3.org/2004/08/
                                        wsdl/in-out"
        soap="true"
        soapAction="findHotels"/>

</beans>
```

Listing 2

```
<project>
 <modelVersion>4.0.0</modelVersion>

 <groupId>net.pleus.globaltravel</groupId>
 <artifactId>booking-ode-su</artifactId>
 <version>1.0</version>
 <packaging>jbi-service-unit</packaging>
 <name>booking-ode-su</name>

 <dependencies>
  <dependency>
   <groupId>org.apache.ode</groupId>
   <artifactId>ode-jbi</artifactId>
   <version>2.0-SNAPSHOT</version>
  </dependency>
 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.apache.servicemix.tooling</groupId>
    <artifactId>jbi-maven-plugin</artifactId>
    <extensions>true</extensions>
   </plugin>
  </plugins>
 </build>

</project>
```

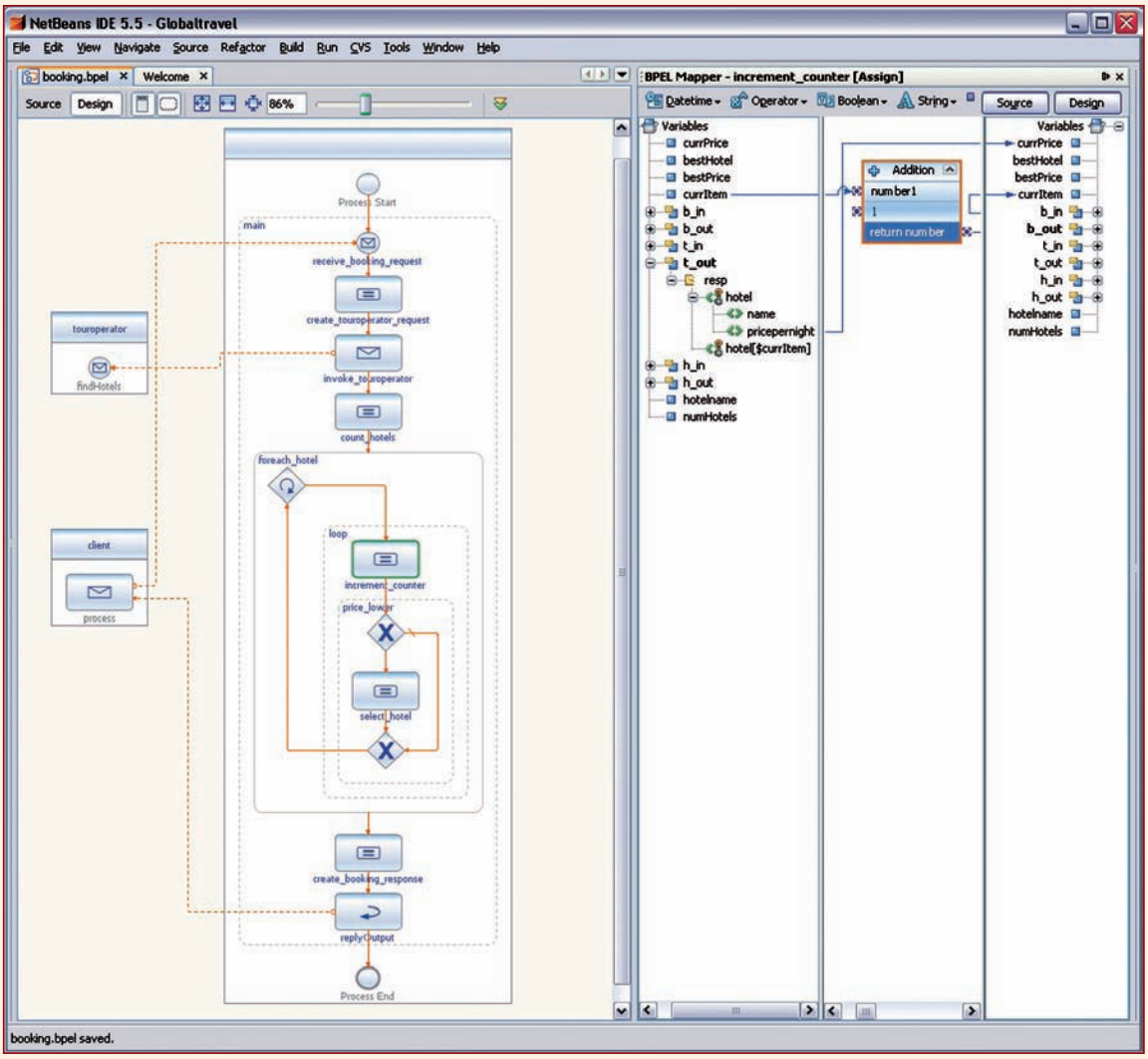

Abb. 3: NetBeans BPEL-Designer

Um den manuellen Aufwand für die Erstellung der JBI-Artefakte zu minimieren, bietet ServiceMix das Maven-Plug-in *jbi-maven-plugin*, mit dessen Hilfe sich Service Units, Service Assemblies und JBI-Komponenten inklusive der *jbi.xml*-Deskriptoren generieren lassen. Listing 2 zeigt die Maven-Datei *pom.xml* zur Generierung einer Service Unit für die Ode BPEL Engine.

Die Komponente, auf die sich die Service Unit bezieht, wird durch das *dependency*-Element angegeben. Die Angabe *<packaging>jbi-service-unit</packaging>* bewirkt die Erstellung einer Service Unit mit Namen *booking-ode-su-1.0.zip*. Um mehrere Service Units zu einer Service Assembly zusammenzufassen, kann das Maven Pom aus Listing 3 verwendet werden.

Dieses Mal wird *<packaging>jbi-service-assembly</packaging>* angegeben. Die *dependencies*-Auflistung enthält Informationen zu den enthaltenen Service Units. Listing 3 verwendet exemplarisch die Service Unit für die Ode BPEL Engine. Das Ergebnis der Ausführung des Kommandos *mvn install* ist eine Service Assembly mit Namen *globaltravel-sa-1.0.zip*.

Deployment und Management

Eine JBI-konforme Laufzeitumgebung muss Java Management Extensions (JMX) in der Version 1.2 unterstützen. Durch die bereitgestellten MBeans lassen sich die folgenden Operationen ausführen:

- Installation von Komponenten und gemeinsam genutzten Bibliotheken im laufenden Server (Hot Deployment)
- Hot Deployment von Service Units und Service Assemblies für installierte Komponenten
- Starten und Stoppen von Komponenten und Service Assemblies

Zusätzlich zu diesem Ansatz unterstützt ServiceMix auch Hot Deployment über das Dateisystem. Nach der Installation der Distribution stehen die Verzeichnisse *install* und *deploy* bereit. Standardmäßig sind dort noch keine Komponenten installiert. Diese stehen im Verzeichnis *components* bereit. Um eine Komponente zu installieren, muss sie lediglich in das *install*-Verzeichnis kopiert werden. Allgemeine Klassen sind in der Bibliothek *servicemix-shared* enthalten. Daher muss diese zuerst installiert werden.

Die Komponente *ode-jbi* für die BPEL-Integration ist nicht Teil der ServiceMix-Distribution und muss manuell erstellt werden. Eine Beschreibung dazu findet sich auf der Website von Apache Ode [2].

Service Assemblies, wie beispielsweise *globaltravel-sa-1.0.zip* aus dem Beispiel, werden durch Kopieren in das *deploy*-Verzeichnis bereitgestellt. Zusätzlich zu den JBI-konformen Merkmalen stellt ServiceMix noch ein leichtgewichtiges Verfahren für das Deployment zur Verfügung. Dabei werden Komponenten direkt in der Startkonfiguration *servicemix.xml* parametrisiert, die sich im Verzeichnis *conf* befindet. Da dieses Verfahren aber kein Hot Deployment erlaubt und nicht standardkonform ist, sollten bestenfalls Prototypen auf dieser Basis erstellt werden. Idealerweise sollte die Datei *servicemix.xml* nur die unmittelbare Konfiguration des Containers enthalten.

Durch die JMX-Fähigkeit kann jeder beliebige JMX-Client, wie beispielsweise JConsole für die Administration genutzt werden (Abb. 4).

Die Abbildung zeigt die installierten Komponenten sowie die Service Assembly und Service Units. Alle MBeans sind ebenfalls über Ant-Tasks erreichbar, die ebenfalls Teil der JBI-Spezifikation sind. Somit lässt sich der gesamte Build-Prozess inklusive Packetierung und Verteilung automatisieren.

Erfahrungen und Potenziale

Die gebotene Funktionalität ist bereits sehr leistungsfähig. Dennoch gibt es Bereiche, in denen es Verbesserungspotenzial gibt. Durch den Zugriff auf den Sourcecode lassen sich fehlende Merkmale mit vertretbarem Aufwand selbst entwickeln. Die folgenden Abschnitte zeigen dies an einigen Beispielen.

Sowohl ServiceMix als auch Ode befinden sich noch im Inkubationsstatus. Obwohl es zahlreiche Implementierungsbeispiele gibt, ist die Entwicklung nicht trivial. Oft sind es Kleinigkeiten, die die Arbeit erschweren. Da viele Fehler erst zur Laufzeit sichtbar werden, wie z.B. unterschiedliche XML-Namensräume in den Deskriptoren, sind diese nur schwer zu lokalisieren. Eine aktive Community und verschiedene Online-Foren [7], [8] stehen als Hilfe zur Verfügung. Die Reaktionszeiten sind kurz und liegen meist unterhalb einer Stunde.

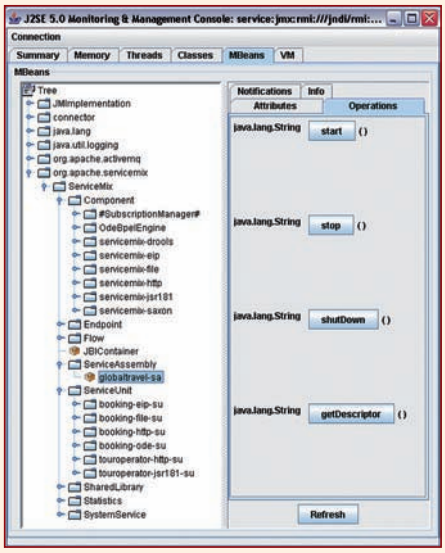

Abb. 4: JMX-Management

Aktivierung

In klassische Anwendungen, wie beispielsweise Webanwendungen, werden

Listing 3

```
<project>
 <modelVersion>4.0.0</modelVersion>

 <groupId>net.pleus.globaltravel</groupId>
 <artifactId>globaltravel-sa</artifactId>
 <version>1.0</version>
 <packaging>jbi-service-assembly</packaging>
 <name>globaltravel-sa</name>

 <dependencies>
  <dependency>
   <groupId>net.pleus.globaltravel</groupId>
   <artifactId>booking-ode-su</artifactId>
   <version>1.0</version>
  </dependency>

  ...

 </dependencies>

 <build>
  <plugins>
   <plugin>
    <groupId>org.apache.servicemix.tooling</groupId>
    <artifactId>jbi-maven-plugin</artifactId>
    <extensions>true</extensions>
   </plugin>
  </plugins>
 </build>

</project>
```

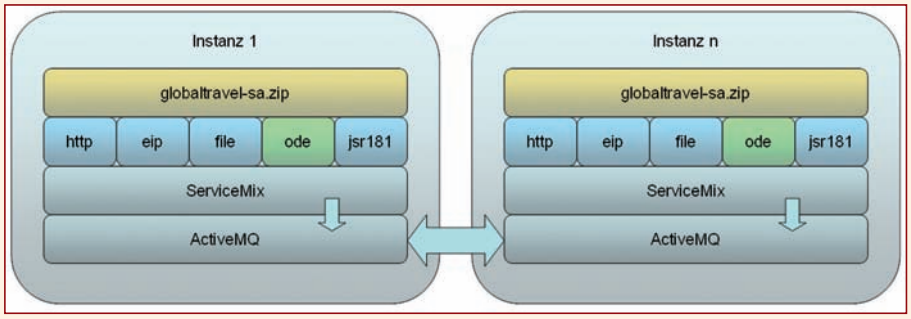

Abb. 5: Broker-Netzwerk

Prozesse häufig durch Zugriff von außen aktiviert. Dieses Muster wird auch von ServiceMix, beispielsweise durch die Binding Component *servicemix-http*, unterstützt. In prozessgetriebenen Szenarien erfolgt die Aktivierung ebenfalls von innen. Beispielsweise dann, wenn regelmäßig Daten von einem FTP Server abgeholt werden sollen. Diese Art von Prozess wird in der Regel zeitgesteuert aktiviert. Die ServiceMix-Binding Components unterstützen einen einfachen, intervallgesteuerten Mechanismus. Bei der Konfiguration einer Komponente für den Datentransfer über FTP oder das Dateisystem wird ein Intervall angegeben, wie das folgende Beispiel zeigt:

```
<ftp:poller service="test:poller"
    endpoint="endpoint"
    targetService="test:receiver"
    period="5000"
    uri="ftp://servicemix:rocks@localhost/smx/test" />
```

Für eine genauere Steuerung steht die Komponente *servicemix-quartz* zur Verfügung. Diese erlaubt die Planung komplexer Abläufe beispielsweise durch Cron-Ausdrücke. Allerdings werden Binding Components nicht durch eine Nachricht über den ESB aktiviert, sodass eine Verbindung von *servicemix-http* und *servicemix-ftp* nicht funktioniert.

Listing 4

```
<sm:container>
 <sm:listeners>
  <bean class="net.pleus.globaltravel.tracking.
                                ServicemixListener"/>
 </sm:listeners>
</sm:container>
```

Eine Lösung besteht in der Entwicklung einer kombinierten Binding Component/Service Engine, die sowohl durch eine Nachricht über den ESB aktiviert als auch Daten externe Systeme anbindet. ServiceMix stellt dafür die Basisklassen *DefaultComponent*, *ProviderEndpoint* und *ConsumerEndpoint* im Paket *org.apache.servicemix.common* zur Verfügung. Im Zusammenhang mit dem ServiceMix-Maven-Plug-in und der Angabe *<packaging>jbi-component</packaging>* beschränkt sich der Aufwand auf ein Minimum. Die Unterscheidung zwischen Binding Components und Service Engines ist eher konzeptioneller Natur. Auf technischer Ebene sind die Unterschiede marginal.

Überwachung

Health Tracking und Activity Monitoring stellen wichtige Bestandteile einer serviceorientierten Architektur dar. Beide basieren auf Informationen, die während der Prozessausführung aufgezeichnet werden. Eine vollständige, produktionsfähige Lösung bietet weder ServiceMix noch Ode an. Beide unterstützen jedoch die Registrierung eigener Listener, die bei ausgewählten Ereignissen aufgerufen werden. Dadurch lässt sich ein eigenes Tracking realisieren. ServiceMix erfordert die Implementierung der Schnittstelle *org.apache.servicemix.jbi.event.ExchangeListener*. Die JAR-Datei mit der Listener-Implementierung muss in das Verzeichnis *lib* der ServiceMix-Distribution kopiert werden. Zudem muss der eigene Listener in der Datei *servicemix.xml* registriert werden (Listing 4).

Der Listener wird bei jeder über den ESB gesendeten Nachricht aufgerufen. Aus den übermittelten Daten lassen sich der Prozessfluss und etwaige Fehler der Komponenten ableiten.

Um einen Listener für Ode zu erstellen, muss die Schnittstelle *org.apache.ode.bpel.iapi.BpelEventListener* implementiert werden. Aufgrund einer Classloader-Problematik muss der Listerner Bestandteil der JAR-Datei der Ode-Komponente werden. Die JAR-Datei mit der Implementierung wird dazu in der Datei *pom.xml* für Ode als Abhängigkeit definiert. Der Listener wird in der Datei *ode-jbi.properties* wie folgt registriert:

```
ode-jbi.event.listeners=net.pleus.globaltravel.tracking.
                                OdeListener
```

Nach der Kompilierung von Ode ist der Listener integraler Bestandteil der Komponente *ode-jbi* und wird mit ihr im ESB installiert. ServiceMix stellt einfache Listener für JDBC und Lucene im Paket *org.apache.servicemix.jbi.audit* bereit.

Produktiver Einsatz

Das wohl wichtigste Kriterium bei der Auswahl einer Technologie ist die Einsetzbarkeit im produktiven Umfeld. Hierbei sind Merkmale wie Skalierbarkeit, Hochverfügbarkeit und Lastverteilung entscheidend. ServiceMix basiert auf Apache ActiveMQ [12] als Message Broker das wiederum Apache Derby [13] als Datenbank für persistentes Messaging einsetzt. Beide Produkte erlauben eine sehr flexible Konfiguration, die sowohl eine eingebettete, als auch eine im Netz verteilte Ausführung ermöglichen. ServiceMix lässt sich sehr gut in einem Broker-Netzwerk betreiben. Dabei werden mehrere identische Knoten eingesetzt und auf der Ebene von ActiveMQ miteinander verbunden. Diese Konfiguration wird als Network of Brokers bezeichnet und lässt sich durch den Einsatz weiterer Knoten sehr gut skalieren (Abb. 5).

Durch den Einsatz identischer Knoten kann Hochverfügbarkeit bezogen auf das Broker-Netzwerk erreicht werden. Wenn einer der Knoten fehlerhaft ist, stehen immer noch die anderen Knoten für die Verarbeitung bereit. Die ausgeführten Prozesse des fehlerhaften Knotens können allerdings nicht fort-

gesetzt werden. Um Hochverfügbarkeit für einzelne Knoten zu erreichen, kann ActiveMQ in einer Master/Slave-Konfiguration betrieben werden, wobei jedem Master Broker ein oder mehrere Slave Broker zur Seite gestellt werden. Diese replizieren alle Nachrichten. Im Fehlerfall übernimmt einer der Slave Broker die Verarbeitung und wird selbst zum Master.

ActiveMQ implementiert Lastverteilung nach dem Round-Robin-Prinzip. Wenn ein Endpunkt auf mehreren Knoten in einem Cluster bereit steht, wird die Last gleichmäßig auf den gesamten Cluster verteilt. Wenn ein Endpunkt nur auf einem Knoten zur Verfügung steht, wird die Nachricht an diesen Knoten gesendet. Dadurch ist auch eine Distributed-Broker-Konfiguration realisierbar. Das bedeutet, der ESB umfasst mehrere Knoten, wobei die Knoten unterschiedliche Komponenten und ServiceAssemblies enthalten können. Ein Mediationsprozess kann so auf den gesamten Cluster verteilt werden. Das kann erforderlich sein, wenn Komponenten eingesetzt werden, die eine spezielle Hardware oder Lizenz benötigen, beispielsweise für eine Verbindung zum Mainframe. Durch einen Distributed Broker lassen sich diese Komponenten vom Rest des Clusters isolieren, wodurch Kosten gespart werden können.

Die beschriebene Lastverteilung bezieht sich auf die Verteilung innerhalb des Clusters. Ein vorgeschalteter Load Balancer wird benötigt, um die Last auf die einzelnen Knoten aufzuteilen. Für eingehende Kommunikation wie HTTP sind Verfahren, wie beispielsweise Hardware Load Balancer weitgehend etabliert.

Für ausgehende Kommunikation, wie FTP oder File, sind diese Verfahren jedoch nicht brauchbar, da diese Komponenten intern aktiviert werden. ServiceMix verfügt noch nicht über Synchronisationsmechanismen für Komponenten im Cluster, sodass in der Beispielanwendung beide File-Komponenten gleichzeitig dieselbe Datei verarbeiten würden. Eine Lösung kann darin bestehen, einen Aktiv/Passiv-Cluster für alle intern aktivierten Komponenten zu verwenden und alle anderen Komponenten in einem Aktiv/Aktiv-Cluster bereitzustellen.

Fazit

Mit Apache ServiceMix und Apache Ode stehen die Kernbausteine für die technische Implementierung einer serviceorientierten Architektur zur Verfügung. Zwar ist an einigen Stellen eigener Entwicklungsaufwand erforderlich, um fehlende Merkmale zu realisieren. Aber dennoch lohnt sich der Aufwand, da sich auf dieser Basis bereits heute eine 100% Open-Source-basierte und standardkonforme SOA-Infrastruktur realisieren lässt.

Der vollständige Quellcode zu diesem Artikel befindet sich auf der beiliegenden CD.

Wolfgang Pleus arbeitet als Technologieberater, Autor und Trainer im Bereich serviceorientierter Architekturen. Seit über10 Jahren unterstützt er internationale Unternehmen bei der Realisierung komplexer Geschäftslösungen auf der Basis von Java EE und .NET. Sie erreichen ihn unter wolfgang.pleus@pleus.net.

Links & Literatur

[1] Apache ServiceMix: incubator.apache.org/servicemix
[2] Apache Ode: incubator.apache.org/ode
[3] Apache Xbean: geronimo.apache.org/xbean
[4] Netbeans: www.netbeans.org
[5] Eclipse BPEL Designer: download.eclipse.org/technology/bpel/update-site
[6] JBI Spezifikation: jcp.org/aboutJava/communityprocess/final/jsr208/index.html
[7] Apache Ode User Forum: www.nabble.com/Apache-Ode-User-f16255.html
[8] ServiceMix User Forum: www.nabble.com/ServiceMix—User-f12050.html
[9] Shibboleth: shibboleth.internet2.edu
[10] jUDDI:ws.apache.org/juddi
[11] JBoss Rules (Drools): labs.jboss.com/portal/jbossrules/docs
[12] Apache ActiveMQ: activemq.apache.org
[13] Apache Derby: db.apache.org/derby
[14] Intalio: www.intalio.com
[15] Codehaus Mule: mule.codehaus.org
[16] Sun OpenESB: open-esb.dev.java.net
[17] JBoss jBPM: www.jboss.com/products/jbpm

Feedback

Die Redaktion freut sich über Ihr Artikel-Feedback:
feedback@javamagazin.de