Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 2995

Multi-tenant-fähige BPEL-Engines

Andreas Buchholz

Studiengang:	Softwaretechnik
Prüfer:	Prof. Dr. Frank Leymann
Betreuer:	Dipl.-Inf. Tobias Anstett
begonnen am:	30. Juli 2009
beendet am:	30. Januar 2010
CR-Klassifikation:	H.4.1, K.6.3

Inhaltsverzeichnis

Abkürzungsverzeichnis **1**

1 Einleitung **3**

2 Grundlagen **5**
2.1 Service-orientierte Architektur 6
2.2 Auslieferungsmodelle 8
2.2.1 Infrastructure as a Service 8
2.2.2 Platform as a Service 13
2.2.3 Software as a Service 14
2.2.4 Rich Internet Applications 17
2.3 Apache Tomcat 19
2.3.1 Umgebungsvariablen einer Web-Applikation 19
2.3.2 Sicherheitsmechanismen 20
2.4 Apache ODE 22
2.4.1 BPEL 23
2.4.2 Entwurf 26

3 Multi-Tenancy **37**
3.1 Motivation für Multi-Tenancy 37
3.2 Grade von Multi-Tenancy 40
3.2.1 Native Multi-Tenancy 43
3.2.2 Gemeinsamkeiten aller Ansätze 45
3.2.3 Multi-Instance Multi-Tenancy 47
3.2.4 Heterogene Lösungen 48
3.3 Nicht-funktionale Anforderungen 49
3.4 Bewertung von Multi-Tenancy-Ansätzen 53
3.4.1 Tabellarischer Vergleich 57
3.5 Szenarien 58
3.5.1 Massenmarkt 58
3.5.2 Großkunden-Sektor 59
3.5.3 Heterogene Kundenlandschaft 59
3.5.4 Sonderlösung: Einzelnes Großunternehmen 60
3.6 Umstellung auf Multi-Tenancy 61
3.6.1 Aus Sicht des Anbieters 61

3.6.2 Aus Sicht des Kunden . . . 63
3.7 In Verbindung mit BPEL . . . 64

4 Konzept & Spezifikation 67
4.1 Konzept . . . 67
4.1.1 IaaS . . . 68
4.1.2 PaaS . . . 69
4.1.3 SaaS . . . 69
4.1.4 Sicherheit . . . 70
4.2 Spezifikation . . . 70
4.2.1 Aktoren . . . 70
4.2.2 Anwendungsfälle PaaS-Modus . . . 70
4.2.3 Prozess-Management . . . 71
4.2.4 Benutzer-Management . . . 73

5 Entwurf & Implementierung 75
5.1 Verwendete Technologien . . . 75
5.2 Datenbank . . . 76
5.3 Umsetzung von Multi-Tenancy . . . 77
5.3.1 Prototyp IaaS . . . 77
5.3.2 Prototyp PaaS . . . 77
5.3.3 Prototyp SaaS . . . 80
5.3.4 Sicherheit . . . 80
5.3.5 Benutzerverwaltung . . . 80
5.4 Erweiterung der Apache ODE . . . 80
5.4.1 MultiTenancyService . . . 81
5.4.2 Web-Interface . . . 82

6 Zusammenfassung 85

A Beispiel-Prozess „DynamicInvoke“ 87

B Installationsanleitung 95

C Screenshots 97

Literaturverzeichnis 103

Abkürzungsverzeichnis

AJAX	Asynchronous Javascript and XML
AMI	Amazon Machine Image
ASP	Application Service Provider
AWS	Amazon Web Services
BPEL	Business Process Execution Language
CBP	Compiled Business Process
CRM	Customer Relationship Management
DAO	Data Access Object
DBMS	Database Management System
EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
ERP	Enterprise Resource Planning
ESB	Enterprise Service Bus
GPL2	GNU General Public License Version 2
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IaaS	Infrastructure as a Service
IDE	Integrated Development Environment
JAAS	Java Authentication & Authorization Service
JBI	Java Business Integration
JDK	Java Development Kit
JSP	Java Server Page
LOB	Line-of-Business

MTBF Mean-Time Between Failure
MTTR Mean-Time To Repair
ODE Orchestration Directors Engine
ORM Object-Relational Mapping
PaaS Platform as a Service
PKI Public Key Infrastructure
POJO Plain Old Java Object
QoS Qualities of Service
RIA Rich Internet Application
RDP Remote Desktop Protocol
S3 Simple Storage Service
SaaS Software as a Service
SCA Service Component Architecture
SCM Supply Chain Management
SLA Service Level Agreement
SMB Small and Mid-Size Businesses
SOA Service-orientierte Architektur
SOC Service-orientiertes Computing
SSH Secure Shell
UDDI Universal Description, Discovery and Integration
VPU Virtual Processing Unit
WSDL Web Service Description Language
XAML eXtensible Application Markup Language
YUI2 Yahoo! User Interface Library Version 2

Kapitel 1

Einleitung

Schon lange bevor der Begriff „Multi-Tenancy“ im IT-Bereich Fuß gefasst hat, wurde er bereits in der Immobilienbranche verwendet. Dort bezeichnet er die Nutzung eines Gebäudes durch mehrere Parteien, welche Teile der Anlage oder verschiedene Einrichtungen darin gemeinsam nutzen, anstelle eines einzelnen Mieters. Dieses Prinzip der Mehrfachnutzung findet sich aber auch in anderen Branchen, auch wenn es dort nicht explizit so genannt wird. In [Kal08] wird beispielsweise erläutert, worin sich die Ausprägungen von Multi-Tenancy bei Kreuzfahrt- und Containerschiffen unterscheiden.

In der IT steht das „Multi“ (lat. ‚viele’) dafür, dass es mehrere Benutzer eines Systems gibt und das „Tenancy“ (engl. ‚Mietverhältnis’/‚Pacht’) deutet darauf hin, dass dieses System von einem Anbieter bereitgestellt und von mehreren Parteien genutzt wird. Die Beziehung zwischen Anbieter und Kunde ist dabei vertraglich geregelt. Wenn man diese Beziehung nicht aus geschäftlicher, sondern aus technischer Sicht betrachtet, spricht man von *Tenants*. Hierbei kommt es ggf. noch zu der Unterscheidung von *Tenant-Administratoren*, die beispielsweise ein konfigurierbares Multi-Tenant-System (z. B. Anwendungen für Enterprise Resource Planning (ERP) oder Customer Relationship Management (CRM)) für ein Unternehmen einrichten, anpassen und betreuen, und *Tenant-Benutzern*, die letztendlich darauf zugreifen.

Typischerweise wird jedem Tenant eine Partition der Ressourcen bzw. Daten und eine eigene Konfiguration des Systems zur Verfügung gestellt. Hierbei kann mit System sowohl Software als auch Hardware gemeint sein. Der Tenant greift jeweils auf eine virtuelle Instanz des Systems zu, sodass eine parallele und isolierte Nutzung möglich ist. Dennoch können auch gemeinsame Ressourcen existieren. Hierbei spricht man von Tenant-übergreifenden Daten und Objekten.

Multi-Tenancy gibt es auf allen Ebenen des IT-Stacks (vgl. Abb. 1.1). Aus diesem Grund orientiert sich dieses Dokument an den Auslieferungsmodellen Infrastructure as a Service (IaaS), Platform as a Service (PaaS) und Software as a Service (SaaS), wodurch alle Ebenen abgedeckt sind.

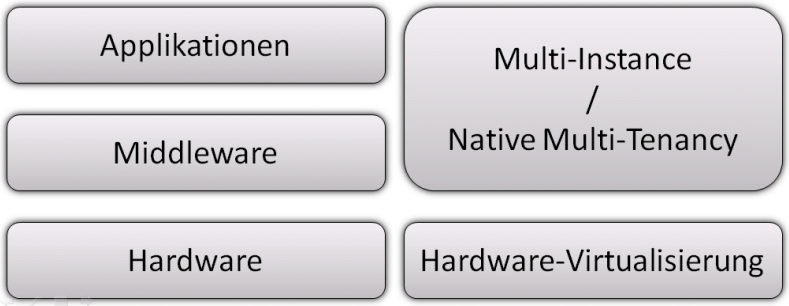


Abbildung 1.1: IT-Stack und Multi-Tenancy

In Abgrenzung zu Multi-Tenancy bezeichnet der Begriff *Single-Tenancy* (auch Single-Instance-Tenancy genannt) ein System, dessen Instanz nur einen einzigen Kunden bedient, allerdings mehrfach parallel instanziiert ist.

Die Business Process Execution Language (BPEL) ist eine auf XML basierende Beschreibungssprache für Geschäftsprozesse, durch welche eine Orchestrierung von Web Services möglich ist. Eine BPEL-Engine ist eine Middleware-Komponente, mit deren Hilfe Geschäftsprozesse ausgeführt werden können.

Wie bereits der Titel dieser Ausarbeitung anspricht, gilt es zu untersuchen, wie man eine BPEL-Engine unter Anwendung von Multi-Tenancy bereitstellen kann. Hierzu ist das Dokument in zwei Teile gegliedert: zunächst werden im theoretischen Teil (Kapitel 2 und 3) die Grundlagen erarbeitet, welche dann im zweiten, praktischen Teil (Kapitel 4 und 5) prototypisch an der BPEL-Engine Apache ODE umgesetzt werden sollen.

Kapitel 2

Grundlagen

Dieses Dokument soll eine in sich geschlossene, sich selbst erklärende Ausarbeitung sein. Da aber selbstverständlich nicht jedes Detail von Grund auf erklärt werden kann, wird allerdings auch Wissen und ein gewisser Grad an Erfahrung im Bereich der Architektur von Anwendungssystemen und des Workflow-Management vorausgesetzt.

In diesem Kapitel werden die Grundlagen gelegt, die für das Verständnis dieser Ausarbeitung ab Kapitel 3 notwendig sind. Dabei hat eine Service-orientierte Architektur (SOA) in Kapitel 2.1 fundamentale Bedeutung, weil diesbezügliches Verständnis notwendig ist, um überhaupt mit BPEL umgehen zu können.

In der Einleitung wurden bereits die drei Auslieferungsmodelle IaaS, PaaS und SaaS angesprochen, bei denen jeweils ein unterschiedlicher Multi-Tenancy-Ansatz erzielt werden kann. In Kapitel 3 werden wir sehen, auf welche Weise dieses möglich ist. In Kapitel 2.2 wollen wir zunächst jedoch die drei Auslieferungsmodelle an sich betrachten. Neben den Grundlagen werden auch konkrete Produkte und Lösungen vorgestellt, um ebenfalls ein praktisches Verständnis zu fördern.

In Kapitel 2.3 werden wir uns Apache Tomcat ansehen. Die Betrachtung umfasst Konfigurations- und Sicherheitseinstellungen, die wir später benötigen.

Anschließend erfolgt in Kapitel 2.4 eine ausführliche Betrachtung der Orchestration Directors Engine (ODE), einer BPEL-Engine, die im praktischen Teil der Diplomarbeit angepasst wird.

Mit der Ausgliederung dieser Grundlagen ist es möglich, den Fokus in Kapitel 3 ganz auf Multi-Tenancy zu legen. Weiterhin kann an dieser Stelle eine erweiterte Betrachtung der Themen stattfinden, die mitunter ein wenig über den zwingend notwendigen Rahmen hinausgeht. Hiermit soll dem Leser ermöglicht werden, einen besseren Überblick und einen guten Einstieg in ein tieferes Studium zu erhalten.

2.1 Service-orientierte Architektur

In diesem Kapitel wollen wir das Konzept einer Service-orientierte Architektur (SOA) genauer betrachten. Die Begriffe dieses Kapitels sind größtenteils aus [WCL+05] und [Erl07] entnommen. Beginnen wir also damit, einige dieser Begriffe zu erläutern.

Services sind Funktionen, die sich über eine eindeutige Netzwerkadresse aufrufen lassen. Der Aufruf kann dabei über verschiedene Transport-Protokolle geschehen, wobei wiederum unterschiedliche Nachrichtenformate zum Einsatz kommen können. Weiterhin können auch die Qualities of Service (QoS) unterschiedlich ausfallen, z. B. wenn *Reliable Messaging* verwendet wird oder nicht. Ein besonderes Charakteristikum für Services ist auch die sog. „always-on"-Semantik, welche sich am Beispiel eines Wasserhahns verdeutlichen lässt: jederzeit ist es möglich den Service „aufzudrehen", um ihn zu benutzen und ihn „zuzudrehen", nachdem man fertig ist. Wenn höhere Anforderungen existieren, lässt sich der Service mit anderen Parametern aufrufen (Wärme-/Kälte-Regler) und man erhält mehr „heißes Wasser". Der Service stellt dabei eine Abstraktionsschicht dar. Wie die Umsetzung geschieht, muss den Nutzer nicht interessieren, er darf einfach wissen, dass er jederzeit auf den Service zugreifen kann, um seinen Bedarf an „richtig-temperiertem Wasser" zu decken. Eine weitere Eigenschaft ist, dass es weder eine Initialisierungsphase noch ein pflichtgemäßes Vorgehen zum Beenden des Services gibt.

Service-orientiertes Computing (SOC) ist das Paradigma hinter Services. Ein Gedanke von SOC ist es, Funktionen zu Services zu bündeln, die sich auf leichte Art und Weise kombinieren und wiederverwenden lassen.

Service-orientierte Architektur ist ein architektureller Stil, um SOC umzusetzen. Eine SOA lässt sich an dem sog. SOA-Dreieck veranschaulichen (siehe Abb. 2.1): in einer SOA gibt es drei unterschiedliche Rollen, den Service-Anbieter, den Service-Konsumenten und das Service-Verzeichnis, welches als eine Art Vermittlungsstelle fungieren kann.

Web Service Technologie wird auch als *WS** bezeichnet, weil sich dahinter verschiedene Technologien und Standards verbergen, die alle mit *WS-* beginnen. Diese sind modular aufgebaut, lose gekoppelt und erfüllen einen spezifischen Zweck zum Aufbau und zur Unterstützung einer SOA. Als Beispiele seien hier *WS-BPEL* [AAA+06] für Workflow-Technologie und *WS-Security* für Sicherheit wie beispielsweise End-to-End-Verschlüsselung genannt.

Häufig wird eine SOA umgesetzt, die *SOAP* als Messaging-Infrastruktur verwendet. SOAP definiert zum einen ein auf XML basierendes Nachrichtenformat, womit typisierte und strukturierte Informationen ausgetauscht werden können. Zum anderen

umfasst es auch ein statusloses, uni-direktionales Nachrichtenaustausch-Paradigma. Für Korrelation, wie beispielsweise bei *Request/Reply*, muss auf Funktionen des darunterliegenden Transport-Protokolls zurückgegriffen werden bzw. diese über zusätzliche Informationen innerhalb der Nachricht erzielt werden.

Web Services können *bottom-up* oder *top-down* entwickelt werden. Bei bottom-up wird zunächst der Quelltext erstellt und anschließend mittels Tool-Unterstützung eine Beschreibung des Web Services in der Web Service Description Language (WSDL) erstellt. Bei top-down erstellt man zunächst eine WSDL-Datei und kann daraus Stubs generieren, in welchen schon die Struktur und die Rümpfe aller Funktionen existieren und nur noch mit der eigentlichen Funktionalität ausprogrammiert werden müssen.

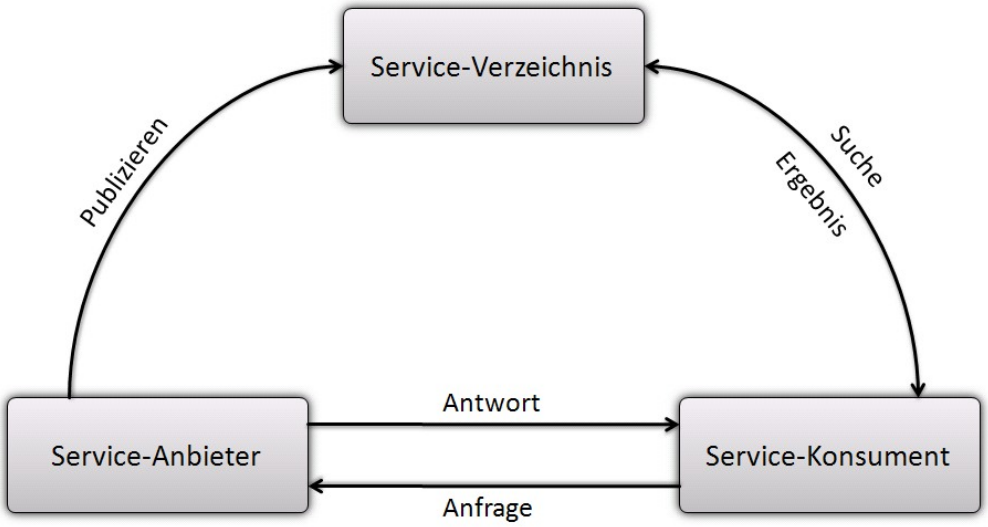


Abbildung 2.1: Grundlegender Aufbau einer Service-orientierten Architektur (vgl. [WCL+05])

Nachdem ein Anbieter einen Web Service entwickelt hat, kann er ihn in einem Service-Verzeichnis publizieren, sodass darin nach passenden Services gesucht werden kann. In den Informationen, die ein Verzeichnisdienst wie *Universal Description, Discovery and Integration (UDDI)* verwaltet, ist auch die Adresse zur WSDL-Datei des Services gespeichert. In dieser Datei findet ein Service-Konsument daraufhin alle notwendigen Informationen, um mit dem Web Service zu kommunizieren.

Für weitere, tiefergehende Informationen zur SOA-Thematik sei an dieser Stelle auf [WCL+05] verwiesen.

2.2 Auslieferungsmodelle

Dieses Kapitel über Auslieferungsmodelle ist für diese Ausarbeitung von fundamentaler Bedeutung, weil untersucht werden soll, wie BPEL mit einem Multi-Tenancy-Ansatz verknüpft werden kann und wie dieses System anschließend bereitgestellt wird, damit die einzelnen Tenants darauf zugreifen können. Die Einteilung in drei Gruppen wurde durch [ALMS09] angeregt und passt sehr gut zur Thematik dieser Arbeit, da die drei Ebenen des IT-Stacks, auf denen Multi-Tenancy eingesetzt werden kann (siehe Abb. 1.1), den Auslieferungsmodellen IaaS (Kap. 2.2.1), PaaS (Kap. 2.2.2) und SaaS (Kap. 2.2.3) entsprechen.

Diese Einteilung setzt sich in den weiteren Kapiteln fort.

2.2.1 Infrastructure as a Service

In diesem Kapitel werden wir IaaS genauer betrachten, bevor wir uns in Abschnitt 2.2.1.1 der Virtualisierung widmen, einer Technologie, die zum Betrieb von IaaS essentiell ist.

Bei IaaS wird die benötigte Server-Infrastruktur nicht wie im klassischen Fall gekauft und im eigenen Rechenzentrum betrieben, sondern die Server werden gemietet und nach Bedarf über das Internet zur Verfügung gestellt. Dieser Ansatz wird auch *On-demand-Infrastruktur* genannt, welcher neben *On-demand-Software* ein Teil des sog. *Cloud-Computing* ist. Die IaaS-Dienstleistung umfasst grundlegend nur die Hardware, für Installation und Wartung von Betriebssystem, Middleware und Anwendungen ist allein der Kunde verantwortlich. Nach [ALMS09] kann IaaS den ersten Schritt einer Outsourcing-Strategie darstellen, um sich mehr auf die eigentliche Geschäftstätigkeit zu konzentrieren.

Ein prominenter Anbieter für IaaS-Dienstleistungen ist Amazon mit den Amazon Web Services (AWS). Diese Dienste enthalten Produkte wie Elastic Compute Cloud (EC2)[1] oder Simple Storage Service (S3)[2].

Amazon EC2

Die Kernfunktionalität von EC2 besteht darin, nach Bedarf System-Images, Amazon Machine Images (AMIs) genannt, zu starten und zu stoppen. Darüber hinaus existieren aber auch Dienste wie Amazon Elastic Block Storage (EBS) zur dynamischen Erweiterung der Speicherkapazität einer AMI-Instanz oder zusätzliche Funktionalität wie dynamisch zugeordnete IP-Adressen (*Elastic IP Adresses*) oder automatisches Load Balancing (*Elastic Load Balancing*) über mehrere gestartete AMI-Instanzen hinweg.

[1] `http://aws.amazon.com/ec2/`
[2] `http://aws.amazon.com/s3/`

Amazon und auch deren Kunden stellen vorgefertigte AMIs teilweise kostenlos zur Verfügung, die sich an die eigenen Bedürfnisse anpassen, als neues AMI bündeln und speichern lassen. Anschließend können dadurch eigens erstellte Images innerhalb kürzester Zeit in beliebiger Anzahl instanziiert werden, um beispielsweise Belastungsspitzen kurzfristig begegnen zu können. Die vorgefertigten AMIs, die einen monatlichen Zusatzbetrag kosten und/oder erhöhte Gebühren für Instanz-Stunden mit sich bringen, decken dadurch auch die Kosten für Software-Lizenzen ab (vgl. Tab. 2.1). Beispiele für solche AMIs sind Windows-Images, die Amazon selbst im Repertoire hat. Zudem gibt es auch Kooperationen mit anderen Anbietern wie z. B. Wowza Media Systems[3], einem Anbieter von Streaming-Software, deren Streaming-Server on-demand über EC2 genutzt werden kann.

Instanz-Größe	Linux	Windows	Wowza Media Server Image
Small	$0,095	$0,12	$0,16
Large	$0,38	$0,48	$0,49
Extra Large	$0,76	$0,96	$0,97

Tabelle 2.1: Kosten für EC2 Instanz-Stunden in Irland

Zur Verwaltung von EC2-Instanzen bietet Amazon neben dem reinen Web Service Interface[4] die *EC2 API Tools*[5], die *EC2 AMI Tools*[6] (auch als EC2 Commandline Tools bezeichnet) und die *AWS Management Console*[7] an. Zudem existieren viele weitere Verwaltungstools anderer Anbieter, darunter auch grafische Programme wie z. B. das quelloffene *ElasticFox*[8], eine Erweiterung des Browsers Mozilla Firefox (siehe Abb. 2.2) oder die Implementierung namens „ECtoo“ aus [Bit08]. Diese Tools richten sich allerdings an Szenarien mit lediglich einem Benutzer, der allumfassende Rechte besitzt. *LabManager Cloud Edition*[9] von der Firma VMLogix ist hingegen ein Beispiel für ein System, welches u.a. eine Benutzerverwaltung und die Einrichtung von Nutzungsrichtlinien von Amazon Web Services unterstützt.

Um EC2 nutzen zu können, ist zunächst die Registrierung eines Amazon-Accounts notwendig[10]. Im Zuge dessen erhält man sowohl eine *Access Key ID* und einen *Secret Access Key* als auch ein X509-Zertifikat mit dazugehörigem privaten Schlüssel. Je nach genutzter API-Funktion können beide oder lediglich eine dieser Authentifizierungsmechanismen verwendet werden. Eine genauere Beschreibung dazu ist beispielsweise im *EC2 UserGuide*[11] enthalten. Zum Starten einer Instanz und zur

[3] http://wowzamedia.com
[4] http://s3.amazonaws.com/ec2-downloads/ec2.wsdl
[5] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=351
[6] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=368
[7] http://aws.amazon.com/console/
[8] http://developer.amazonwebservices.com/connect/entry.jspa?externalID=609
[9] http://www.vmlogix.com/Products/LabManager-Cloud-Edition/
[10] https://aws-portal.amazon.com/gp/aws/developer/registration/index.html
[11] http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/

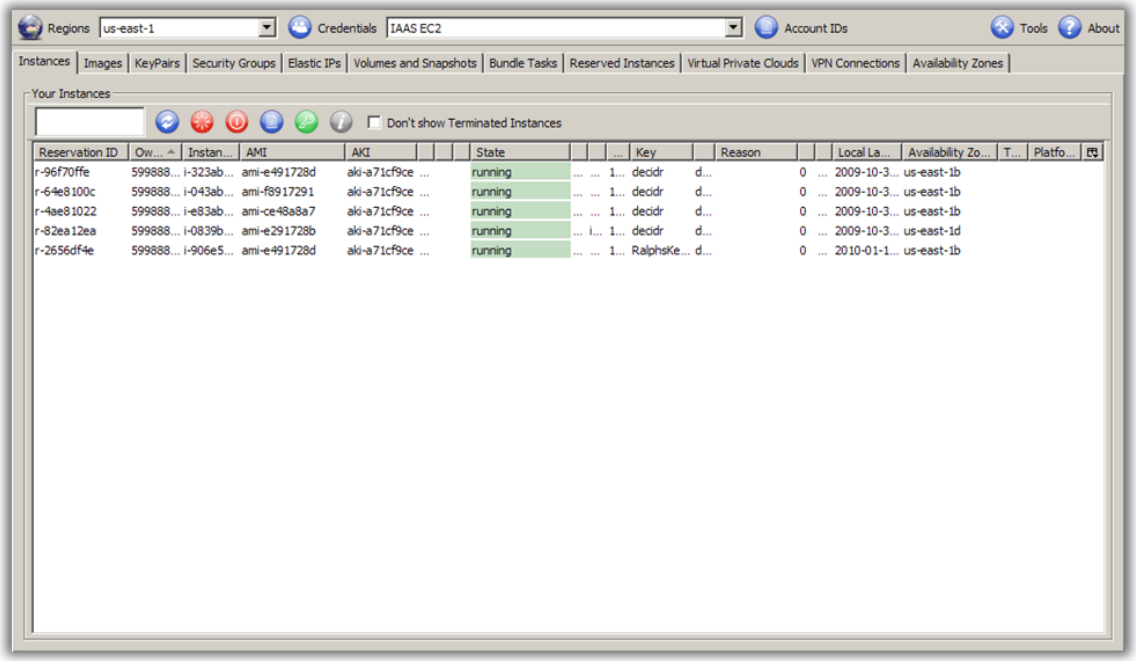


Abbildung 2.2: Browser-Plugin „ElasticFox“ für Mozilla Firefox

Verbindung per SSH wird weiterhin ein sog. *KeyPair* benötigt, welches durch die unterschiedlichen Tools erstellt werden kann. Bevor dieses KeyPair beispielsweise mit *PuTTY*[12] Verwendung findet, muss unter Verwendung des privaten Schlüssels und einer gewählten Passphrase ein „PuTTY Private Key“ (Endung `.ppk`) durch das Programm *PuTTYGen*[13] erzeugt werden.

Amazon S3

Ein weiterer wichtiger Dienst aus der Reihe der Amazon Web Services ist S3. Hierbei handelt es sich um einen skalierbaren Online-Speicher für Dateien jeglicher Art von bis zu 5GB Größe. Die Organisation der Dateien erfolgt in sog. „Buckets“ (Ordnern), welche global eindeutig benannt werden. Der Dienst lässt sich über standardisierte Interfaces (REST und SOAP) ansprechen und als Download-Möglichkeit wird grundlegend das Hypertext Transfer Protocol (HTTP) verwendet. Nach und nach werden allerdings auch andere Protokolle verfügbar gemacht, z. B. ein Interface für BitTorrent. Ein Beispiel zur Nutzung von S3 ist, dass ein Unternehmen Off-Site-Backups erstellen und dazu auf garantierte Speicherung in einer von mehreren Erdregionen (Irland, West- oder Ost-USA) zurückgreifen kann. Außerdem werden eigene AMIs in S3-Buckets gespeichert.

[12] `http://www.chiark.greenend.org.uk/~sgtatham/putty/`
[13] von demselben Anbieter wie PuTTY

2.2.1.1 Virtualisierung

Der Begriff „Virtualisierung“ wird sowohl für ein grobes Paradigma als auch für eine konkrete Lösung verwendet. Bei Virtualisierung existiert eine Komponente, welche eine Zwischenschicht zwischen Ressourcen und Nutzer bildet und somit eine Abstraktionsebene darstellt. Diese Komponente, die als Basis für sog. *virtuelle Maschinen* dient, wird auch *Hypervisor*[14] genannt. Man unterscheidet zwei Typen von Hypervisors: der Typ-1-Hypervisor setzt direkt auf der Hardware auf und benötigt deshalb eigene Treiber, während die Basis für einen Typ-2-Hypervisor ein vollständiges Betriebssystem ist, dessen Gerätetreiber mitbenutzt werden.

Weiterhin unterscheidet man zwischen Virtualisierung und Paravirtualisierung. Bei der Virtualisierung verhält sich der Hypervisor so, als ob er nicht existieren würde und emuliert die darunterliegende Hardware komplett. Dies hat zur Folge, dass die Gast-Betriebssysteme nicht angepasst werden müssen. Paravirtualisierung hingegen stellt eine Softwareschnittstelle zur Verfügung, die der eigentlichen Hardware lediglich ähnelt, aber zu dieser nicht identisch ist. Wenn die Anpassung möglich ist, wirkt sich die Paravirtualisierung leistungssteigernd auf die darauf ausgeführten virtuellen Maschinen aus.

Durch Virtualisierung ist es möglich, mehrere Instanzen gleicher oder auch unterschiedlicher Betriebssysteme auf einem physischen Rechner zu betreiben. Die Vorteile dadurch lassen sich an einigen Beispielen veranschaulichen. Erstens kann ein Anwender parallel mit unterschiedlichen Betriebssystemen arbeiten, indem er sein Hauptbetriebssystem als Host verwendet und nach Bedarf unterschiedliche Client-Systeme startet. Zweitens lässt sich eine logische Trennung zwischen Diensten oder Systemen erreichen. Auf einem physikalischen Server kann somit eine Systemumgebung jeweils für den Webserver, den Mailserver, den Fileserver etc. laufen, wodurch ein Plus an Sicherheit zu verzeichnen ist. Außerdem können auf diese Weise Server-Produkte angeboten werden, bei denen die Kunden Administrations-Rechte besitzen und dennoch nicht die Kosten für die komplette Hardware zu zahlen haben. Und Drittens lässt sich die Auslastung der Hardware durch solche Systeme optimieren.

Nachfolgend werden *VMWare* und *Xen*, zwei große Vertreter aus dem Bereich der Virtualisierungslösungen, vorgestellt. Dies dient zum einen der besseren Veranschaulichung und zum anderen werden wir im Laufe dieser Ausarbeitung über EC2 indirekt Bekanntschaft mit einer dieser Lösungen machen.

VMWare

Die Firma VMWare bietet unterschiedliche Produkte aus dem Bereich der Virtualisierung an. Das am weitesten verbreitete Produkt ist *VMWare Workstation*[15], eine

[14] von lat. *hyper* = über und *videre* = sehen ⇒ Hypervisor ≈ „Aufseher“
[15] `http://www.vmware.com/de/products/ws/`

Software, die eine komplette x86- bzw. x86-64-Architektur emuliert. Weitere Produkte für den Server-Bereich heißen *VMWare Server*[16] (kostenlos) und *VMWare Infrastructure*, ein Paket aus den Bestandteilen *VMWare ESX Server*[17] und den optionalen Komponenten *HA* für Hochverfügbarkeit und *DRS* für die Lastverteilung. Interessante Produkte sind aber auch vor allem das sog. *VirtualCenter* und *VMotion*. Durch VirtualCenter ist eine zentrale Administration und Überwachung aller ESX-Server und virtuellen Maschinen möglich, sodass recht komfortabel komplette Server-Infrastrukturen betrieben werden können. Durch VMotion ist es möglich, laufende virtuelle Maschinen zwischen ESX-Servern zu verschieben. VMWare Server ist ein Hypervisor vom Typ-2, braucht also ein Wirt-System, während der ESX Server dies als Typ-1-Hypervisor nicht benötigt, weil dieser auf einem eigenen Kernel basiert.

Xen

Xen steht als Open-Source unter der GNU General Public License Version 2 (GPL2) zur Verfügung[18]. Die Software wurde ursprünglich an der Cambridge University entwickelt, weshalb das Projekt gut dokumentiert ist und auch im akademischen Bereich behandelt wird [BDF+03, Xen10, CFH+05, ADC05]. Xen kann gleichzeitig als Hypervisor von Typ-1 und Typ-2 betrieben werden, sodass beispielsweise Linux und Windows parallel betrieben werden können. Die Architektur ist in Abb. 2.3 dargestellt. Die virtuellen Maschinen werden bei Xen als *Domänen* bezeichnet. Die Domäne 0 (kurz: Dom0) muss ein modifiziertes Gastsystem sein, welches zur Verwaltung aller auf dem System verfügbaren Domänen verwendet wird. Hierzu erhält die Domäne 0 zusätzliche Privilegien.

Für Xen existieren Verwaltungstools in verschiedenen Linux-Distributionen. Bei Debian verwendet man beispielsweise die *xen-tools*[19], eine Sammlung einfacher Perl-Skripte. Unter SuSE Linux verwaltet man die Xen-Images durch *YaST* und bei Fedora steht der sog. *Virtual Machine Manager*[20] zur Verfügung.

Xen wird beispielsweise auch von Amazon EC2 verwendet. Allerdings können dort keine Standard-Images für beliebige Xen-Installationen verwendet werden. Dennoch lassen sich die Images zunächst lokal in einer Xen-Umgebung erstellen, sodass eine einfachere Möglichkeit gegeben ist, den genauen Inhalt und die Konfiguration des Images festzulegen. Anschließend ist es notwendig, dass beispielsweise unter Unix-artigen Betriebssystem die Dateisystemtabelle `/etc/fstab` an die EC2-Umgebung angepasst wird. Wenn das AMI später öffentlich zugänglich gemacht werden soll, müssen zusätzlich Schritte unternommen werden, damit das Passwort des Administrators `root` beim Start einen Zufallswert erhält und die SSH-Konfiguration für dieses Vorhaben sinnvoll ist. In den EC2-AMI-Tools

[16] `http://www.vmware.com/de/products/server/`
[17] `http://www.vmware.com/de/products/vi/esx/`
[18] `http://www.xen.org`
[19] `http://www.xen-tools.org/software/xen-tools/`
[20] `http://virt-manager.et.redhat.com/`

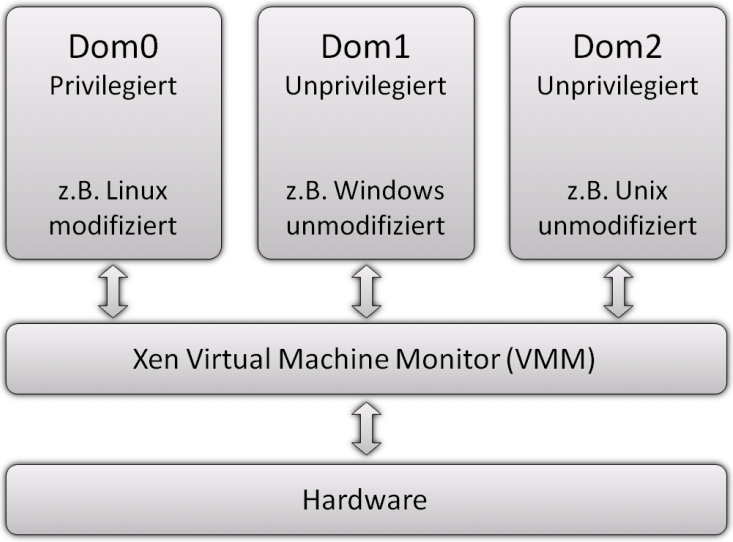


Abbildung 2.3: Xen-Architektur (vgl. [ADC05])

sind Programme enthalten, die das Verpacken eines AMIs ermöglichen. Für Public AMIs lautet der Befehl `ec2-bundle-vol`, bei anderen Images verwendet man `ec2-bundle-image`. Anschließend können die erstellten Teile des Images mittels `ec2-upload-bundle` in einen S3-Bucket kopiert werden.

2.2.2 Platform as a Service

Im Gegensatz zu IaaS versteht man unter Platform as a Service (PaaS) den Ansatz, dass bereits eine vollständige Umgebung aus Betriebssystem und anderer benötigter Software installiert ist, sodass sich Anwendungen auf dieser Plattform entwickeln und ausführen lassen und diese als Services angeboten werden.

Beispiele aus dem Bereich der PaaS-Lösungen sind *Force.com*[21] von Salesforce und Google's App Engine[22].

Force.com stellt eine Plattform dar, auf der Anwendungen mittels der eigenen, java-ähnlichen Sprache APEX entwickelt werden können.

Google's App Engine ist ein Dienst, durch den Anwendungen in verschiedenen Programmiersprachen auf der Google-Plattform entwickelt werden können. Hierzu setzt Google eine Java Runtime Environment ein, durch die Programme in Java (wie z. B. Servlets) oder auch in Interpretersprachen wie JavaScript und Ruby ausgeführt werden können.

[21] `http://force.com`
[22] `https://appengine.google.com/`

2.2.3 Software as a Service

SaaS ist mittlerweile ein häufig verwendeter Begriff, deswegen zunächst ein kleiner Überblick, in welchem vier wichtige Kernpunkte angesprochen werden, die SaaS charakterisieren:

1. Zugriff über Netzwerk

Zunächst lässt sich festhalten, dass der Zugriff auf die SaaS-Anwendung typischerweise über ein Netzwerk wie das Internet oder das Intranet eines Unternehmens erfolgt. Neben einigen Vorteilen eines solchen Client-/Server-Szenarios ergibt sich dadurch natürlich im Vergleich zu lokal installierter Software eine Abhängigkeit bzgl. des Übertragungsmediums.

2. Minimalanforderungen an IT

Die erforderliche IT-Infrastruktur, um SaaS nutzen zu können, ist minimal. In der Regel erfolgt der Zugriff mittels eines Web-Browsers über HTTP. Dieses ist vorteilhaft, weil ein solches Programm quasi auf jedem Client-Rechner weltweit verfügbar ist. Die Anwendung selbst ist meist eine Rich Internet Application (RIA) (siehe Abschnitt 2.2.4), sodass lediglich JavaScript aktiviert oder möglicherweise ein zusätzliches Plugin installiert sein muss.

3. On-demand Verfügbarkeit

Die Eigenschaften von Services umfassen unter anderem auch das Verhalten, dass sie ähnlich einer Wasser- oder Gasversorgung permanent verfügbar sind, ohne dass ein Aufwand für Konstruktion und Destruktion betrieben werden muss (vgl. Kap. 2.1).

4. Bezahlung nach Nutzungszeit

Im Vergleich zum klassischen Kauf von Software durch einmaligen Erwerb einer Nutzungslizenz sticht SaaS hervor, indem exakt nach Nutzungsdauer abgerechnet werden kann. Hierbei sind beliebige Modelle für die Abrechnung eines Abonnements möglich: nach Stunden/Tagen/Wochen, immer an Wochenenden, pro Kalendermonat, jährlich etc.

Im Umfeld von SaaS werden auch die folgenden beiden Begriffe verwendet, die um der Klarheit willen kurz erwähnt werden sollen. Der erste Begriff lautet *Applikations-Host* und bezeichnet einen Anbieter, der lediglich eine Anwendung zur Verfügung stellt, während der zweite Begriff *Application Service Provider (ASP)* einen Anbieter beschreibt, der bei seiner Dienstleistung rund um die Anwendung einen noch umfassenderen Service anbietet, indem er z. B. auch die Benutzerverwaltung übernimmt.

Wenn man sich darüber im Klaren ist, welche Charakteristika SaaS ausmachen, kann man nun einen Schritt weiter gehen und die verschiedenen SaaS-Services kategorisieren. In [Mic06a] werden die Services in zwei grundlegende Gruppen eingeordnet, die sog. *Line-of-Business Services* und *kundenorientierte Services*.

LOB-Services

Der Begriff Line-of-Business (LOB) bezeichnet ein oder mehrere Produkte, die elementare Bausteine für die Geschäftstätigkeit einer Firma darstellen. LOB Services sind meist große, anpassbare Geschäftslösungen und richten sich an Firmen

und Organisationen jeglicher Größe. Beispiele für solche Lösungen betreffen alle möglichen Bereiche der Geschäftsprozesse wie beispielsweise Supply Chain Management (SCM), CRM oder ERP. Diese Services werden üblicherweise gegen eine regelmäßige Gebühr zur Verfügung gestellt.

Kundenorientierte Services

Kundenorientierte Services hingegen richten sich an die allgemeine Öffentlichkeit und sind in der Regel eher von geringerem Umfang. Sie werden gelegentlich zwar auch gegen regelmäßige Gebühr zur Verfügung gestellt, aber meistens sind sie für die Kunden zur kostenlosen Nutzung freigegeben und werden ggf. durch Werbung finanziert.

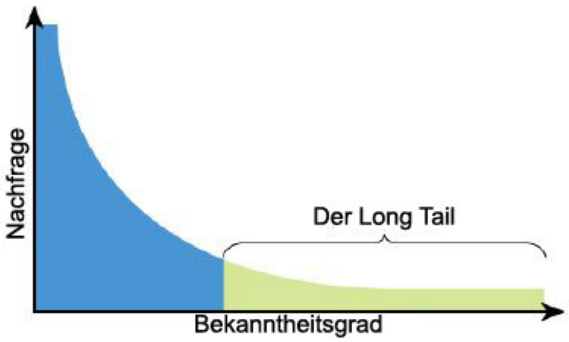


Abbildung 2.4: Graphische Veranschaulichung des sog. „Long Tail“ [Mic06a]

SaaS spielt auch in folgender Betrachtung eine wichtige Rolle: Für manche Produkte ist es nicht rentabel, wenn man den klassischen Vertriebsweg über den Einzelhandel nutzt. Unter der großen Anzahl verschiedener Produkte sind nur wenige Produkte überhaupt für den Massenmarkt konzipiert. Erst ab einer gewissen Verkaufsrate lohnt es sich, einzelne Anwendungen auf einem Datenträger im Einzelhandel anzubieten. Hierbei handelt es sich um das Problem des sog. „Long Tail“, welcher in Abb. 2.4 dargestellt wird. Der Long Tail bezeichnet also das sehr breite Produktspektrum mit relativ niedriger Nachfrage. [And04] beschreibt z. B., auf welche Weise Online-Händler wie Amazon effektiv mit dem Long Tail umgehen können. In Abb. 2.5 wird dargestellt, wie SaaS dazu beitragen kann, die Kosten zur Auslieferung von Software soweit abzusenken, dass auch der Long Tail der Software-Produkte vermarktet werden kann (vgl. [Mic06a]).

Das Ziel dieser Diplomarbeit ist es, Möglichkeiten zu analysieren, wie BPEL und Multi-Tenancy miteinander verknüpft werden können. Doch in welcher Beziehung steht SaaS hierzu? Erläutern wir doch zunächst die Beziehung zwischen SaaS und Multi-Tenancy.

Für SaaS ist es nicht zwingend notwendig, dass im Hintergrund Multi-Tenancy eingesetzt wird. Multi-Tenancy dreht sich grundlegend darum, möglichst viele Nutzer mit möglichst wenigen Ressourcen zu bedienen (für Details, siehe Kap. 3).

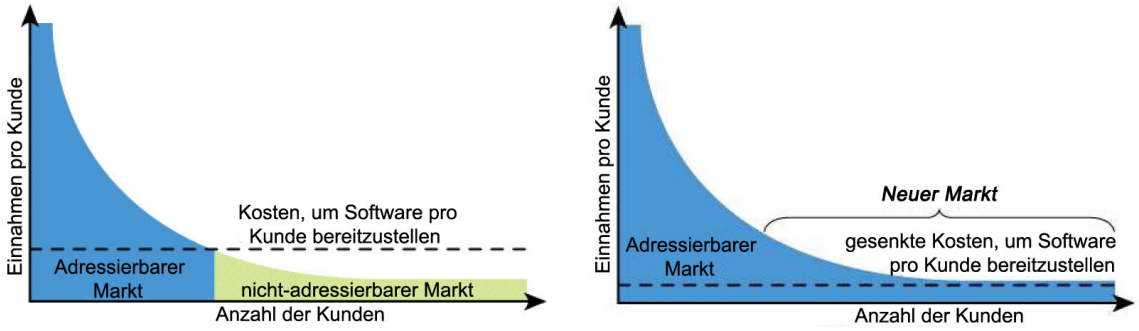


Abbildung 2.5: Einfluss von SaaS im Bezug auf den sog. „Long Tail“ [Mic06a]

Obwohl sich dieses teilweise mit dem Gedanken hinter SaaS überschneidet, dreht sich SaaS dennoch vielmehr um die technische Umsetzung, d. h. um die Art und Weise und die Randbedingungen des Zugriffs auf die Software. Beide Ansätze besitzen also unterschiedliche Schwerpunkte. Von daher kann es durchaus bestimmte Voraussetzungen geben, die einen Lösungsansatz unter Verwendung von SaaS ohne Multi-Tenancy erfordern. In vielen Fällen ist es aber dennoch so, dass SaaS erst durch Multi-Tenancy effizient und skalierbar wird (vgl. [Mic06a]). Auf der anderen Seite ergibt Multi-Tenancy auf Applikationsebene allerdings keinen Sinn, wenn sie nicht als SaaS zur Verfügung gestellt wird. Trotz der Verschiedenheit lässt sich also dennoch festhalten, dass eine enge Verknüpfung beider Ideen besteht.

Wie steht es nun um das Verhältnis zwischen SaaS und BPEL? Hierbei ist entscheidend, wie man festlegt, was als eigentliche Anwendung angesehen wird. Sieht man die BPEL-Engine als die Anwendung an, kann man die Prozessmodelle zur Konfiguration zählen. Man könnte allerdings auch die Sichtweise vertreten, dass Hardware, Betriebssystem, Middleware und die BPEL-Engine selbst zur Plattform gezählt werden und sowohl die BPEL-Prozesse als auch ggf. installierte Web Services die eigentliche SaaS darstellen (vgl. [ALMS09]. Dabei würden lediglich vorgefertigte, möglichst konfigurierbare Prozessmodelle zum Einsatz kommen. Sobald ein Tenant allerdings eigene Prozessmodelle hochladen kann, handelt es sich nicht mehr um SaaS, sondern PaaS.

Ein prominenter Vertreter im SaaS-Umfeld ist *Salesforce*[23]. Die Produktpalette dieses Anbieters umfasst sowohl LOB-Services wie das Salesforce CRM[24] als auch kundenorientierte Services, die auf dem eigenen Anwendungs-Marktplatz AppExchange[25] vertrieben oder auch kostenlos zur Verfügung gestellt werden. Auf der Plattform *Force.com* können Anwendungen außerdem auch von anderen Parteien entwickelt werden (siehe Kap. 2.2.2). Von diesem Ansatz können z. B. Startup-Unternehmen

[23] `http://salesforce.com`
[24] `http://www.salesforce.com/de/crm/`
[25] `http://sites.force.com/appexchange/home`

stark profitieren: entweder können sie ihren eigenen Bedarf an Anwendungen preislich gut kalkulierbar durch die angebotene SaaS decken oder, wenn sie selbst entwickeln wollen, gibt ihnen das vorhandene Know-How, die Infrastruktur und auch der leichtere Vertrieb über den AppExchange eine solide Basis, auf der sie operieren können.

2.2.4 Rich Internet Applications

RIAs sind SaaS-Anwendungen, die meist einen relativ großen Umfang besitzen und sich in Fragen der Handhabung an den klassischen Desktop-Anwendungen orientieren. RIAs können dabei mittels verschiedener Technologien entwickelt und bereitgestellt werden. Nachfolgend werden einige dieser Technologien angeführt, damit man sich ganz praktisch vorstellen kann, wie SaaS und RIAs funktionieren.

Adobe Flash

Adobe Flash ist eine Plattform für Multimedia-Inhalte, mit welcher sog. Flash-Filme im SWF-Format[26] erstellt werden können. Dieses Format ist in der Lage, sowohl Vektor- als auch Rastergrafiken verarbeiten zu können und bietet vielseitige Unterstützung zur Animation, Interaktion und zum bidirektionalen Streaming von Audio- und Videoinhalten. Die eingesetzte Skriptsprache *ActionScript* ist wie JavaScript und JScript ein Dialekt von ECMAScript[27].

Die typischen Anwendungsfelder von Flash waren zunächst Animationen und andere grafische Komponenten für Webseiten. Später wurde Flash auch zur Erstellung von RIAs gebraucht. Salesforce.com setzt beispielsweise auf Flash, um RIAs auf ihrer Plattform bereitstellen zu lassen[28].

Adobe FLEX

Adobe FLEX[29] richtet sich von Anfang an mehr an Programmierer als an Grafiker. Flex-Anwendungen bestehen aus MXML, einer XML-basierten Beschreibungssprache, und ActionScript. Ein spezieller Flex-Compiler übersetzt von MXML nach ActionScript, sodass im weiteren Schritt nur noch ActionScript-Quelltext vorliegt, der nach Flash übersetzt werden kann. Ausführungsvoraussetzung ist der Adobe Flash Player ab Version 9.

Microsoft Silverlight

Microsofts Ansatz für RIAs nennt sich Silverlight[30]. Hierbei handelt es sich um eine Browser-Erweiterung, die auf dem .Net-Framework basiert. Die Oberfläche einer Silverlight-Anwendung wird mit eXtensible Application Markup Language (XAML) beschrieben, während die Programmierung seit Version 2 in einer beliebigen .NET-Sprache erfolgen kann.

[26] SWF = **S**hock**w**ave **F**lash
[27] `http://www.ecmascript.org/`
[28] `http://developer.force.com/flashbuilder`
[29] `http://www.adobe.com/de/products/flex/`
[30] `http://silverlight.net/`

Java-Techologien

Auch mit Mitteln aus dem Java-Umfeld lassen sich RIAs erstellen. Ein Beispiel dafür ist die Grafikbibliothek *UltraLightClient* (ULC)[31], mit dessen Hilfe Java-Anwendungen mit Swing-Komponenten als Java Applets direkt im Browser oder per Java Web Start[32] ausgeführt werden können.

AJAX

Obwohl Asynchronous Javascript and XML (AJAX) aus technischer Sicht keine wirkliche Neuerung darstellte, wurde gerade die Kombination der zugrundeliegenden Technologien zu einem riesigen Erfolg. Das Prinzip ist einfach: Früher musste für jede Interaktion mit einer Webseite eine Anfrage an den Server gesendet werden, der diese verarbeitete und eine komplett neu gerenderte Seite an den Browser zurückgeschickt hat. AJAX sendet nun durch JavaScript im Hintergrund asynchrone Anfragen an den Server, welcher anstatt einer Seite strukturierte Daten z. B. in XML, SOAP oder JSON zurückliefert, die ebenfalls wieder über JavaScript in den Baum des Document Object Models eingefügt werden können. Konkret heißt das, dass Teile einer Seite neu geladen werden können, sodass sich das Verhalten und Aussehen immer stärker an klassische Desktop-Anwendungen anlehnt. Der Vorteil ist, dass auf quasi allen Computern die Mindestvoraussetzung für die Nutzung solcher RIAs vorhanden ist, nämlich ein einigermaßen moderner Internet-Browser mit eingeschaltetem JavaScript, ohne dass zusätzliche Plugins installiert werden müssen.

OpenLaszlo

OpenLaszlo[33] ist vor allem deshalb interessant, weil es zwei Technologien gleichermaßen verwenden kann. Es handelt sich dabei nämlich um ein Framework mit mehreren Ausgabemöglichkeiten. Anwendungen werden in Laszlo (kurz: LZX) programmiert, einer XAML-ähnlichen Sprache, die aus XML-Tags und ECMAScript-Teilen besteht. Laszlo-Anwendungen können on-the-fly aus den LZX-Quellen übersetzt werden, je nach Bedarf in die Zielformate Flash oder DHTML.

Offline-Nutzung

Eine wichtige Funktion, damit RIAs auch wirklich mit klassischen Desktop-Anwendungen konkurrieren können, ist eine mögliche Offline-Nutzung. Es wäre z. B. kaum vorstellbar, dass eine Anwendung wie *Google Docs*[34] in der Lage sein könnte, ein Office-Paket wie *OpenOffice.org* zu ersetzen, wenn man, um beispielsweise einen Brief zu schreiben, durchgehend online sein muss. In der Spezifikation von HTML5[35], die sich momentan noch in der Entwicklung befindet, ist ein Entwurf enthalten, wie eine Offline-Nutzung von Webseiten ermöglicht werden kann.

Um z. B. die Offline-Nutzung von RIAs schon vor der Fertigstellung von HTML5 und dessen Umsetzung in den Browsern verfügbar zu machen, hat Google ein Projekt namens *Gears*[36] initiiert, welches diese Funktionalität mit sich bringt. Hierzu

[31] `http://canoo.com/ulc/`
[32] `http://java.sun.com/products/javawebstart/`
[33] `http://www.openlaszlo.org`
[34] `http://docs.google.com/`
[35] `http://dev.w3.org/html5/spec/Overview.html#offline`
[36] `http://gears.google.com/`

muss ein Benutzer Gears installiert haben und kann auf einer kompatiblen Seite die Daten zwischen Server und Client synchronisieren und somit offline weiterarbeiten.

SaaS-Anwendungen sind eigentlich kaum Grenzen gesetzt. Wie bereits angesprochen, können klassische Anwendungen durch SaaS ersetzt werden. Hierzu zählen auch Integrated Development Environments (IDEs), denn Plattformen wie *force.com* bieten für ihre eigens entwickelten Sprachen Online-Editoren an, jedoch existieren auch rein web-basierte IDEs, wie z. B. *CodeRun*[37].

2.3 Apache Tomcat

Bei Apache Tomcat handelt es sich um einen unter der Apache License 2.0 stehenden Servlet-Container. Die Software besteht aus drei Hauptkomponenten: dem Servlet Container *Catalina*, dem HTTP Connector *Coyote*, welcher auf einem bestimmten HTTP-Port auf Verbindungen wartet, um die Anfragen an die Engine weiterzuleiten, und der JSP-Engine *Jasper*, welche eine Java Server Page (JSP) in ein Servlet kompiliert, damit diese anschließend von Catalina ausgeführt werden kann.

Tomcat wird als Container verwendet, in welchem die BPEL-Engine Apache ODE (siehe Kap. 2.4) deployt werden kann. In diesem Kapitel werden wir einige Besonderheiten dieser Software betrachten, welche wir für die spätere Implementierung benötigen. Für weitere Informationen zu Tomcat wird auf die Dokumentation im Internet verwiesen[38].

2.3.1 Umgebungsvariablen einer Web-Applikation

Im Deployment-Deskriptor einer Web-Applikation können Umgebungsvariablen gesetzt werden, die anschließend in der Anwendung verfügbar sind. Die Definition geschieht in der Datei `web.xml` nach folgendem Schema:

In den Servlets und JSPs wird dann wie folgt auf die Umgebungsvariablen zugegriffen:

[37] `http://www.coderun.com/`
[38] `http://tomcat.apache.org/tomcat-5.5-doc/index.html`

```
<web-app>
  <env-entry>
    <env-entry-name>path</env-entry-name>
    <env-entry-value>/foo/bar/test</env-entry-value>
    <env-entry-type>java.lang.String</env-entry-type>
  </env-entry>
  ...
</web-app>
```

Listing 2.1: Definition einer Umgebungsvariablen im Deployment-Deskriptor

```
Context env = (Context) new InitialContext().lookup("java:comp/env");
String docBase = (String) env.lookup("path");
```

Listing 2.2: Zugriff auf Umgebungsvariable einer Web-Applikation

2.3.2 Sicherheitsmechanismen

Der beliebte Authentifizierungsmechanismus für den Schutz von Ressourcen mittels einer `.htaccess`-Datei (Basic Authentication) steht bei Apache Tomcat nicht zur Verfügung. Ein vergleichbarer Mechanismus lässt sich aber erzielen, indem man den Deployment-Deskriptor `web.xml` einer Webapplikation anpasst (siehe Listing 2.3. Dort lässt sich ein `security-constraint` definieren, der in Verbindung mit einem `login-config`-Eintrag zur Absicherung der angegebenen Ressourcen beiträgt. Darin stehen über den Eintrag `<auth-method>...</auth-method>` vier verschiedene Methoden zur Verfügung: *HTTP Basic Authentication*, *HTTP Digest Authentication*, *Form Based Authentication* und *HTTPS Client Authentication*.

In Listing 2.3 wird allerdings nur auf Rollen verwiesen, die an anderer Stelle definiert sein müssen. Dies geschieht durch einen sog. *Realm*. Die Angabe des *Realms* legt fest, woher Tomcat Benutzername mit Passwort und auch Rollen für die Authentifizierung von Benutzern auslesen kann. Der Standard ist der sog. „MemoryRealm“, der allerdings nicht für Produktionsumgebungen vorgesehen ist, weil die Zugriffsberechtigungen lediglich beim Start von Tomcat aus der Datei `$CATALINA_HOME/conf/tomcat-users.xml` ausgelesen werden und im laufenden Betrieb nicht verändert werden können. Listing 2.4 zeigt solch einen beispielhaften Eintrag in der *tomcat-users.xml*).

Neben dem MemoryRealm bietet Tomcat noch weitere Realms an, die nachfolgend aufgelistet sind:

- *JDBCRealm*: Verwendung einer relationalen Datenbank, die über einen JDBC-Treiber angesprochen wird.

```
<web-app>
        ...
        <security-constraint>
                <web-resource-collection>
                        <web-resource-name>admin-files</web-resource-name>
                        <url-pattern>/admin/*</url-pattern>
                </web-resource-collection>
                <auth-constraint>
                        <role-name>admin</role-name>
                </auth-constraint>
        </security-constraint>
        <login-config>
                <auth-method>BASIC</auth-method>
                <realm-name>Administration</realm-name>
        </login-config>
        <security-role>
                <role-name>admin</role-name>
        </security-role>
</web-app>
```

Listing 2.3: Absicherung von Ressourcen bei Apache Tomcat

```
...
<role rolename="admin"/>
<user username="andreas" password="secret" roles="admin"/>
...
```

Listing 2.4: Rollen und Benutzer beim MemoryRealm

- *DataSourceRealm*: Verwendung einer relationalen Datenbank, die über einen JNDI JDBC DataSource-Namen angesprochen wird.
- *JNDIRealm*: Verwendung eines LDAP-basierten Verzeichnisdienstes, auf welchen mit Hilfe eines JNDI-Provider zugegriffen wird.
- *JAASRealm*: Zugriff auf die Benutzerdaten per Java Authentication & Authorization Service (JAAS)

Realms können drei verschiedenen Gültigkeitsbereichen zugewiesen werden, je nachdem, an welcher Stelle sie definiert werden:

- Innerhalb eines `<Engine>`-Elements für alle Web-Applikationen und alle Virtual-Hosts, es sei denn, durch ein eingebettetes `<Host>` oder `<Context>` eingeschränkt.
- Innerhalb eines `<Host>`-Elements für alle Web-Applikationen dieses konkreten Virtual-Hosts, es sei denn, durch ein eingebettetes `<Context>` eingeschränkt.

- Innerhalb eines <Context>-Elements für genau diese Web-Applikation.

```
<Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
       driverName="org.gjt.mm.mysql.Driver"
       connectionURL="jdbc:mysql://servername/dbname"
       userTable="users" userNameCol="user_name" userCredCol="user_pass"
       userRoleTable="user_roles" roleNameCol="role_name"/>
```

Listing 2.5: Definition eines JDBCRealm

Um einen JDBCRealm einzusetzen, benötigt man eine Realm-Definition wie z. B. in Listing 2.5 und ein entsprechendes Datenbank-Schema. Mit den drei SQL-Befehlen aus Listing 2.6 lässt sich ein solches Schema einrichten.

```
CREATE TABLE `user_groups` (
  `group_id` int(11) NOT NULL AUTO_INCREMENT,
  `group_name` char(24) DEFAULT NULL,
  `parent_id` int(11) NOT NULL DEFAULT '-1',
  PRIMARY KEY (`group_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8 AUTO_INCREMENT=1 ;

CREATE TABLE `user_roles` (
  `user_name` varchar(32) NOT NULL,
  `role_name` varchar(32) NOT NULL,
  PRIMARY KEY (`user_name`,`role_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

CREATE TABLE `users` (
  `user_name` varchar(32) NOT NULL,
  `user_pass` varchar(32) NOT NULL,
  `user_groupid` int(11) NOT NULL DEFAULT '-1',
  PRIMARY KEY (`user_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Listing 2.6: Datenbankschema für einen JDBCRealm

2.4 Apache ODE

Dieses Kapitel setzt sich mit der Apache Orchestration Directors Engine (ODE) auseinander, einer BPEL-Engine, die mit dem OASIS-Standard WS-BPEL 2.0 und gleichzeitig auch mit der alten Spezifikation BPEL4WS 1.1 umgehen kann. Diese Untersuchung ist insofern von großer Wichtigkeit, weil ein Hauptaspekt dieser Diplomarbeit darin besteht, wie bei der Bereitstellung von BPEL-Prozessen durch die Apache ODE Multi-Tenancy zum Einsatz kommen kann. Weil dies wahrscheinlich eine Anpassung der Engine bedeutet, lohnt sich eine genaue Analyse der Architektur und der Funktionen dieser BPEL-Engine. Bevor wir uns allerdings

den Aufbau und Umfang der Apache ODE anschauen, wollen wir uns zunächst in Abschnitt 2.4.1 der Business Process Execution Language (BPEL) widmen, um später besser beurteilen zu können, worin die Funktionalität der Engine besteht.

2.4.1 BPEL

BPEL steht für Business Process Execution Language und umfasst mehrere Aspekte. Zunächst ist BPEL eine auf XML basierende Beschreibungssprache von Geschäftsprozessen, wodurch sich Web Services orchestrieren lassen und die Prozesse auch selbst als Web Services ansprechbar sind. Somit handelt es sich bei BPEL auch um ein rekursives Aggregationsmodell für Web Services.

In Abb. 2.6 lässt sich erkennen, dass BPEL modular aufgebaut ist. Der *BPEL Core* ist die Basis für Erweiterungen wie *Abstract BPEL*, *Executable BPEL* oder *BPEL4People*. Abstrakte Prozesse, wie durch Abstract BPEL eingeführt, sind in der Regel nicht ausführbar und dienen lediglich als „Absprache“ oder „Vertrag“ über die Kommunikation zwischen den Partnern. Durch Executable BPEL lassen sich Prozessmodelle spezifizieren, welche in allen Umgebungen ausgeführt werden können, die den Standard unterstützen. Durch BPEL4People wiederum können in den Prozessen auch Menschen als Kommunikationspartner auftreten.

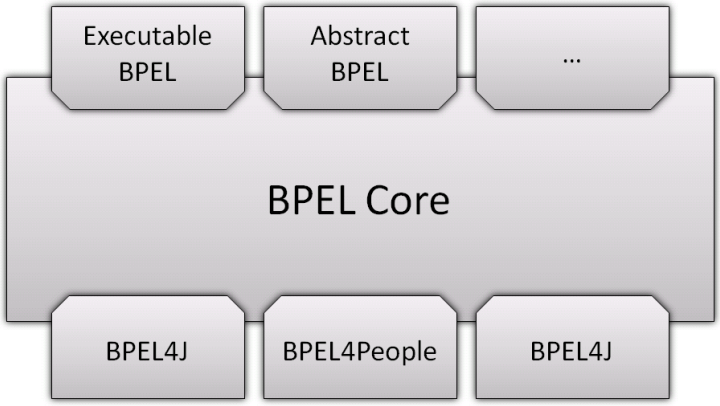


Abbildung 2.6: Architektur von BPEL

2.4.1.1 Bestandteile eines BPEL-Prozesses

Eine Geschäftsprozessdefinition in BPEL beinhaltet folgende fünf Elemente:

Partner Links sind die konkrete Umsetzung eines PartnerLink-Typs. Dieser stellt gewissermaßen einen „Kanal“ dar, über den Nachrichten ausgetauscht werden können, indem darin bis zu zwei Port-Typen miteinander verknüpft werden, welche als Rollen angesehen werden.

Variablen bieten die Möglichkeit zur Speicherung von Daten im Prozess. Die Zuweisung geschieht innerhalb einer `<assign>`-Aktivität.

Correlation Sets dienen der eindeutigen Zuordnung von Nachrichten zu einer bestimmten Prozessinstanz, z. B. wenn es mehrere Instanzen eines langlebigen Prozesses gibt. Bsp.:

```
<correlationSets>
       <correlationSet name="MyCorrelationSet"
           properties="wsdl:orderId" />
</correlationSets>
```

In der dazugehörigen WSDL-Datei werden dann Properties und PropertyAliase definiert. Bsp.:

```
<bpws:property name="orderId" type="xsd:string" />
<bpws:propertyAlias messageType="tns:itemData"
       part="orderId" propertyName="tns:orderId" query="" />
```

Handler sind Behandlungsroutinen, die einem `<scope>` zugeordnet sind. Es existieren drei Arten von Handlern:

Event-Handler werden installiert, wenn ein Scope aktiviert wurde. Events können dann parallel und mehrfach ausgeführt werden, bis der Handler zusammen mit dem umschließenden Scope deaktiviert oder terminiert wird.

Ein *Fault-Handler* setzt ein, wenn die normale Ausführung des Prozesses innerhalb des Scopes nicht weitergehen kann. Selbst wenn der Fault-Handler erfolgreich ausgeführt werden konnte, wird innerhalb dieses Scopes nicht mehr weitergearbeitet. Ist kein expliziter Fault-Handler angegeben, existiert ein Default Fault-Handler mit spezifischem Verhalten.

Compensation-Handler dienen der Umkehrung eines bereits vollständig ausgeführten Scopes und können lediglich durch den Fault-Handler des selben Scopes oder durch den Compensation-Handler des unmittelbar übergeordneten Scopes ausgelöst werden.

Activities lassen sich in zwei Kategorien aufteilen. Zum einen gibt es die sog. *Basic Activities*, welche gewissermaßen atomare Aktivitäten sind, weil sie keine weiteren Aktivitäten beinhalten. Zu dieser Kategorie gehören `<assign>`, `<empty>`, `<invoke>`, `<receive>`, `<reply>`, `<throw>` und `<wait>`. Zum anderen gibt es auch die *Structured Activities*, wodurch sich komplexere Prozessmodelle

erzeugen lassen. Zu dieser Kategorie gehören `<compensate>`, `<flow>`, `<pick>`, `<scope>`. `<sequence>`, `<switch>`, `<terminate>` und `<while>`.

2.4.1.2 Beispiel-Prozess

In Abb. 2.7 ist ein beispielhafter BPEL-Prozess dargestellt (siehe Kap. A für komplettes Listing). Als übergeordneter Scope existiert darin eine `<sequence>`-Aktivität mit dem Namen „main“. Diese beinhaltet zunächst eine `<receive>`-Aktivität namens „receiveInput“, gefolgt von einer `<if>`-Aktivität, die je nach Auswertung der Bedingung ein bestimmtes `<assign>` ausführt. Der Wert dieser temporären Variable wird danach in einer weiteren <assign>-Aktivität einer Variable zugewiesen, die dafür verantwortlich ist, welcher Service bei der `<invoke>`-Aktivität namens „DynamicHelloWorldInvoke“ aufgerufen wird. Anschließend geschieht noch ein `<assign>`, bevor die Antwort dieser Sequenz mittels einer `<reply>`-Aktivität zurückgegeben wird.

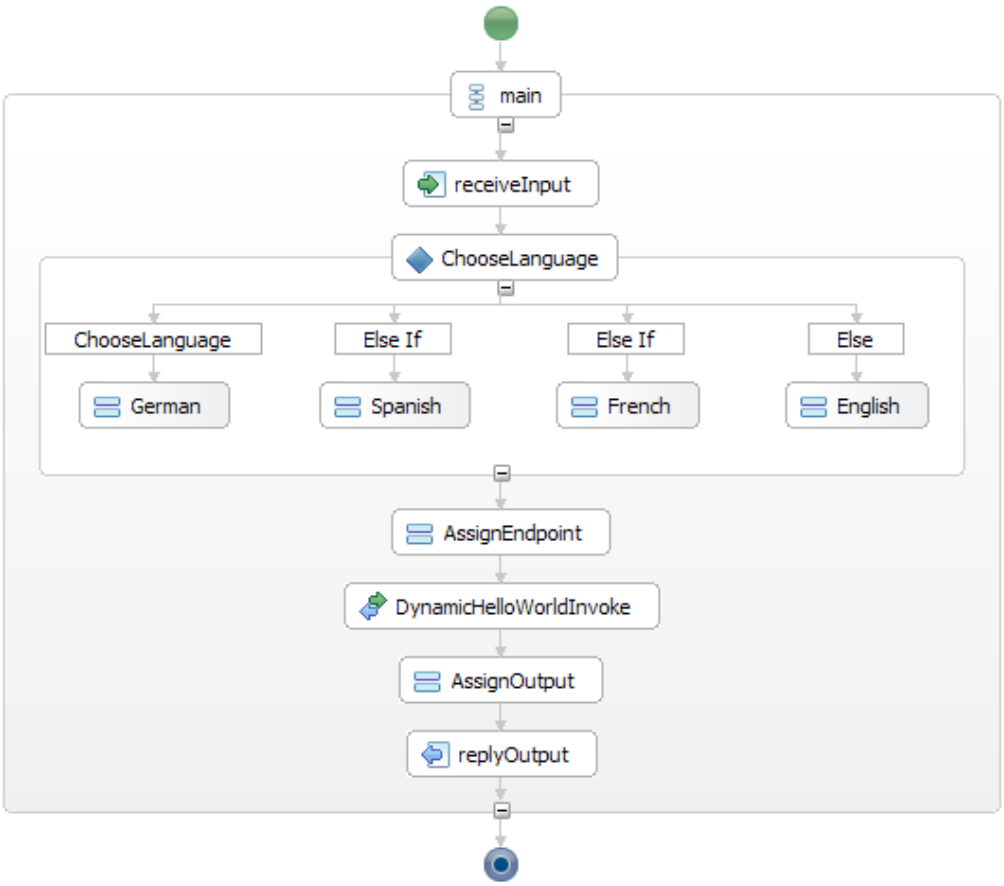


Abbildung 2.7: Beispielprozess „DynamicInvoke“

2.4.2 Entwurf

In Abb. 2.8 ist die Architektur der Apache ODE dargestellt. Neben den reinen Bestandteilen lässt sich auch erkennen, in welcher Beziehung diese zueinander stehen. Nachfolgend werden wir uns die Bestandteile einzeln anschauen.

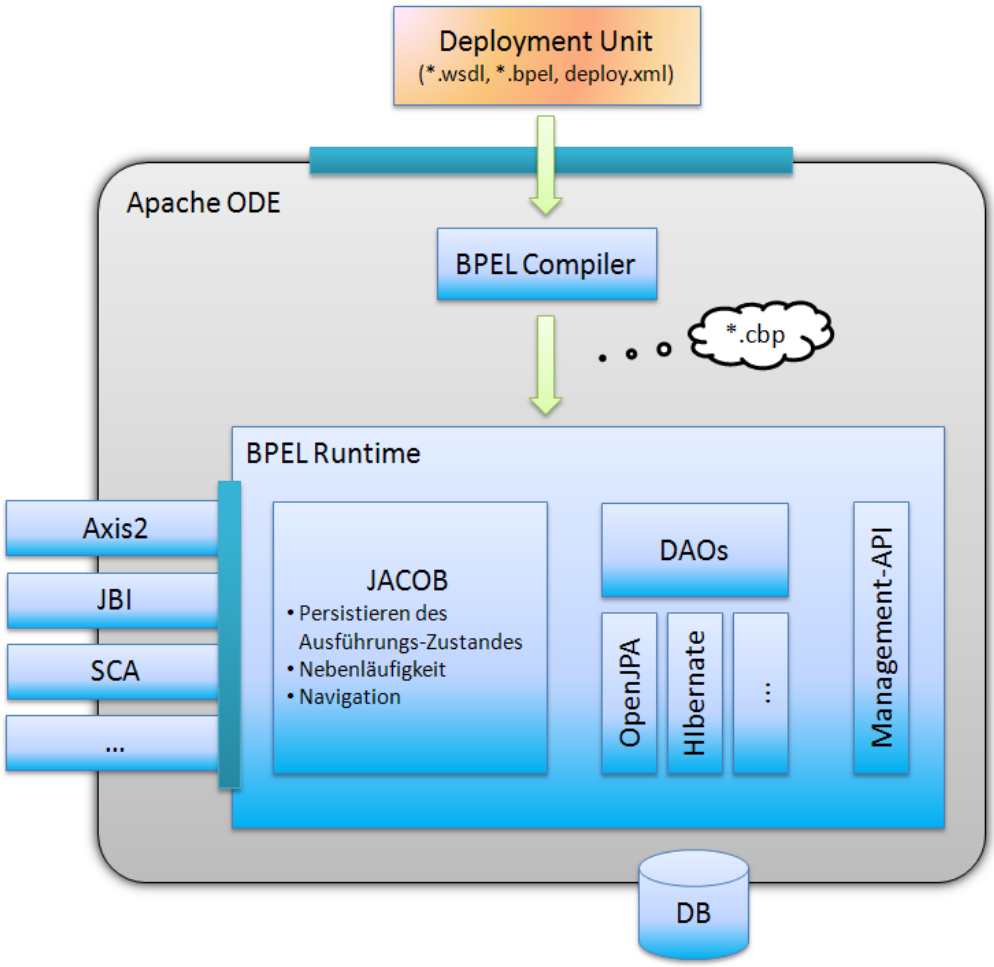


Abbildung 2.8: Architektur der Apache ODE (vgl. [Gör09])

2.4.2.1 BPEL-Compiler

Nachdem die Apache ODE korrekt installiert und gestartet wurde, ist der nächste Schritt, um tatsächlich mit dieser Workflow-Engine zu arbeiten, dass Prozessmodelle in der Engine deployt werden. Sowohl das Deployment als auch das Undeployment können während der Laufzeit der Apache ODE durchgeführt werden. Hierzu wird eine Deployment-Unit bestehend aus einem Deployment-Descriptor

`deploy.xml` und den `*.wsdl`- und `*.bpel`-Dateien vom Compiler verarbeitet und in eine Struktur aus Java-Objekten überführt, das sog. *OModel*. Dieses wird anschließend serialisiert und in eine `.cbp`-Datei (CBP = Compiled Business Process) geschrieben, damit der Prozess nicht bei jedem Start der Engine von neuem kompiliert werden muss.

2.4.2.2 Integration Layer

Der Integration Layer ist für die Kommunikation der Apache ODE mit der „Außenwelt" verantwortlich. Die Engine muss sowohl Web Services aufrufen können, als auch selbst an einem Endpunkt lauschen, um in der Lage zu sein, Nachrichten zu empfangen.

Die Anbindung kann mit den zur Zeit mit Apache ODE ausgelieferten Mitteln über Axis2[39], Java Business Integration (JBI) [THW05] und Service Component Architecture (SCA) [SOA09] erfolgen, wobei Axis2 in der Standardkonfiguration voreingestellt ist.

Apache Axis2 (Apache eXtensible Interaction System Version 2) ist ein in Java geschriebenes Framework für Web Services. Der Kernbestandteil ist ein SOAP-Server, welcher sowohl als Standalone-Server als auch innerhalb eines Servlet-Containers, wie z. B. Apache Tomcat, bereitgestellt werden kann.

Axis2 kann leicht durch Plugins erweitert werden, um weitere Funktionalität hinzuzufügen. Beispielsweise gibt es solche Module für verschiedene Elemente aus dem WS-*-Stack, wie z. B. WS-ReliableMessaging (durch Apache Sandesha2[40]), WS-Coordination und WS-AtomicTransaction (durch Apache Kandula2[41]), WS-Security (durch Apache Rampart[42]) und WS-Addressing (bereits im Axis2-Core enthalten).

In dem Framework sind weiterhin Tools enthalten, die dem Programmierer bei der Code-Generierung viel Arbeit abnehmen. Das Programm `wsdl2java` erstellt bei Übergabe einer WSDL-Datei wahlweise Stubs auf der Server- oder auf der Client-Seite.

Die Implementierung der Kommunikation per JBI geschieht durch ServiceMix[43], einen quelloffenen Enterprise Service Bus (ESB). Für die Integration per SCA arbei-

[39] `http://ws.apache.org/axis2/`
[40] `http://ws.apache.org/sandesha/sandesha2/`
[41] `http://ws.apache.org/kandula/`
[42] `http://ws.apache.org/axis2/modules/rampart/1_2/security-module.html`
[43] `http://servicemix.apache.org/`

tet Apache ODE mit einem weiteren Apache-Projekt namens „Tuscany“[44] zusammen.

2.4.2.3 JACOB

Die BPEL-Implementierung der Apache ODE basiert auf einem Framework namens *JACOB* (**JA**va **C**oncurrent **OB**jects). Dahinter verbergen sich Komponenten, die sich um die folgenden Aufgaben der Apache ODE bei der Implementierung von BPEL-Konstrukten kümmert:

1. Persistierung des Ausführungszustands
2. Nebenläufigkeit

BPEL-Konstrukte werden in der Apache ODE so umgesetzt, dass deren Implementierung sich auf die BPEL-spezifische Funktionalität konzentrieren können. Das JACOB-Framework kümmert sich darüber hinaus um die notwendige Infrastruktur. Nachfolgend werden die einzelnen Bestandteile dieses Frameworks beschrieben.

Channel sind Inferfaces, durch welche Activities miteinander kommunizieren können. Es existieren verschiedene Typen von Channeln, jeweils mit selbsterklärenden Namen, z. B. *TerminationChannel*, *ParentScopeChannel* oder *CompensationChannel*.

Alle Activities werden von der Klasse *JacobRunnable* abgeleitet, welche wiederum von *JacobObject* abgeleitet ist. JacobObject und JacobRunnable stellen eine Implementierung einer Methode dar, die durch die `run()`-Methode ausgeführt wird. Die Implementierung einer Activity überschreibt diese `run()`-Methode.

Message Lists (MLs) sind quasi das „andere Ende“ eines Channels, welches man auch als eine Art Listener ansehen kann. Dieser feuert aber nicht sofort, wenn eine Channel-Methode aufgerufen wird, sondern erst dann, wenn die JACOB-Engine diesen Aufruf vom Ausführungsstapel herunternimmt. Durch diesen Ausführungsstapel und dadurch, dass sich die einzelnen Activities nicht direkt aufrufen, wird die Nebenläufigkeit von JACOB erzielt.

In der Virtual Processing Unit (VPU) läuft die ganze Verarbeitung durch JACOB ab. Die sog. *ExecutionQueue* ist ein Container, um alle von der VPU verwalteten Artefakte in Queues zu halten.

Für weitere Details und Beispiele sei hiermit auf die Apache ODE Dokumentation[45] verwiesen.

[44] `http://tuscany.apache.org/`
[45] `http://ode.apache.org/jacob`

2.4.2.4 Data Access Objects

Data Access Objects (DAOs) (vgl. [SUN03]) sind das Bindeglied zwischen der BPEL-Engine und dem Daten-Layer. Typischerweise befindet sich im Daten-Layer eine relationale Datenbank, die per JDBC angesprochen wird. Für diesen Fall sind DAOs durch die OpenJPA-Bibliothek[46] implementiert.

Hibernate ist ein Java-Framework für Object-Relational Mapping (ORM). Durch Konfiguration können bestimmte SQL-Dialekte verwendet werden, damit Plain Old Java Objects (POJOs) mit ihren Attributen in der entsprechenden relationalen Datenbank abgelegt und später wieder ausgelesen werden können. Weiterhin können auch die Beziehungen zwischen den Objekten persistiert werden.

MemDao ist eine Implementierung der BPEL DAO-Interfaces, die komplett im Hauptspeicher arbeitet und nicht auf Festspeicher persistiert wird. In Fällen, in denen Verlässlichkeit nicht notwendig ist (z. B. in Test-Umgebungen), kann hierdurch ein Performance-Plus erzielt werden.

2.4.2.5 BPEL-Runtime

Die BPEL-Runtime ist das Herzstück der Apache ODE und ist für die Ausführung der kompilierten Geschäftsprozesse (*.cbp-Dateien) zuständig. Im Runtime-Modul befinden sich auch die Implementierung der einzelnen BPEL-Konstrukte und der Prozess-Management-API (siehe Kap. 2.4.2.6). Weiterhin stellt die Runtime die Logik zur Verfügung, die darüber entscheidet, wann eine neue Prozessinstanz gestartet wird und an welche Instanzen eingehende Nachrichten geroutet werden.

Bei der Apache ODE gibt es sowohl implizite als auch explizite Korrelation, wie eine eingehende Nachricht einer Prozessinstanz zugeordnet werden kann. Explizit heißt, dass über den Standard-BPEL-Mechanismus mittels eines <*correlation*>-Elements die Korrelation hergestellt werden kann. Implizite Korrelation kann sowohl zwischen zwei Prozessen (wenn derselbe PartnerLink benutzt wird) als auch zwischen einem Prozess und einem Web Service stattfinden. Für Beispiele und weitere Details wird auf die Apache ODE Dokumentation[47] verwiesen

2.4.2.6 Management-API

Die Apache ODE verfügt über eine Management-API, bestehend aus drei Web Services, über die sich die Prozesse und Prozessinstanzen verwalten lassen. Nachfolgend sind die einzelnen Operationen der Web Services aufgelistet. Eine genaue

[46] `http://openjpa.apache.org/`
[47] `http://ode.apache.org/implicit-correlations.html`

Analyse aller API-Funktionen ist deswegen interessant, weil man einen Überblick haben sollte, welche Möglichkeiten man überhaupt hat und wie sich die vorhandenen Funktionen für Multi-Tenancy nutzen und kombinieren lassen.

Deployment

Der *DeploymentService* stellt die folgenden Methoden zur Verfügung:

deploy: Erwartet einen Namen und ein Zip-Archiv als Parameter und liefert bei Erfolg eine `deployUnit`-Beschreibung zurück. Das Zip-Archiv muss so aufgebaut sein, dass die WSDL- und BPEL-Dateien direkt im Wurzelverzeichnis abgelegt sind. Der Name des Archivs hingegen spielt keine Rolle.

undeploy: Erwartet einen Package-Namen als Parameter und gibt einen Bool'schen Wert zurück, der über den Erfolg der Methode Auskunft gibt.

listDeployedPackages: Der Aufruf erfolgt ohne Parameter und es wird die Liste der Namen von deployten Packages zurückgegeben.

listProcesses: Erwartet einen Package-Namen als Parameter und gibt eine Liste der Prozess-IDs zurück.

getProcessPackage: Erwartet einen Prozess-Namen als Parameter und liefert den Namen des dazugehörigen Packages zurück.

Process-Management

Der *ProcessManagementService* stellt die folgenden Methoden zur Verfügung:

listProcesses: Erwartet als Parameter einen *Filter*-String und einen *orderKeys*-String. Gefiltert werden kann nach folgenden Eigenschaften:

- name: Der Name des Prozessmodells.
- namespace: Der Namespace, der dem Prozessmodell zugeordnet ist.
- status: Der Status des Prozessmodells, entweder „activated“ oder „retired“.
- deployed: Relativer Zeitpunkt des Deployments im ISO-8601-Datums-Format oder ISO-8601-Datums/Zeit-Format.

Die einzelnen Filter können Leerzeichen-separiert angegeben werden. Laut Dokumentation der Apache ODE sollte bei den Namen nicht auf Groß- und Kleinschreibung geachtet werden, allerdings verhält sich zumindest die eingesetzte Version (Entwicklerversion vom 19. November 2008, Revision 718763) anders, d.h. dass mittels `name=multi*` auch wirklich nur der Prozess „multitenancy“ und nicht „MultiTenancy“ gefunden würde. In diesem Beispiel sieht man außerdem, dass beim Namensfilter auf gleichen Anfang geprüft werden kann. Diese Funktionalität ist beim Namespace-Filter equivalent, die anderen Filter unterstützen lediglich exakte Übereinstimmung. Wird kein expliziter Filter angegeben, kommt der Standardfilter `name=* namespace=*` zum Einsatz.

Geordnet werden können die Prozessmodelle nach den Schlüsseln „name“, „namespace“, „version“, „status“ und „deployed“. Die Richtung wird durch ein vorangestelltes „+“ oder „-“ angegeben.

listAllProcesses: Der Aufruf erfolgt ohne Parameter und es wird eine Liste von Informationen aller Prozessmodelle zurückgegeben.

listProcessesCustom: Erwartet wie `listProcesses` als Parameter einen *Filter*-String und einen *orderKeys*-String und zusätzlich kann auch ein Parameter *customizer* angegeben werden. Dessen Werte können sein:

- `SUMMARY`: Gibt zusätzlich Statistiken über dessen Instanzen zurück (Anzahl an Instanzen mit Status `ACTIVE`, `COMPLETED`, `ERROR`, `FAILED`, `SUSPENDED`, `TERMINATED`).
- `PROPERTIES`: Gibt eine Liste der optional gesetzten Properties zurück.
- `ENDPOINTS`: Gibt eine Liste der Endpoints zurück.

Jeder Customizer gibt zusätzlich eine Liste der zum Prozessmodell gehörenden Dokumente (Name, Typ und Pfad) zurück. Customizer können auch miteinander kombiniert werden, indem sie Komma-separiert angegeben werden. Die Namen müssen zwangsläufig in Großbuchstaben geschrieben und ohne führende oder folgende Leerzeichen angegeben werden.

getProcessInfo: Erwartet als Parameter eine Prozess-ID und gibt Informationen über diesen Prozess zurück.

getProcessInfoCustom: Erwartet als Parameter zusätzlich zur Prozess-ID einen Customizer wie bei `listProcessesCustom`, allerdings muss hier mindestens ein Wert angegeben werden.

getExtensibilityElements: Erwartet eine Prozess-ID und beliebig viele Activity-IDs als Parameter und gibt entsprechende Zeilenangaben im Prozessmodell zurück.

setProcessProperty: Erwartet als Parameter eine Prozess-ID, einen Property-Namen und einen Property-Wert als SimpleType und gibt eine Prozess-Information zurück.

setProcessPropertyNode: Erwartet als Parameter eine Prozess-ID, einen Property-Namen und einen Property-Wert als DOM-Knoten und gibt eine Prozess-Information zurück.

activate: Erwartet als Parameter eine Prozess-ID, aktiviert den entsprechenden Prozess und gibt eine Prozess-Information zurück.

setRetired: Erwartet als Parameter eine Prozess-ID und einen Bool'schen Wert, der angibt, ob der Prozess deaktiviert („retired“) oder aktiviert werden soll und gibt eine Prozess-Information zurück.

Bei Prozess-IDs und bei Property-Namen muss ein Namespace wie in Listing 2.7 angegeben werden.

```
<pid xmlns:ns="http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess">
  ns:DynamicInvokeProcess-1
</pid>
```

Listing 2.7: Beispiel-Angabe eines Namespaces

Instance-Management

Der *InstanceManagementService* stellt die folgenden Methoden zur Verfügung:

listInstances: Erwartet als Parameter einen *Filter*-String, einen *orderKeys*-String und eine Limitierungsangabe und gibt eine Liste von Instanz-Informationen zurück. Gefiltert werden kann nach folgenden Eigenschaften:

- name: Der Name der Prozessinstanz.
- namespace: Der Namespace, der dem Prozessmodell zugeordnet ist.
- status: Der Status der Prozessinstanz, entweder „activ“, „suspended“, „error“, „completed“, „terminated“ oder „faulted“.
- started: Relativer Startzeitpunkt im ISO-8601-Datums-Format oder ISO-8601-Datums/Zeit-Format.
- last-active: Relativer Zeitpunkt, seitdem die Instanz zuletzt Arbeit verrichtet hat, entweder bis er abgeschlossen wurde oder seitdem er auf eingehende Nachrichten wartet.
- $property: Korrelations-Eigenschaft der Prozessinstanz.

Diese Filter funktionieren analog zu den Filtern im ProcessManagementService (Details siehe dort). Geordnet werden können die Instanzen nach den Schlüsseln „pid“, „name“, „namespace“, „version“, „status“, „started“ und „last-active“. Die Richtung wird ebenfalls durch ein vorangestelltes „+“ oder „-“ angegeben.

queryInstances: Erwartet als Parameter einen Payload-String, der z. B. einen Namens-Filter enthalten kann. Es wird eine Liste von Instanz-Informationen zurückgegeben.

listAllInstances: Der Aufruf erfolgt ohne Parameter und es wird eine Liste von Instanz-Informationen zurückgeliefert.

listAllInstancesWithLimit: Erwartet als Parameter einen Payload-String, der einen nicht-negativen Integer als Limitierungsangabe enthält, und gibt eine Liste von Instanz-Informationen zurück.

getInstanceInfo: Erwartet als Parameter eine numerische Instanz-ID und gibt die entsprechende Instanz-Information zurück.

getScopeInfo: Erwartet als Parameter eine Scope-Instanz-ID und gibt eine Scope-Information zurück.

getScopeInfoWithActivity: Analog zu `getScopeInfo` mit einem zusätzlichen Bool'schen Parameter, der angibt, ob auch Aktivitäts-Informationen des Scopes zurückgegeben werden sollen.

getVariableInfo: Erwartet als Parameter eine numerische Scope-ID und einen Variablen-Namen und gibt die entsprechende Variablen-Information zurück.

listEvents: Erwartet als Parameter einen Instanz-Filter, einen Event-Filter (`TYPE` und `TIMESTAMP`) und eine numerische Angabe zur Beschränkung der Prozessinstanzen, die bei nicht-positivem Wert ignoriert wird. Es wird eine Liste von Event-Informationen zurückgegeben.

getEventTimeline: Erwartet als Parameter einen Instanz-Filter und einen Event-Filter und liefert eine Liste von Datumswerten als String zurück.

suspend: Erwartet als Parameter eine Instanz-ID, suspendiert diese Instanz und gibt die entsprechende Instanz-Information zurück.

resume: Erwartet als Parameter eine Instanz-ID, lässt die zuvor suspendierte Prozess-Instanz weiterlaufen und gibt eine Instanz-Information zurück.

terminate: Erwartet als Parameter eine Instanz-ID, terminiert diese Instanz umgehend, ohne dass Fault- oder Compensation-Behandlung ausgeführt wird, und gibt eine Instanz-Information zurück.

fault: Erwartet als Parameter eine Instanz-ID, beendet diese Prozess-Instanz als nicht erfolgreich, indem im globalen Scope ein Fehler ausgelöst wird, der allerdings durch Fault- und Compensation-Behandlung verarbeitet werden kann, und gibt eine Instanz-Information zurück.

delete: Erwartet einen Instanz-Filter, der angibt, welche beendeten Instanzen gelöscht werden sollen und gibt eine Liste der IDs der gelöschten Instanzen zurück.

recoverActivity: Erwartet als Parameter eine Instanz-ID, eine Aktivitäts-ID und einen Aktions-String („cancel“, „retry“ oder „fault“), versucht die Aktivität wiederherzustellen und gibt eine Instanz-Information zurück.

In [Gör09] wird ein weiterer Service zum Sicherheits-Management eingeführt, dessen detaillierte Beschreibung der Interfaces dort nachgelesen werden kann.

Security-Management

2.4.2.7 Web-Interface

Die Apache ODE besitzt weiterhin ein Web-Interface, welches aus verschiedenen HTML-Seiten besteht, die den Zugriff auf einige Funktionen der Management-API erlauben. Bei der Umsetzung wurde umfangreicher Gebrauch von AJAX (vgl. Kap. 2.2.4) gemacht, welches durch die Verwendung der Yahoo! User Interface Library Version 2 (YUI2)[48] möglich ist.

Die Arbeit mit dem Web-Interface umfasst den gesamten Lebenszyklus eines Geschäftprozesses vom Deployment eines Packages, über das Verwalten von Prozessmodellen (aktivieren, deaktivieren, ...) und deren Instanzen (beenden, unterbrechen, fortsetzen) bis hin zum Undeployment des Packages. Zusätzlich existieren einfache Statistiken über die Anzahl der Prozesse und Instanzen, sowie eine genaue Kategorisierung aller Prozessinstanzen nach ihrem Status (siehe Abb. 2.9).

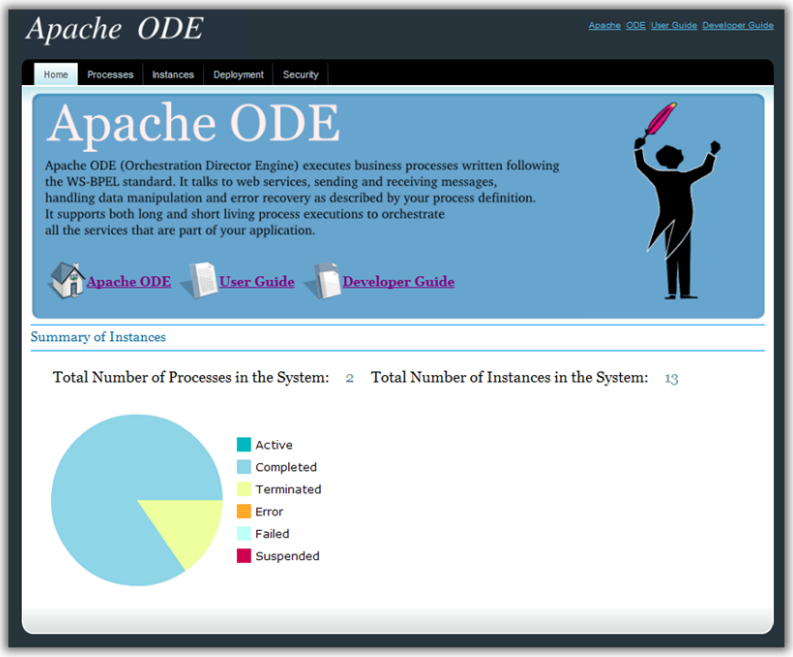


Abbildung 2.9: ODE-Statistik im Web-Interface

In [Gör09] ist ein zusätzlicher Menüpunkt hinsichtlich der Sicherheitseinstellungen für Prozesse zum Web-Interface hinzugekommen.

[48] `http://developer.yahoo.com/yui/2/`

Das Web-Interface ist im Auslieferungszustand der Engine völlig ungesichert, sodass ohne besondere Vorkehrungen jeder, der darauf zugreift, alle Funktionen nutzen kann.

Kapitel 3

Multi-Tenancy

Das übergeordnete Themengebiet dieser Diplomarbeit lautet Multi-Tenancy, weshalb wir diesen Ansatz im folgenden Kapitel differenziert betrachten werden. Die Grundlagen zu Multi-Tenancy haben wir bereits in Kapitel 1 kennengelernt. In Kapitel 3.1 widmen wir uns nun zunächst der Motivation, wann Multi-Tenancy zum Einsatz kommen sollte. In den Abschnitten 3.2 bis 3.4 werden verschiedene Sichtweisen und Ansätze diskutiert, welche abschließend in einer Bewertung gegenübergestellt werden. Abschnitt 3.5 liefert uns ein paar Beispielszenarien zur Veranschaulichung. Kapitel 3.6 befasst sich mit Fragen, die sich Anbieter und Kunden beim Umstieg auf Multi-Tenancy stellen könnten und sollten. Schließlich wird in Abschnitt 3.7 über Multi-Tenancy in Verbindung mit BPEL gesprochen, welches eine Überleitung zum praktischen Teil dieser Arbeit darstellt.

3.1 Motivation für Multi-Tenancy

Wann sollte man sich für Multi-Tenancy entscheiden? Was sind die Vorteile und unter welchen Umständen ist von Multi-Tenancy vielleicht sogar abzuraten? Nicht für jedes Szenario ist ein derartiger Ansatz nämlich sinnvoll, aber grundlegend kann Multi-Tenancy sowohl für Anbieter als auch für Kunden von Interesse sein.

Als allgemeine und wichtigste Motivation für Multi-Tenancy lässt sich natürlich Kostenreduktion festhalten. In diesem Abschnitt werden die Anreize jedoch differenzierter beleuchtet. Wie wir dann in Kap. 3.2 sehen werden, gibt es unterschiedliche Ansätze und Grade von Multi-Tenancy mit jeweils spezifischen Vor- und Nachteilen für Anbieter und Kunden. Die nachfolgenden Punkte sind zunächst aber grundlegende Anreize, sich überhaupt für Multi-Tenancy zu entscheiden und dieses zu implementieren. Auf Anbieterseite sind das:

- *Bessere Auslastung der Ressourcen:*
  Dedizierte Systeme lasten die zur Verfügung stehenden Ressourcen oftmals

nur zu einem geringen Teil aus. Als Beispiel kann man sich eine typische Webpräsenz mit Webserver, dynamisch generierten HTML-Seiten und Anbindung an eine Datenbank vorstellen. Der Apache Webserver, der Skriptsprachen-interpreter PHP und ein MySQL-Server sind weit verbreitete Lösungen für dieses Szenario. Hierbei halten sich der Speicherverbrauch und die CPU-Auslastung selbst für eine Webpräsenz mittlerer Größe in Grenzen. Da der Apache Webserver und der MySQL-Server multi-tenant-fähig sind, lassen sich auf diese Weise ohne Probleme so viele Webpräsenzen auf einem System betreiben, bis eine optimale Hardware-Auslastung erreicht ist. Dieses Prinzip lässt sich auf jedes Multi-Tenancy-Szenario übertragen.

- *Benutzung einer einzelnen Plattform:*
  Die einzelnen Tenants sind nicht für die Infrastruktur verantwortlich, sodass es keine unterschiedlichen Plattformen gibt, die es zu unterstützen gilt. Vielmehr wird das System nur auf einer einzelnen Plattform eingerichtet und alle Tenants greifen gleichermaßen darauf zu. Weil nicht mehr auf unterschiedlichen Plattformen getestet werden muss, verringert sich der Test-Aufwand und die Testphasen fallen kürzer aus. Dieses kann sich auch positiv auf die Dauer der Entwicklungszyklen und ggf. auch auf die sog. „time to market“ auswirken, was einen Wettbewerbsvorteil schaffen kann.

Kunden sind weitaus weniger an der konkreten technischen Umsetzung interessiert als vielmehr an der letztendlichen Funktionalität, der QoS und einem möglichst niedrigen Preis der Nutzung des Systems. Vor allem im Hinblick auf die Kosten kann Multi-Tenancy aber auch auf Seite des Kunden von Interesse sein.

Zudem ist auf Kundenseite auch der folgende Punkt ein sehr positiver Aspekt, den Multi-Tenancy mit sich bringt:

- *Vielfalt des Angebots:*
  In [Mic06a] wird davon gesprochen, dass SaaS ein Mittel ist, um Nischensoftware – den sog. „Long Tail“ – anbieten zu können (vgl. Kap. 2.2.3). Weiterhin heißt es dort, dass in der oberen Hälfte der Reifeskala von SaaS ein Multi-Tenant-Ansatz unverzichtbar ist. Somit kann ein Kunde durch die unterschiedlichen Delivery Models in Verbindung mit Multi-Tenancy mitunter auf eine größere Produktpalette angebotener Software zugreifen.

Darüber hinaus gibt es auch gemeinsame Vorteile für Anbieter und Kunden, zu welchen folgende Punkte zu zählen sind:

- *Zentralisierung:*
  Ein Bestreben von Multi-Tenancy ist zu generalisieren und zu abstrahieren, um letztendlich so weit wie möglich zusammenfassen und zentralisieren zu können. In [Cle07] werden verschiedene Gründe genannt, warum Zentralisierung für den Anbieter sinnvoll bzw. notwendig sein kann.

Ein Aspekt betrifft die Höhe des Budgets. Es kann durchaus vorkommen, dass ein bestimmtes Budget gekürzt wird, ohne dass Abstriche an der Qualität oder am Umfang der Services gemacht werden sollen. Dies bedeutet letztendlich, dass mehr Arbeit mit weniger Ressourcen erledigt werden muss. Weiterhin könnte der Platz im Datenzentrum ausgehen. Ein Umzug bzw. eine Auslagerung von Teilen der Infrastruktur könnte jedoch deshalb nicht in Frage kommen, weil z. B. zusätzliche Mittel fehlen oder eine Auslagerung zu Performance-Einbrüchen innerhalb der Infrastruktur führen würde. Zusätzlich können schwierig zu kalkulierende Größen wie die Kosten für Strom und Kühlung steigen, sodass man auch versuchen muss, diesbezügliche Kosten im Rahmen zu halten. Eine harte Obergrenze wird auch durch die Stromversorgung des Datenzentrums definiert. Diese kann ab einem gewissen Umfang der Infrastruktur ihre Belastungsspitze erreichen. Neben den bislang genannten Argumenten sind auch steigende Kosten der Systemadministration nicht zu vernachlässigen. In all diesen Szenarien kann Multi-Tenancy Abhilfe schaffen oder zumindest Teil eines Lösungsansatzes sein.

Weiterhin profitieren beide Seiten bei der Zentralisierung durch Multi-Tenancy von ggf. wegfallenden Lizenzkosten für Betriebssysteme, Middleware und Anwendungen und der ermöglichten zentralen Installation und Wartung. Ein Release erfolgt somit nicht mehr nach dem Schema, dass ein Installationspaket zusammengestellt wird und sich ggf. eine Fachkraft darum kümmert, dass das System lokal beim Kunden eingerichtet wird, sondern der Vorgang betrifft lediglich ein oder mehrere zentrale Server in den Rechenzentren des Anbieters.

- *Gemeinsame Code-Basis für alle Tenants:*
  Für den Anbieter ist eine gemeinsame Code-Basis vorteilhaft, weil es weniger Aufwand in Entwicklung und Verwaltung bedeutet. Je nachdem wie groß die Anpassungen in einem Entwicklungszweig sind, müssen ggf. für jeden Zweig Aufgaben im Bereich Projektmanagement, Kundenkontakt mit Anforderungsanalyse, Qualitätssicherung etc. vergeben werden. Dieser Mehraufwand für zusätzliche Zweige fällt bei gemeinsamer Code-Basis komplett weg bzw. wird z. B. im Bereich des Kundenkontakts anders ausfallen und auf jeden Fall stark reduziert.

  Die Kunden profitieren davon, dass durch die Konzentration auf eine einzige Produktlinie die Entwicklungszyklen kürzer ausfallen können und dass neu entwickelte Funktionen bei jedem Release auch allen Kunden bereitgestellt werden können.

3.2 Grade von Multi-Tenancy

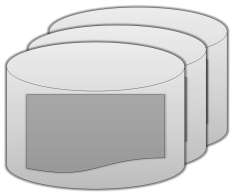
Multi-Tenancy ist für eine eindeutige Definition schwer zu greifen. Man könnte beispielsweise einen sehr idealistischen Ansatz wählen und Multi-Tenancy stark eingrenzen, indem man nur solche Systeme als multi-tenant-fähig bezeichnet, deren Komponenten nur ein einziges Mal instanziiert werden. Man könnte aber auch den pragmatischen Ansatz vertreten und sagen, dass bestimmte Kriterien (z. B. ein gemeinsames Datenbankschema für alle Tenants, weil dadurch der Administrationsaufwand sinkt) grundlegend das Ziel sein sollten, aber in manchen Situationen abgeschwächt werden müssten, um auf spezielle Anforderungen der Systemumgebung oder des Kunden eingehen zu können. Dieses würde das System an sich aber trotzdem multi-tenant-fähig sein lassen. Allgemein sollte man Abstand davon nehmen, nach „gut“ und „schlecht“ zu kategorisieren. Die Anforderungen der Kunden können nämlich so unterschiedlich sein, dass es gar keine ideale Lösung für alle Szenarien zugleich geben kann. Deshalb wird in diesem Kapitel von (Reife-)Graden gesprochen, die sich anhand von bestimmten Eigenschaften ergeben.

Zu den Punkten, die den Grad an Multi-Tenancy festlegen, zählen der Grad der gemeinsamen Nutzung des Datenbankschemas und der Grad der Konfigurierbarkeit des Datenmodells, welche in den folgenden Abschnitten erläutert werden. Im weiteren Verlauf dieses Kapitels werden wir uns dann den verschiedenen Ansätzen von Multi-Tenancy widmen und die jeweiligen Vor- und Nachteile herausarbeiten.

Grad der gemeinsamen Nutzung des Datenbankschemas

Grundlegend gibt es drei Möglichkeiten, den Tenants die Daten zur Verfügung zu stellen. Hier sind diese aufsteigend nach dem Grad an Multi-Tenancy sortiert:

DBs: n

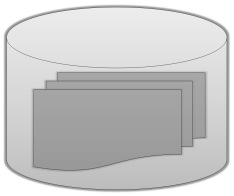


Die erste Möglichkeit besteht darin, jedem Tenant eine eigene Datenbank zur Verfügung zu stellen. Dadurch wird die höchstmögliche Isolation zwischen den Tenants erreicht und auch das einspielen von Backups ist sehr einfach, weil andere Tenants beim Importieren der Daten nicht beeinflusst werden. In [JA07] wird jedoch mittels eines Experiments festgestellt, wie sich die Auslastung von Arbeitsspeicher und Festplattenspeicher bei steigender Anzahl an Schemainstanzen verhält. Aus Tabelle 3.1 lässt sich leicht ableiten, dass eine Datenbank mit jeweils einem Schema für 10.000 Tenants eine deutlich höhere Auslastung von Arbeitsspeicher und Festplatte mit sich bringt als eine Datenbank, in welcher 10.000 Tenants je eine Schemainstanz zugewiesen bekommen. Zwar schwankt die Auslastung des Speichers zwischen den einzelnen Datenbankprodukten mitunter sehr stark, trotzdem können letztendlich aus Mangel an Ressourcen bei allen verglichenen Produkten nur relativ wenige Tenants auf einem einzelnen Server bedient

werden. Aus diesem Grund lässt sich dieser Ansatz nur dann sinnvoll anwenden, wenn Isolation höchste Priorität hat.

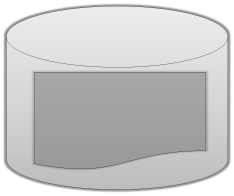
Alle Angaben in [MB]	Arbeitsspeicher		Festplatte	
	eine Schemainstanz	10.000 Schemainstanzen	eine Schemainstanz	10.000 Schemainstanzen
PostgreSQL	55	79	4	4488
MaxDB	80	80	3	1168
Commercial1	171	616	200	414210
Commercial2	74	2061	3	693
Commercial3	273	359	1	13610

Tabelle 3.1: Speicherverbrauch von Schemainstanzen nach [JA07]



DBs: 1
Schemata: n

Wie bereits angesprochen, ist die Ressourcenauslastung wesentlich geringer, wenn sich alle Tenants eine einzelne Datenbank teilen. Jedem Tenant kann dabei ein eigenes Datenbankschema zugewiesen werden, d.h. er bekommt eigene Tabellen, die vom Aufbau identisch mit allen anderen Tenants sind, jedoch einen anderen Namen tragen, beispielsweise nach dem Schema `TableName_<TenantID>`. Bei diesem Ansatz ist zumindest die logische Isolation der Daten gewährleistet. Das Erstellen von Backups ist leichter, weil nur eine Datenbank genutzt wird, allerdings ist das Einspielen der Daten aus dem Backup erheblich schwieriger, weil man beim Einspielen des kompletten Backups auch die möglicherweise nicht beeinträchtigten Tabellen unterschiedlicher Tenants zurücksetzen würde. [Mic06a] schlägt vor, das Backup auf einem temporären System einzuspielen und nur die befallenen Tabellen auf das eigentliche System zu übertragen.



DBs: 1
Schemata: 1

Bei der dritten Variante teilen sich alle Tenants die selben Tabellen. Es existiert dabei eine zusätzliche Spalte mit einem Tenant-Identifier (vgl. Tab. 3.3), anhand dessen eine Partitionierung der Daten stattfindet. Eine Isolation der Daten findet auf diesem Weg praktisch nicht statt. Wenn das Database Management System (DBMS) keine native Unterstützung zur Selektion der tenant-spezifischen Daten anhand der Tenant-ID mit sich bringt, muss dies vollständig im Anwendungsprogramm erfolgen. So könnte es allerdings leicht geschehen, dass durch einen Programmierfehler Zugriff auf Daten eines anderen Tenants erfolgen kann. Von daher kann es z. B. hilfreich sein, einen View zu erzeugen, der anhand der ID des eingeloggten Datenbankbenutzers eine Verknüpfung zur Tenant-ID herstellt und die jeweilige Tabelle danach filtert. Der einfachste Fall ist der, wenn die ID des Datenbankbenutzers direkt der Tenant-ID entspricht (siehe Listing 3.1).

Ein nicht zu vernachlässigender negativer Aspekt bei einem gemeinsam genutzten Datenbankschema ist das Einspielen von Backups, welches hierbei auf Zeilenebene geschehen muss und deshalb sehr aufwändig und fehleranfällig ist.

```
CREATE VIEW TenantEmployees AS
SELECT * FROM Employees
WHERE Tenant-ID = SUSER_SID()
```

Listing 3.1: Zugriff auf tenant-spezifische Daten mithilfe eines Views

Letztendlich spart man durch ein einziges Datenbankschema zu einem gewissen Grad Speicherressourcen, jedoch muss man bedenken, dass die Größe der Tabellen möglicherweise limitiert sein kann, zum einen durch das DBMS selbst und zum anderen durch die höchstmögliche Dateigröße auf der verwendeten Plattform. Zudem beeinträchtigen sehr große Tabellen auch die Geschwindigkeit, mit der Anfragen vom DBMS bearbeitet werden können.

Grad der Konfigurierbarkeit des Datenmodells

Ein grundlegendes Problem ist, dass in den meisten Fällen jeder Tenant ein eigenes Datenmodell in seiner Organisation erarbeitet hat und verwenden will. Ohne tenant-spezifische Anpassungen am Quelltext könnte der Anbieter einem Tenant aber nicht sein eigenes Datenmodell verwenden lassen, weshalb dies unter Multi-Tenancy-Aspekten für den Anbieter nicht in Frage kommt. Das schränkt den Kunden ggf. ein und es kann mitunter auch schwierig bis unmöglich sein, das eigene Datenmodell auf das generische Datenmodell des Anbieters abzubilden.

Ein häufig angewandter Kompromiss ist, dass die angebotenen Anwendungen ein Standard-Datenmodell verwenden, welches sich von den einzelnen Tenants zu einem gewissen Grad anpassen lässt.

Die erste Variante hierfür ist es, eine feste Anzahl zusätzlicher Spalten einzuführen. Die Schwierigkeit hierbei besteht darin, eine sinnvolle Anzahl an zusätzlichen Spalten zu bestimmen. Es könnte auch ohne weiteres der Fall eintreten, dass `n` frei belegbare Spalten definiert sind, ein Tenant allerdings (vielleicht erst nach einer gewissen Zeit) `n + 1` zusätzliche Spalten benötigt. Dieser Ansatz ist vergleichsweise leicht zu implementieren, allerdings bemerkt man recht schnell dessen Grenzen, denn weiterhin bleibt die Frage offen, von welchem Typ die Spalten sein sollen: jeweils ein paar Spalten vom Typ `String`, vom Typ `Integer` etc.? Um zumindest dieses Problem in den Griff zu bekommen, legt man sich auf den Datentyp `String` für alle zusätzlichen Spalten fest und führt zusätzlich eine Metadaten-Tabelle ein, in welcher die Spalten-Bezeichnung und der Datentyp gespeichert werden (siehe Tab. 3.2). Hierdurch lässt sich pro Tenant festlegen, welchen Datentyp die Werte in den zusätzlichen Spalten haben sollen. Eine automatische Validierung der Werte durch das Datenbanksystem ist hierbei natürlich nicht möglich. Die Validierung muss in der Anwendung selbst erfolgen, was zu einem höheren Aufwand führt und fehleranfälliger ist.

ID	NAME	C_1	C_2	C_3
31	Hans	Stuttgart	5	Informatik
32	Kathrin	10.09.1981	verheiratet	2

TABLE-ID	C_1-DATATYPE	C_1-LABEL
192	String	Wohnort
221	Integer	Prüfungsnote

TABLE-ID	C_3-DATATYPE	C_3-LABEL
192	Integer	PLZ
221	String	Telefonnummer

Tabelle 3.2: Erweiterung durch feste Anzahl zusätzlicher Spalten (vgl. [Mic06b])

Die zweite Variante mittels Schlüssel-/Wert-Paaren (siehe Tabelle 3.3) ist am flexibelsten und ist für ein gemeinsames Schema für alle Tenants am besten geeignet. Allerdings wird hierbei die Performanz durch viele `join`-Operationen stark beeinträchtigt.

TENANT-ID	NAME	GEBURTSDATUM	RECORD-ID
31	Stefan	26.11.1986	11
42	Martin	17.10.1964	26

RECORD-ID	EXT-ID	VALUE
11	13	Mannheim
11	14	1
26	54	Abteilungsleiter

TENANT-ID	EXT-ID	EXT-LABEL	EXT-DATATYPE
31	13	Wohnort	String
31	14	Semester	int
42	54	Position	String

Tabelle 3.3: Erweiterung des Datenmodells mittels Schlüssel-/Wert-Paaren (vgl. [Mic06b])

3.2.1 Native Multi-Tenancy

Wenn man an Multi-Tenancy denkt, hat man zuerst native Multi-Tenancy im Kopf. Der Grundgedanke ist: Jede Instanz einer Anwendung hat einen gewissen Overhead, warum sollten also verschiedene Instanzen gestartet werden, wenn alle Kunden auch auf dieselbe Instanz zugreifen können?

Native Multi-Tenancy bedeutet also, dass in allen Schichten einer Schichtenarchitektur nur eine Instanz einer Komponente läuft. Im Fall der Daten-Schicht beispielsweise heißt dies, dass eine einzige Instanz eines Datenbank-Servers läuft und dass nur ein Schema für alle Tenants verwendet wird. Dieses kann natürlich etwas aufgeweicht werden, wenn man mehrere Datenbank-Server beispielsweise mittels synchroner Replikation betreibt oder eine Cluster-Lösung einsetzt, allerdings kann man dieses wiederum als eine einzelne, zusammengesetzte Komponente betrachten, die für alle Tenants gleich ist.

In [GSH+07] wird schwerpunktmäßig auf native Multi-Tenancy eingegangen. Es wird ein Framework vorgestellt, welches die Entwicklung einer Multi-Tenancy-

Anwendung erleichtert. Der Fokus wird dabei auf native Multi-Tenancy gelegt. Es werden Services eingeführt, die vor allem Isolation und Anpassung für Multi-Tenancy ermöglichen („Enablement“). Es wird auch dafür plädiert, zwei Gruppen von Entwicklern getrennt von einander einzusetzen. Die Teile der Anwendung, die für Multi-Tenancy verantwortlich sind, sollten von wenigen Spezialisten entwickelt werden, sodass die meisten Anwendungsentwickler sich auf die Business-Logik konzentrieren können. Eine besondere Schwierigkeit dabei sei es, wie man die Services, die Multi-Tenancy ermöglichen, mit multi-tenant-fähigen Ressourcen verknüpfen könne. Hierfür sei eindeutig Expertenwissen von Nöten. Außerdem wird argumentiert, dass ein besonderes Augenmerk auf Effizienz und Skalierbarkeit liegen müsse, weil sich Multi-Tenancy-Services vor allem an SMB-Kunden richten würden.

Vorteile:

- *Lizenzen:*
  Bei native Multi-Tenancy ist ein entscheidender Vorteil, dass man die Lizenzkosten für Betriebssysteme, DBMS und andere Software-Komponenten auf das absolute Minimum reduzieren kann.
- *Personal:*
  Durch die Zentralisierung ist zumindest für den Betrieb und die Wartung der Hardware weniger Personal notwendig.
- *Backups:*
  Native Multi-Tenancy in der Datenschicht ermöglicht es, auf einfache Art und Weise Backups des gesamten Systems anzulegen.
- *Skalierbarkeit:*
  Das grundlegende Ziel von Multi-Tenancy ist es, eine optimale Ressourcenauslastung zu erreichen. Dieses Bewusstsein ist bei native Multi-Tenancy während der Planung und Entwicklung in besonderem Maße vorhanden und lässt den Anbieter nach den besten bekannten Technologien zur Förderung dieses Ziels suchen und diese verwenden. Dies führt dazu, dass ein Multi-Tenancy-System im allgemeinen recht gut skaliert und mit gängiger Hardware sehr viele Tenants gleichzeitig bedient werden können.

Nachteile:

- *Mangelnde Flexibilität:*
  Indem man festlegt, dass alle Tenants von einer einzigen Instanz einer Komponente bedient werden, nimmt man dem System die Flexibilität. Es ist somit

schwieriger, nach Bedarf einzelnen Kunden spezielle SLA anbieten und garantieren zu können oder auf vorübergehend abweichende Anforderungen einzugehen.

- *Minimale Isolation:*
  Je mehr Kunden einem System zugeordnet sind, desto mehr fällt die geringe Isolation ins Gewicht. Dieses kann z. B. die Performanz oder gar die Verfügbarkeit des Systems beeinträchtigen.
- *Erhöhte Anforderungen an Verfügbarkeit:*
  Zentralisierung birgt Risiken wie Single Point of Failure und einzelne Komponenten können zum Flaschenhals werden, weshalb in diesen Bereichen besondere Vorkehrungen zu treffen sind. Dass es sich hierbei um ein Problem aus dem Alltag handelt, wird in [AFG+09] dargelegt. Dort sind einige Beispiele von Ausfällen prominenter Service-Anbieter aufgelistet, die durch eher kleine Probleme aufgetreten sind, sich aber teilweise über mehrere Stunden hingezogen haben (siehe Tab. 3.4).

Dienst und Ausfälle	Dauer	Datum
Ausfall Amazon S3: Nicht-Verfügbarkeit durch Überlastung des Authentifizierungs-Services	2 Stunden	15.02.08
Ausfall Amazon S3: Single Bit Fehler, der das Gossip-Protokoll lahmgelegt hat	6-8 Stunden	20.07.08
Teilausfall Google App Engine: ausgelöst durch einen Programmierfehler	5 Stunden	17.06.08
Nichtverfügbarkeit der GMail-Webseite: hervorgerufen durch Ausfall im Kontakte-System	1,5 Stunden	11.08.08

Tabelle 3.4: Service-Ausfälle prominenter Internet-Dienstleister [AFG+09]

3.2.2 Gemeinsamkeiten aller Ansätze

Welche Eigenschaften verbinden alle Multi-Tenancy-Ansätze dieses Kapitels miteinander? Ab welchem Punkt kann man selbst nach dem pragmatischen Ansatz nicht mehr von Multi-Tenancy sprechen? Im vorigen Abschnitt haben wir native Multi-Tenancy betrachtet, welches das eine Ende des Multi-Tenancy-Spektrums markiert. Wieviel Abweichung davon ist aber möglich, ohne die Multi-Tenancy-Eigenschaft zu verlieren? In diesem Abschnitt soll es darum gehen, welche Eigenschaften als fundamentale Annahmen für Multi-Tenancy angesehen werden können.

Wenn statt einer einzelnen Instanz mehrere Instanzen eingesetzt werden, um die verschiedenen Tenants zu bedienen, sollte eine grundlegende Übereinstimmung sein, dass alle Instanzen auf eine gemeinsame Code-Basis zurückgreifen. Hierbei ist

die Konfigurierbarkeit des Systems von allerhöchster Bedeutung, damit die Tenants das System an ihre eigenen Bedürfnisse anpassen können. Dieses verbirgt vor den Endbenutzern in gewisser Weise auch die Tatsache, dass sie auf ein grundlegend generisches System mit unterschiedlichen Benutzern zugreifen. Aber selbst wenn es gar nicht darum geht, alle Konfigurationsmöglichkeiten an den Tenant weiterzugeben, empfiehlt es sich, das System so konfigurierbar wie möglich zu halten. [Ham07] spricht davon, dass selbst Werte, die sich im laufenden Betrieb vermutlich nie ändern werden, sich ohne Codeanpassungen verändern lassen sollten, solange es leicht umzusetzen ist.

Demnach kann man in einem Multi-Tenancy-Szenario drei Arten von Anpassungsmöglichkeiten und Konfigurierbarkeit unterscheiden:

(a) *änderbar ausschließlich durch Anbieter*:
Hierbei handelt es sich um Eigenschaften, die das gesamte System betreffen.

(b) *einmalig, durch Tenant-Administrator*:
Diese Kategorie betrifft allgemeine Eigenschaften der Benutzungsoberfläche (z. B. Logo, Farbschema, Layout etc.), Anpassungen am Datenbankschema (z. B. festlegen des Datentyps von benutzerdefinierbaren Spalten) und weitere Metadaten wie beispielsweise Benutzergruppen, Zugriffsrechte und Endpunktdefinitionen.

(c) *nach Bedarf, durch Tenant-Endbenutzer*:
Dieses umfasst z. B. die Möglichkeit, Verfeinerungen am Layout und weitere benutzerabhängige Einstellungen für eine spezifische Anwendung vornehmen zu können.

Zunächst müssen die Anpassungsmöglichkeiten von den Anwendungsentwicklern zur Verfügung gestellt werden. Dies kann sowohl manuell als auch automatisiert geschehen (vgl. [GSH+07]). Hierbei kann es durchaus zu einer großen Anzahl von Anpassungsmöglichkeiten kommen. Dabei ist es wichtig, das Interface intuitiv und übersichtlich zu gestalten und den Tenants soviel Hilfestellung wie möglich zu bieten (z. B. geführte Wizards, ausgiebige Online-Hilfe etc.), damit der Benutzer alle Möglichkeiten erkennen kann und auch in der Lage ist, diese zu nutzen. Außerdem muss auch eine Eingabevalidierung stattfinden, sodass fehlerhafte Einstellungen und Eingaben so früh wie möglich erkannt und vermieden werden können und auch einstellungsübergreifenden Konflikten in der Konfiguration aus dem Weg gegangen werden kann.

Ein ganz wichtiger Punkt ist zudem die Möglichkeit zur Anpassung im laufenden Betrieb, sodass die Einstellungen von Tenant A nicht die Nutzung des Systems durch Tenant B beeinträchtigt. Wünschenswert wäre in diesem Zusammenhang natürlich auch, dass nach Möglichkeit sogar Software-Updates ohne Unterbrechung des laufenden Betriebs durchgeführt werden könnten. Weil dies aber schon rein

technisch nicht für jede Art von Änderungen und Updates des Systems möglich ist, kann dies auch nicht als grundlegende Eigenschaft für alle Ansätze von Multi-Tenancy gefordert werden.

Eine weitere Gemeinsamkeit ist, dass eine gemeinsame Infrastruktur besteht, auf welche zugegriffen werden kann. Obwohl in den meisten Fällen bereits die Plattform durch die gemeinsame Code-Basis vorgegeben ist, könnte es sein, dass alle Komponenten des Multi-Tenancy-Systems plattformübergreifend lauffähig sind (z. B. Servlets und Services in Java geschrieben, die mittels Apache Tomcat[1] und Apache Axis2[2] zur Verfügung gestellt werden und auf eine IBM DB2-Datenbank[3] zugreifen), sodass einzelne Komponenten auch auf mehreren Plattformen parallel instanziiert werden könnten.

Vorteile:

- Benutzer können weltweit von jedem Rechner, der die Minimalanforderungen erfüllt (z. B. JavaScript-fähiger Internet-Browser ist installiert, vgl. Kap. 2.2), ohne aufwändige Software-Installation auf das System zugreifen.

Nachteile:

- Abhängigkeit nicht nur von der Verfügbarkeit der Server, auf denen das System läuft, sondern auch von den Übertragungsmedien (z. B. Intranet und Internet).

3.2.3 Multi-Instance Multi-Tenancy

Bei diesem Ansatz entscheidet man sich dafür, einzelne Komponenten des Systems nach Bedarf mehrfach zu instanziieren, um Engpässen in Speicher, Rechenkapazität, Bandbreite oder ähnlichen Eigenschaften vorzubeugen oder diese zu beheben.

Hierbei müssen zwar mehrere Instanzen betreut werden, dennoch handelt es sich jeweils um die gleiche Umgebung: die Instanzen sind jeweils identisch, es wird das gleiche Datenbankschema verwendet usw., mit dem Unterschied, dass mehrere Server zum Einsatz kommen. Ein besonderer Vorteil dieser Variante ist, dass Kunden mitunter zwischen den einzelnen Instanzen „verschoben“ werden können, wenn spezielle SLA ausgemacht werden oder bestehende Ressourcen nicht

[1] http://tomcat.apache.org
[2] http://ws.apache.org/axis2/
[3] http://www.ibm.com/db2/

mehr ausreichen und zusätzliche Server eingesetzt werden müssen. Ein konkretes Szenario dafür wäre ein Anbieter, der verschiedene Ausbaustufen je nach Größe und Zahlungsfähigkeit des Kunden anbietet. Jeder Tenant könnte das System zunächst in einer kostenlosen Test-Umgebung ohne Service Level Agreements (SLAs) ausprobieren. Sobald er sich für den Anbieter entscheidet, zieht er in eine Produktiv-Umgebung um und bezahlt für den Service. Je nach Bedarf und angebotenen Ausbaustufen kann der Tenant weiterhin in andere Umgebungen mit höherwertigen SLAs umziehen, sodass jeweils das beste Kosten-/Nutzen-Verhältnis besteht.

Nachteilig an dieser Variante von Multi-Tenancy ist vor allem der höhere Personalaufwand und die wegfallende Möglichkeit zentraler Installation und Wartung.

3.2.4 Heterogene Lösungen

Neben den reinen Multi-Tenancy-Lösungen, entweder native oder Multi-Instance Multi-Tenancy, gibt es auch Lösungen, die zwar Aspekte von Multi-Tenancy umsetzen, aber dennoch in der Gesamtheit nicht als reine Multi-Tenancy-Systeme angesehen werden können, weil auch andere Paradigmen eine Rolle spielen, die die Multi-Tenancy-Eigenschaften weiter aufweichen oder aufheben.

- *Multi-Tenancy on-premise*:
  Stellen wir uns einmal folgendes Szenario vor: Eine große Firma oder Organisation (der Tenant) hat viele Endbenutzer, die auf ein Multi-Tenancy-System zugreifen würden. Es könnte sein, dass die Geschäftsleitung von einem Anbieter einer Multi-Tenancy-Lösung überzeugt ist, allerdings ein paar berechtigte Bedenken hegt, da es bestimmte Anforderungen gibt, die der Anbieter noch nicht oder nicht vollständig nach Wunsch des Tenants umsetzen bzw. garantieren kann. Beispielsweise könnte es um die Frage der Sicherheit des Systems gehen, weil mit hochsensiblen Daten umgegangen werden würde. Oder es könnte die nötige hohe Verfügbarkeit betreffen, die über das Internet nicht ständig oder zumindest nicht mit konstant hoher Bandbreite garantiert werden kann.

  Da der Tenant allerdings bereit ist, für alle zusätzliche Anstrengungen und Leistungen seitens des Anbieters großzügig zu bezahlen, könnte man sich auf einen Kompromiss einigen.

  Der Anbieter könnte beispielsweise nach dem Ansatz von Intalio[4] „Multi-Tenancy on-premise“ (Multi-Tenancy im Hause des Tenants) die notwendige Hardware kaufen, das eigene Multi-Tenancy-System darauf einrichten, um im Anschluss alles im Datenzentrum des Tenants unterzubringen. Es wäre

[4] http://intalio.com

nach wie vor gewährleistet, dass es sich um die gleiche, auch sonst verwendete Plattform handelt und auch die Code-Basis identisch wäre. Lediglich in Betrieb und Wartung entstehen zusätzliche Aufwände, für die der Tenant aber bereit ist, alle Kosten zu übernehmen, weil das System nun vollständig seinen Anforderungen genügt.

Letztendlich könnte man dieses Vorgehen übergeordnet als Single-Tenancy beschreiben, während das eingesetzte System, welches mehrfach instanziiert wird, eigentlich die Multi-Tenancy-Fähigkeit besitzt.

3.3 Nicht-funktionale Anforderungen

In diesem Abschnitt werden wir ein wenig genauer auf einige nicht-funktionale Anforderungen eingehen, die sich speziell auf Multi-Tenancy-Systeme beziehen und in der Betrachtung der verschiedenen Reifegrade noch nicht angesprochen wurden.

Flexibilität

Unter diesem Punkt versteht man die Unterstützung von Standards, damit beispielsweise Bibliotheken nach Bedarf gegeneinander ausgetauscht werden können. Momentan existieren im Bereich von Multi-Tenancy keine herstellerübergreifenden Standards. Es existieren zwar Plattformen wie *force.com* oder *Google App Engine* (siehe Kap. 2.2.2), die als Basis für mehrere Anwendungen genutzt werden können, oder Ansätze wie in [GSH+07], wo ein Framework vorgestellt wird, das Native Multi-Tenancy ermöglichen soll, allerdings ist fraglich bis zu welchem Grad überhaupt Standards möglich sind. Dagegen könnte es durchaus zu Standards in Teilbereichen kommen, beispielsweise auf Datenebene, wie mit mehreren Tenants verfahren werden kann.

Datensicherheit

Die Sicherheitsaspekte eines Multi-Tenancy-Systems sind breitgefächert, allerdings werden wir uns hier auf den Kernpunkt konzentrieren, nämlich den Schutz der Daten des Tenants. Hierzu gehören:

- *Informationen, die direkt vom Tenant abgelegt werden*
  Hierzu gehören beispielsweise Mitarbeiter-Akten oder eine Kundenkontakt-Datenbank. In solchen Angelegenheiten ist gesetzlich geregelt, dass ein Datenschutz gewährleistet wird.

- *alle Einstellungen und auch Daten, die für den Betrieb des Systems notwendig sind*
  In diese Kategorie fallen beispielsweise Adressen von Endpunkten für externe Services oder auch Prozessmodelle. Dieses Wissen gehört zu den Geschäftsgeheimnissen eines Unternehmens und sollte deshalb unter besonderen Schutz gestellt werden.

- *alle Arten von im Betrieb errechneten (Zwischen-)Ergebnissen*
  Selbst wenn alle Daten in der Datenbank verschlüsselt abgelegt werden, könnte jemand mit physischem Zugriff auf die Server trotzdem an die Inhalte kommen, sodass bei Bedarf auch hier schützend eingegriffen werden muss.

- *alle anfallenden nicht-inhaltlichen Daten*
  Hierzu zählt alles, was nicht inhaltlich zu den Daten des Tenants gehört, sondern z. B. für Fehleranalyse und Wartung erzeugt wurde, wie alle Arten von Log-Dateien. Wichtig ist dies insofern, als dass unter Verwendung bestimmter Data-Mining-Algorithmen z. B. aus Log-Dateien einer BPEL-Engine komplette Prozessmodelle rekonstruiert werden können [AGL98, AWM04, ADH+03].

- *Übertragungsmedien*
  Multi-Tenancy-Systeme sind in der Regel nach einem Client-/Server-Modell aufgebaut. Deswegen ist es unablässig, nicht nur das System auf dem Server abzusichern, sondern auch die komplette bidirektionale Datenübertragung zwischen Client und Server.

Um all diesen Sicherheitsrisiken zu begegnen, sollte man sich folgende Punkte überlegen:

- *Verschlüsselung*:
  Ganz allgemein kann Verschlüsselung auf allen Ebenen eine Hilfe sein. Bezogen auf das Übertragungsmedium heißt das im Falle des Internets z. B. Übertragung per Hypertext Transfer Protocol Secure (HTTPS). Beim Versenden von SOAP-Nachrichten müsste man sich zusätzlich noch Gedanken über Verschlüsselung des Payloads machen, um nach Bedarf End-To-End-Verschlüsselung zu erreichen.

  Auf anderen Ebenen kann es aber durchaus problematisch sein, sinnvolle Verschlüsselung zu ermöglichen, denn wenn beispielsweise alle Daten in der Datenbank verschlüsselt sind, lassen sich Tabellen nicht mehr ohne weiteres über beliebige Spalten mittels des `JOIN`-Operators miteinander verknüpfen. Diese Berechnungen müssen folglich alle in der Anwendung selbst stattfinden, was zu einem deutlich höheren Rechenaufwand und größeren und fehleranfälligeren Programmen führt. Es ist also immer eine Gratwanderung zwischen der Verschlüsselung dessen, was unbedingt gesichert werden muss auf der einen Seite und Performanz-Aspekten auf der anderen Seite.

Als Anbieter einer PaaS-Lösung muss man sich weiterhin Gedanken um das Vertrauen der Tenants machen, die teilweise Geschäftsgeheimnisse wie ihre Prozessmodelle in fremde Hände geben sollen. [ALMS09] schlägt deshalb vor, eine Kombination aus einer Public Key Infrastructure (PKI) und einer (Teil-)Verschlüsselung der Prozessmodelle zu verwenden, um Geheimhaltung, Integrität und Authentizität zu gewährleisten. Weil BPEL auf XML basiert, können für die Verschlüsselung die W3C-Empfehlungen *XML-Encryption* [IDS02] und *XML-Signature* [BBF+08] eingesetzt werden. Damit die entsprechende BPEL-Engine wie z. B. Apache ODE damit zurechtkommt, müssten verständlicherweise alle betroffenen Komponenten (z. B. bezüglich des Deployments) angepasst werden.

Um sich noch tiefer mit der Materie auseinandersetzen zu können, wird an dieser Stelle auf [Mic06b] verwiesen. Dort werden z. B. asymmetrische und symmetrische Verschlüsselungsverfahren im Multi-Tenancy-Umfeld erläutert.

- *Nicht-autorisierter Zugriff*:
  Vor allem im Geschäftsumfeld, bei LOB-Services, muss eine sichere, zuverlässige Authentifizierung der Tenant-Administratoren und Tenant-Endbenutzer erfolgen.

- *Abschirmung/Isolation der Daten pro Tenant*:
  Wenn man wirkliche Isolation erreichen will, kann man sich nicht für den Ansatz eines gemeinsamen Datenbankschemas entscheiden. Aus verschiedenen Gründen, beispielsweise durch einen Programmierfehler hervorgerufen, könnte die falsche Tenant-ID gewählt werden und auf diese Weise andere Daten beeinflusst oder unberechtigterweise ausgelesen werden. Dies ließe sich zwar durch Verschlüsselung pro Tenant kompensieren, allerdings zu dem Preis des Aufwands der Ver- und Entschlüsselung.

- *Schutz vor physischem Zugriff*:
  Um den Tenants sogar Schutz vor ungewolltem Zugriff durch das eigene Personal des Anbieters (z. B. Administratoren) garantieren zu können, muss Sicherheit wirklich auf allen Ebenen erzielt werden. Eine spezielle Herausforderung hierbei stellen mögliche Speicherzugriffe dar: Weil Lese-Operationen im Hauptspeicher viel schneller sind als Zugriffe auf die Festplatte, belassen auch BPEL-Engines Daten zur aktuellen Prozessausführung und andere regelmäßig verwendete Daten im Hauptspeicher. Dort sind diese Daten allerdings anfällig für potentielle Mining-Algorithmen. Deswegen kann Vertrauen in dieser Hinsicht nur dann aufgebaut werden, wenn sichere Co-Prozessoren [SS01] zum Einsatz kommen.

Verfügbarkeit

Schauen wir uns zunächst einmal an, was überhaupt unter Verfügbarkeit zu verstehen ist.

Die Verfügbarkeit einer „normalen“ Single-Instance-Anwendung kann wie folgt definiert werden:

$$\textit{Verfügbarkeit} = \frac{MTTR}{MTTR + MTBF}$$

Hierbei ist Mean-Time Between Failure (MTBF) die Zeit zwischen zwei aufgetretenen Fehlern und Mean-Time To Repair (MTTR) steht für die Zeit, die benötigt wird, um einen aufgetretenen Fehler zu korrigieren.

Im Multi-Tenancy-Kontext muss die Gleichung natürlich angepasst werden. Angenommen N ist die Anzahl aller Tenants und X ist die Anzahl der „infizierten“ (sprich: fehlerhaften) Tenants. Dann kann die Verfügbarkeit nach [GSH+07] folgendermaßen angegeben werden:

$$\textit{Verfügbarkeit}_{MT} = 1 - \frac{X}{N} * \frac{MTTR}{MTTR + MTBF}$$

Hieraus lässt sich leicht ableiten, dass der Wert von X, also die Anzahl der fehlerhaften Tenants, von übergeordneter Bedeutung ist. Das Ziel sollte also darin bestehen, den Wert von $\frac{X}{N}$ möglichst klein zu halten.

Verfügbarkeit von Multi-Tenancy-Lösungen kann zu einem realen Problem werden. Offensichtliche Gefahren sind natürlich Ausfälle jeglicher Art (z. B. Hardware-Defekte), aber auch wenn einzelne Komponenten zum Flaschenhals werden, kann mitunter das ganze System zum Stillstand kommen, wie man am Beispiel von Amazon S3 erkennen kann, wo die Authentifizierungseinheit unvorhergesehenerweise überlastet wurde und für über zwei Stunden keine Anfragen mehr beantwortet werden konnten [Ste08]. Gerade weil Multi-Tenancy auf hohe Nutzerzahlen abzielt, können leichter Flaschenhälse auftreten.

Um eine hohe Verfügbarkeit sicherzustellen, bietet sich folgender Ansatz an, der auf mehreren Ebenen greift:

1. *Fehlererkennung und -diagnose*:
   Der erste Schritt zur Problembeseitigung ist, dass eine aufgetretene Unregelmäßigkeit im Betrieb überhaupt festgestellt wird und dass erkannt wird, wie viele und welche Tenants davon betroffen sind. Die Voraussetzung dafür ist natürlich, dass jeder Tenant einzelne Monitoring-Lösungen besitzt und die Ergebnisse zeitnah zur Monitoring-Instanz des gesamten Systems gesendet werden. Dieses kann beispielsweise durch einen sog. *Heartbeat* geschehen, ein periodisches „Lebenszeichen“ der Monitoring-Instanz.

2. *Vermeidung von Fehlerpropagierung*:
   Wenn erst einmal der Fehler bei einem oder mehreren Tenants erkannt wurde, muss ggf. verhindert werden, dass sich die Fehler auf andere Tenants übertragen und somit die „Infektionsrate“ $\frac{X}{N}$ ansteigt. Wenn auch kein allgemeiner Ansatz dafür gefunden werden kann, weil die Fehler sehr unterschiedlicher Natur sein können, so lässt sich auf jeden Fall festhalten, dass in kürzester Zeit alle Ressourcen der infizierten Tenants freigegeben werden sollten und dass Partitionierung in jeglicher Form natürlich dazu beiträgt, dass Fehler erst gar nicht propagierbar sind.

3. *Reparatur im laufenden Betrieb*:
   Gerade bei einer großen Anzahl von Tenants ist es absolut notwendig, dass die Behebung aller Probleme im laufenden Betrieb stattfinden sollte. Wenn nun ein Tenant fehlerhaft ist, stehen dem Anbieter die folgenden Alternativen zur Verfügung:

   - *Tenant-orientierter Service-Neustart*:
     Alle Ressourcen des entsprechenden Tenants werden freigegeben, ggf. werden auch eigene Instanzen des Tenants beendet und all seine Benutzer werden vom Service getrennt. Nun kann versucht werden, das eigentliche Problem zu beheben, indem man z. B. potentiell fehlerhafte Daten korrigiert. Letztendlich kann wieder „neugestartet“ werden, indem alle Instanzen erneut hochgefahren und Benutzer wieder aktiviert werden.

   - *Tenant-orientierte Service-Wiederherstellung*:
     Falls der Fehler nach Einsicht in die Daten immer noch nicht ersichtlich ist, bleibt nur noch die Möglichkeit, alle Daten des befallenen Tenants aus dem System zu löschen und den aktuellsten gespeicherten Stand aus dem Backup einzuspielen.

3.4 Bewertung von Multi-Tenancy-Ansätzen

In diesem Kapitel wollen wir die Gedanken über die verschiedenen Ansätze von Multi-Tenancy noch einmal übersichtlich zusammenfassen. Hierbei wird ein Bewertungsschema eingeführt, um die Ergebnisse unter einer konkreten Fragestellung besser miteinander vergleichen zu können.

Das Bewertungsspektrum besteht lediglich aus den drei Werten *niedrig* (dargestellt durch „•“), *mittel* (dargestellt durch „••“) und *hoch* (dargestellt durch „• • •“). Die Bewertungskriterien sind Tabelle 3.5 zu entnehmen.

Individualität	Mit diesem Punkt wird die Eigenschaft beschrieben, wie stark sich die Lösung spezifisch auf den einzelnen Tenant einzustellen vermag. Aus Kundensicht ist demnach ein möglichst hoher Wert erwünscht, während der Anbieter eher auf einen niedrigeren Wert aus ist, weil weniger individuelle Arbeit für einen Tenant anfällt, bzw. der Aufwand, um das System individuell gestaltbar zu machen, sehr hoch ist.
Skalierbarkeit	Ein System ist dann gut skalierbar, wenn es bei doppelter Last nur etwa doppelt so viele Ressourcen benötigt. Demzufolge ist eine höhere Bewertung in diesem Punkt positiv.
Isolation	In nur sehr spezifischen Fällen möchte man vielleicht explizit eine geringe Isolation haben, aber gerade in Multi-Tenancy-Szenarien wird man eine möglichst hohe Trennung zwischen den einzelnen Tenants erreichen wollen. Demzufolge ist eine hohe Isolation positiv.
Ressourcenverbrauch	In ausnahmslos allen Szenarien ist ein niedriger Ressourcenverbrauch erstrebenswert. Deshalb ist eine hohe Bewertung in diesem Punkt negativ.

Tabelle 3.5: Bewertungskriterien für Multi-Tenancy-Systeme

Native Multi-Tenancy

Die Bewertungen zu native Multi-Tenancy sehen folgendermaßen aus:

Individualität: •
Nur eine Instanz des Systems laufen zu lassen, bietet keinen großen Spielraum für Individualität.

Skalierbarkeit: • • •
Eine Stärke von native Multi-Tenancy ist, dass kein unnötiger Overhead dadurch entsteht, dass mehrere Instanzen des gleichen Programms gestartet werden müssen. Von daher ist dieser Ansatz geradezu prädestiniert, um gut zu skalieren.

Isolation: •
Die Isolation ist eindeutig sehr gering und könnte oftmals sogar ein Ausschlusskriterium für native Multi-Tenancy darstellen.

Ressourcenverbrauch: •
Native Multi-Tenancy geht im allgemeinen so sparsam wie nur möglich mit den verfügbaren Ressourcen um.

Fazit:
Der Grundgedanke hinter native Multi-Tenancy ist gut und kann in vielen Situationen Vorteile mit sich bringen. Es ist aber aufgrund der Nachteile leicht ersichtlich,

dass native Multi-Tenancy in vielen Szenarien gar nicht zu einer befriedigenden Lösung führen kann.

Multi-Instance Multi-Tenancy

Die Bewertungen zu Multi-Instance Multi-Tenancy sehen folgendermaßen aus:

Individualität: ••
Weil auf einzelne Anforderungen des Kunden eingegangen werden kann (z. B. mehr garantierter Hauptspeicher), dürfte sich die Individualität etwa im mittleren Bereich bewegen.

Skalierbarkeit: •(•)
Wegen der mehrfach instanziierten Komponenten dürfte sich die Skalierbarkeit eher im unteren bis mittleren Bereich bewegen.

Isolation: •(•)
Im Vergleich zu native Multi-Tenancy mit sehr geringer Isolation kommt es bei Multi-Instance Multi-Tenancy wahrscheinlich zu ein wenig mehr Isolation, sodass diese sich im unteren bis mittleren Bereich bewegen dürfte.

Ressourcenverbrauch: ••
Im Vergleich zu native Multi-Tenancy mit sehr geringem Ressourcenverbrauch benötigt dieser Ansatz gerade durch die Mehrfachinstanziierung mehr Ressourcen, sodass sich der Verbrauch insgesamt etwa im mittleren Bereich befinden dürfte.

Fazit:
In den angesprochenen Bewertungspunkten erhält dieser Ansatz eine ziemlich durchschnittliche Bewertung, was ihn zu einem guten Kompromiss werden lässt. Zusätzlich wird auch der Aufwand für die Betreuung des Systems im Vergleich zu native Multi-Tenancy gesteigert, aber dennoch erhält man die oft benötigte Flexibilität.

Heterogene Lösungen

Die Bewertungen zu Multi-Tenancy durch heterogene Lösungen sehen folgendermaßen aus:

Individualität: • • •
Heterogene Lösungen stehen gerade dafür, dass das eine oder andere Prinzip für Multi-Tenancy leicht aufgeweicht wird, um speziell auf Kundenbedürfnisse eingehen zu können. Von daher ist die Individualität hoch.

Skalierbarkeit: (● ● ●)
Aufgrund der Individualität lässt sich keine wirkliche Aussage darüber treffen, wie skalierbar das System letztendlich ist.

Isolation: ●●(●)
Da spezielle Lösungen für einzelne Tenants oder Gruppen von Tenants (Premium-Kunden) eingesetzt werden, wird mindestens eine mittlere Isolation stattfinden bis hin zur völligen Isolation.

Ressourcenverbrauch: (● ● ●)
Aufgrund der Individualität lässt sich keine wirkliche Aussage darüber treffen, wie hoch der Ressourcenverbrauch letztendlich ist, tendenziell dürfte er aber eher hoch ausfallen.

Fazit:
Auch wenn es nicht das Ziel sein sollte, eine solche heterogene Lösung umzusetzen, dürfte es in vielen Fällen dennoch hilfreich sein, diesen Ansatz im Hinterkopf zu behalten, damit man sich die Flexibilität behält, um auf Kundenwünsche reagieren zu können.

Single-Tenancy

Obwohl sich die bisherigen Untersuchungen komplett auf Multi-Tenancy bezogen haben, wollen wir trotzdem auch Single-Tenancy betrachten, damit in diesem Vergleich eine noch bessere Veranschaulichung stattfinden kann. Die Bewertungen hierzu sehen folgendermaßen aus:

Individualität: ●●
Das System muss lediglich damit zurechtkommen, mit einer Instanz einen Tenant bedienen zu können, weswegen es sicherlich leichter ist, die Software auf die Bedürfnisse zuzuschneiden, z. B. durch einmalige Konfiguration bei der Instanziierung. Allerdings sollte für alle Tenants die Code-Basis die gleiche sein, weswegen sich das System von der Funktionalität nur dann unterscheiden kann, wenn dieses durch Konfiguration geschieht. Daraus lässt sich schlussfolgern, dass Single-Tenancy in der Regel eine mittlere Individualität besitzt.

Skalierbarkeit: ● ● ●
Allgemein dürfte ein Single-Tenancy-System recht gut skalieren, weil der Ressourcenverbrauch sich in etwa proportional zur Anzahl der Tenants vervielfältigt.

Isolation: ●●(●)
Isolation durch logische Trennung findet bei Single-Tenancy in jedem Fall statt, allerdings könnte bei Fehlverhalten oder Absturz das ganze physische System betroffen sein, auf dem auch noch andere Tenant-Instanzen laufen. Je nachdem, wie

die einzelnen Tenant-Instanzen betrieben werden, ergibt sich deshalb eine mittlere bis hohe Isolation.

Ressourcenverbrauch: • • •
Der Ressourcenverbrauch bei Single-Tenancy kann sehr schnell sehr stark anwachsen, weil quasi keine Ressourcen zwischen den Tenants gemeinsam genutzt werden.

Fazit:
Single-Tenancy ist im Vergleich sehr einfach zu implementieren, denn es bedarf quasi keinerlei Anpassung unter Multi-Tenancy-Gesichtspunkten, bei der Entwicklung kann man sich voll und ganz auf Ein-Tenant-Betrieb einstellen. Obwohl die Skalierbarkeit allgemein gut ist, legt gerade der Speicherverbrauch immer wieder eine Obergrenze fest, bis zu wie viele Tenants bedient werden können. Diese Zahl ist meist sehr gering, was aus Anbietersicht natürlich unerwünscht ist.

3.4.1 Tabellarischer Vergleich

In Tabelle 3.6 sind nochmal einige wichtige Punkte aus dem Vergleich der verschiedenen Ansätze und Grade von Multi-Tenancy zusammengefasst. An dieser Stelle sei noch einmal darauf hingewiesen, dass bei Flexibilität, Skalierbarkeit und Isolation eine hohe und bei Ressourcenverbrauch eine niedrige Bewertung positiv ist.

( ) ⇒ *Möglichkeit* • ⇒ *niedrig* • • ⇒ *mittel* • • • ⇒ *hoch*	Flexibilität	Skalierbarkeit	Isolation	Ressourcen-verbrauch
Native Multi-Tenancy	•	• • •	•	•
Multi-Instance Multi-Tenancy	••	•(•)	•(•)	••
Heterogene Lösungen	• • •	(• • •)	••(•)	(• • •)
Single-Tenancy	••	• • •	••(•)	• • •

Tabelle 3.6: Übersicht über Eigenschaften der Grade von Multi-Tenancy

Aus dem Vergleich der verschiedenen Ansätze und Grade von Multi-Tenancy kann man nur zu dem Schluss kommen, dass es niemals *die* Lösung geben kann, die für alle Kunden in allen Situationen maßgeschneidert ist, denn die Kundenlandschaft ist dafür zu breit gefächert und heterogen. Effizienz auf Seiten des Anbieters und das optimale Kosten-/Nutzen-Verhältnis auf Seiten des Tenants spielen eine sehr wichtige Rolle. Aus diesem Grund muss Multi-Tenancy auf die konkrete Situation ausgerichtet sein und der Ansatz von Fall zu Fall neu erarbeitet werden. Ein solcher Entscheidungsprozess wird in Kapitel 3.6 skizziert.

3.5 Szenarien

Wir befinden uns nach wie vor in einem Multi-Tenancy-Umfeld und betrachten nun drei verschiedene Szenarien und welche Ansätze dabei vermutlich gewählt werden. Die angebotenen Services sind aufgrund von Multi-Tenancy nicht speziell für einen Kunden entwickelt, d.h. die Services erfüllen Aufgaben die weit verbreitet sind wie beispielsweise das CRM.

3.5.1 Massenmarkt

Szenario:

Die Firma *NicheSoft* will ihr Produktspektrum erweitern und analysiert den Markt, um sich in einer lukrativen Marktlücke zu positionieren. Dabei kommt heraus, dass es in vielen Unternehmen kleine logistische Aufgaben gibt, die meist von einzelnen Personen bearbeitet werden. Dabei kommen des öfteren Excel-Tabellen zum Einsatz, welche der Verantwortliche lokal auf seinem Rechner speichert und die ausschließlich für die reine Erfassung der Daten gebraucht werden können.

Das erste Produkt, was Nichesoft in diesem Bereich anbieten will, behandelt die Verwaltung der Getränke in einem Unternehmen. Es soll erfasst werden können, wie groß der aktuelle Stand an Kästen Mineralwasser, Packungen Tee, Flaschen Saft und Packungen Kaffee ist, ab welchem Stand eine neue Bestellung zu veranlassen ist und wer die Verantwortung für die Bestellung trägt. Weiterhin sollen Händler erfasst werden können, ggf. sogar mit der Möglichkeit, automatische Bestellungen absetzen zu können.

Lösungsansatz:

Der Service ist lediglich von geringem Umfang und betrifft keine geschäftlichen Aspekte der Kunden. Aus diesem Grund erwartet NicheSoft, keine großen Gebühren für den Service erheben zu können. Der Service zielt also auf den Massenmarkt ab und es sollten mit möglichst geringem Aufwand so viele Kunden wie möglich bedient werden können, weswegen sich NicheSoft für native Multi-Tenancy entscheidet. Weiterhin wird dieser Ansatz dadurch gestützt, dass Sicherheit der behandelten Daten keine hohe Priorität hat und die Performance des Systems auch keine entscheidende Rolle spielt.

Weil NicheSoft sich zwar Flexibilität erhalten will, aber kein Interesse daran hat, sich selbst um die Bereitstellung der Hardware zu kümmern, wird die Entscheidung getroffen, einen IaaS-Anbieter in Anspruch zu nehmen. Dieser bietet mehrere Ausbaustufen der Hardware an, sodass bei erhöhter Nachfrage zügig auf einen Server mit mehr Ressourcen und Rechenleistung gewechselt werden kann.

Varianten:

NicheSoft entwickelt die eigenen Produkte vollständig auf einer SaaS-Plattform wie force.com, um sich voll und ganz auf die Software selbst konzentrieren zu können. Außerdem wird davon profitiert, dass eine solche Plattform eine gute Anlaufstelle für potentielle Kunden darstellt.

3.5.2 Großkunden-Sektor

Szenario:

Das Softwarehaus *BigBusinessWare* entwickelt Lösungen für Zeitungsverlage. Die Produktpalette wird immer umfangreicher und auch der Kundenkreis wächst. Mittlerweile gehören viele namhafte, überregionale Zeitungen zur Kundschaft. Bereits von Beginn an hatte BigBusinessWare auf die Konfigurierbarkeit des Systems durch die einzelnen Kunden Wert gelegt, weshalb die Code-Basis für alle Verlage identisch ist. Das System ist in Form einer Client-/Server-Architektur aufgebaut.

Der bisherige Vertrieb der Produkte als On-Premise-Software wird momentan vom Markt nicht mehr angenommen, weil Verlage immer schwieriger abschätzen können, wie schnell die Anschaffungskosten amortisiert werden können oder ob sie sich mit dem Kauf womöglich überfordern.

Lösungsansatz:

Der Vorstand von BigBusinessWare ist optimistisch, dass sich der Markt mittelfristig erholen wird, sodass eine Übergangslösung eingeführt werden soll: Die Server-Software wird in den Rechenzentren von BigBusinessWare betrieben und die Kunden greifen über das Internet darauf zu. Um das Angebot attraktiv zu gestalten, bekommt ein Verlag den ersten Nutzungsmonat beitragsfrei und für jeden weiteren Monat wird eine Gebühr erhoben. Zudem soll es keine Mindestvertragslaufzeit geben und die monatlichen Gebühren können sogar auf einen späteren Kauf der Software angerechnet werden.

Varianten:

Die Anwendungen haben sich über die Jahre stetig weiterentwickelt und es gibt Teile des Systems, die nur noch mit erheblichem Aufwand wartbar sind. Aus diesem Grund entscheidet sich BigBusinessWare, das System von Grund auf neu als Software as a Service zu entwickeln.

3.5.3 Heterogene Kundenlandschaft

Szenario:

Die Firma *CloudSoft* bietet eine CRM-Lösung als SaaS an. Das System ist

als native Multi-Tenant-Software implementiert, das Geschäft läuft gut, zu den Kunden zählen namhafte SMB- und Großkunden gleichermaßen. Der eine oder andere Großkunde hat schon einmal verlauten lassen, dass er für Premium-Services bereit ist, einen Aufpreis zu zahlen.

Lösungsansatz:
Für Premium-Kunden kann auf zusätzlichen Servern eine weitere Installation des Systems eingerichtet werden. Je nach den Anforderungen können diese Kunden so auf einem Server zusammengefasst werden, dass eine optimale Auslastung erreicht wird. Auf diesem Weg können individuelle SLA mit den Kunden ausgehandelt werden.

Varianten:
Ein Premium-Kunde mit besonders hohen Ansprüchen könnte theoretisch auch einen eigenen Server zur Verfügung gestellt bekommen. Die Ressourcen-Auslastung ist dann ggf. nicht mehr in allen Bereichen optimal, aber so können die höchstmöglichen individuellen SLA garantiert werden.

Eine andere Variante ist mit dem eben erwähnten Modell verwandt, aber dennoch grundlegend verschieden. Die Ähnlichkeit besteht darin, ebenfalls eigene Server für einen Kunden bereitzustellen, allerdings diesmal in seinem eigenen Rechenzentrum, also Multi-Tenancy *on-premise*. Dies kann durch besonders hohe Anforderungen an Datensicherheit und Verfügbarkeit notwendig sein.

3.5.4 Sonderlösung: Einzelnes Großunternehmen

Szenario:
In einem großen Unternehmen existiert pro Abteilung ein gewisser Geschäftsprozess, der nur alle paar Wochen abgearbeitet wird. Für die IT-gestützte Erfassung und das Monitoring des Zustands wurden diese Geschäftsprozesse nach BPEL abgebildet. Das Unternehmen hat allerdings kein Interesse daran, eine dedizierte IT-Infrastruktur für diesen Workflow aufzubauen und zu betreuen.

Lösungsansatz:
Es wird ein Image mit einer installierten BPEL-Engine auf den Servern eines IaaS-Anbieter hochgeladen (vgl. Kap. 2.2.1). Für jede Abteilung wird dieses Image instanziiert. Über das Interface des IaaS-Anbieters kann jede Abteilung ihre Instanz eigenständig nach Bedarf starten und stoppen.

3.6 Umstellung auf Multi-Tenancy

In diesem Abschnitt wird darauf eingegangen, wie Umstellung auf Multi-Tenancy für Anbieter und Kunden aussehen könnte. Dabei wird erläutert, welche Fragen man sich als Software-Anbieter stellen sollte, wenn man bestehende Software multi-tenant-fähig machen oder von Beginn an Software mit Multi-Tenancy-Ansatz entwickeln will, wie dabei die Vorgehensweise ist und was es für Hindernisse geben kann, die gegen solch ein Vorhaben sprechen. Aus Kundensicht gibt es ebenfalls einige zu klärende Fragen, bevor man möglicherweise ein Multi-Tenancy-System benutzt, worauf wir hier eingehen werden.

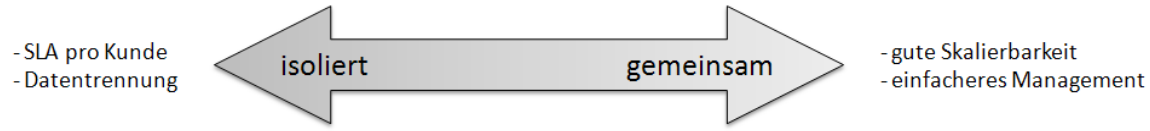


Abbildung 3.1: Wahl eines Reifegrades für Multi-Tenancy [Mic06a]

3.6.1 Aus Sicht des Anbieters

Als allererste Aufgabe hat der Anbieter zu klären, was seine konkrete Situation ist. Hat er bereits bestehende Software ohne Multi-Tenancy-Ansatz? Wenn ja, gibt es Probleme mit diesem Produkt, sodass eine Änderung der Situation herbeigeführt werden muss? Ist z. B. die Ressourcen-Auslastung mangelhaft (vgl. Kap. 3.1) oder ist das Software-System unkontrolliert gewachsen, sodass es nicht mehr wartbar ist? Wie wir gesehen haben, könnte Multi-Tenancy ein Lösungsansatz für derartige Probleme sein. *Schritt 1*

Eine weitere, denkbare Situation ist, dass ein neues Produkt entwickelt werden soll und der Anbieter fragt sich, ob er nun Multi-Tenancy verwenden soll oder nicht.

Im nächsten Schritt auf dem Weg zu Multi-Tenancy muss nun geklärt werden, für welche Zielgruppe die Software entwickelt werden soll. In Abb. 3.1 erkennt man, dass man sich für einen Ansatz entscheiden muss, dessen Eigenschaften oberste Priorität haben. Wie wir bisher gesehen haben, gibt es Lösungen über das gesamte Spektrum hinweg. Hierbei kann es notwendig sein, ein Profil von bestehenden und auch von potentiellen Kunden zu erstellen. Brauchen sie mehr Flexibilität? Achten sie vorrangig auf den günstigsten Preis? Diese oder ähnliche Fragen gilt es zu beantworten. *Schritt 2*

Wenn man sich dann entschieden hat, dass man auf Multi-Tenancy setzen will, muss man sich festlegen, wie die Software an die Kunden ausgeliefert werden soll (siehe Abb. 3.2). Hier lässt man nun die Ergebnisse aus dem vorherigen Schritt einfließen. *Schritt 3*

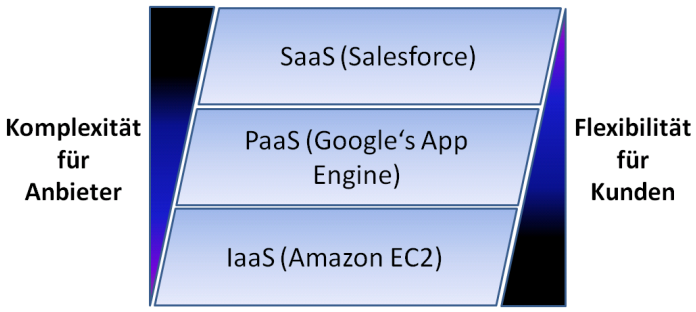


Abbildung 3.2: Wahl des Auslieferungsmodells [ALMS09]

Es gilt zu klären, wieviel Aufwand man betreiben und wie komplex das System aufgebaut sein muss, um ein optimales Kosten-/Nutzen-Verhältnis zu erzielen. Man muss sich dabei entscheiden, ob man ein komplett eigenes System entwickeln oder eine Plattform wie z. B. *force.com* oder Google's *App Engine* benutzen will. Ein Gedankenanstoß dazu wird in [Gun08] gegeben. Dort wird argumentiert, dass die Entwicklungszeit auf *force.com* um den Faktor fünf bis zehn kleiner ist als in einer klassischen Applikationsentwicklungsumgebung.

Nun wollen wir noch konkreter den Fall behandeln, bei dem man ein bestehendes Software-System hat, dass Multi-Tenancy-fähig gemacht werden soll. In [Mik08] wird darauf eingegangen, dass man dazu zunächst folgende Fragen beantworten muss:

1. *Code-Basis:*
   Habe ich eine Code-Basis oder wird für jeden Kunden eine separate Code-Basis verwaltet? Wenn eine gemeinsame Code-Basis besteht, ist der Grundstein für Multi-Tenancy bereits gelegt. Ist dies nicht der Fall, wird es abhängig vom Umfang der Unterschiede schwierig bis unmöglich werden, das System Multi-Tenancy-fähig zu machen.

2. *Applikationsinstanz:*
   Habe ich eine Applikationsinstanz, um alle Kunden zu bedienen oder hat jeder Kunde eine einzelne Instanz? Wenn lediglich eine Instanz läuft, ist das Ziel Multi-Tenancy schon so gut wie erreicht. Wenn nicht, muss geklärt werden, warum ursprünglich mehrere Instanzen benutzt wurden. Wie leicht ist die Umstellung auf nur eine laufende Instanz?

3. *Datenbank-Schema:*
   Habe ich ein Schema oder gibt es verschiedene Schemata für die einzelnen Kunden? Gibt es kein gemeinsames Schema, kann es sehr schwierig werden, nachträglich ein gemeinsames Schema zu gestalten. Dies könnte auf dem Weg, die Anwendung Multi-Tenancy-fähig zu machen, unter Umständen sogar zu einer unüberwindbaren Hürde werden.

4. *Datenbank-Instanz:*
   Habe ich eine Datenbank-Instanz oder verschiedene Instanzen für die einzelnen Kunden? Diese Frage ist zwar nicht von besonderer Wichtigkeit, allerdings hilft sie dabei, herauszufinden, ob sogar Umstellung auf native Multi-Tenancy möglich ist.

Weitere Fragen, die es sich als Anbieter zu stellen gilt:

5. *Know-How:*
   Habe ich das notwendige Wissen, um eine Multi-Tenancy-System zu entwickeln? Kann ich meine Entwickler schulen oder gibt es auf dem Arbeitsmarkt verfügbare Fachkräfte?

6. *Kosten/Nutzen:*
   Was ergibt eine Kosten-/Nutzen-Abschätzung? Lohnt sich der Aufwand? Sind die spezifischen Eigenschaften Multi-Tenancy den Aufwand wert? Gibt es alternative Ansätze, die mit weniger Aufwand zu erzielen sind? Reichen mir vielleicht schon Vorstufen, z. B. Single-Tenancy mittels virtueller Maschinen?

7. *Möglichkeit der Anpassung/Konfigurierbarkeit:*
   Wie viel Anpassung brauchen die Tenants und wie viel Anpassung lässt sich bei gleicher Code-Basis für alle Tenants realistisch mit normalem Aufwand ermöglichen?

Damit man das Rad nicht von Grund auf neu entwickeln muss, kann man möglicherweise schon auf bestehendes Wissen, Best Practices oder sogar bestehende Bibliotheken zurückgreifen. In [GSH+07] z. B. wird ein Framework vorgestellt, wie Multi-Tenancy erreicht werden kann. Dort heißt es, es sei sinnvoll, das Entwicklungsteam in zwei Gruppen aufzuteilen: die erste Gruppe kümmert sich um den geschäftsspezifischen Teil der Anwendung und programmiert gegen eine Schnittstelle, den *Multi-Tenancy Enablement Layer*, während Gruppe zwei aus Multi-Tenancy-Spezialisten besteht, die bereits Erfahrung in Bezug auf Datenbanken, Sicherheit und Skalierbarkeit bei Multi-Tenancy haben.

3.6.2 Aus Sicht des Kunden

Interessant ist natürlich, auch einmal die andere Seite zu betrachten und die Situation aus der Sicht des Kunden zu beurteilen. Ein Kunde stellt sich im Bezug auf Multi-Tenancy beispielsweise folgende Fragen, aus denen auch der Anbieter Schlüsse ziehen kann:

- Wie viel Anpassung am Multi-Tenancy-System ist möglich? Hat ein Anbieter einer Multi-Tenancy-Lösung das volle Potenzial zur Schaffung von Anpassungsmöglichkeiten ausgereizt? Oder handelt es sich zumindest um einen guten Kompromiss zwischen Aufwand und Nutzen?
- Mit welchen technischen Mitteln können die angegebenen SLAs garantiert werden?
- Wie sieht der Markt allgemein aus? Gibt es andere Anbieter mit einer ähnlichen technischen Umsetzung? Was sind dabei die konkreten Unterschiede? Oder wenn es ein ähnliches Angebot mit anderer Technologie im Hintergrund gibt, was sind dabei die Unterschiede?

  Die Technik ist für den Kunden zwar hintergründig, es kommt ihm mehr auf die QoS und die letztendliche Funktionalität an, allerdings kann es dem Kunden auch nicht egal sein, welche Technologien eingesetzt werden, denn er ist natürlich auch daran interessiert, dass beispielsweise am besten die Technologien eingesetzt werden, die sich bereits im längerfristigen Einsatz bewährt haben. Weiterhin ist es ihm ein wichtiges Anliegen, sich selbst ein Urteil über die Kosten-/Nutzen-Effizienz bilden zu können, damit er womöglich doch auf bessere oder günstigere Alternativen zurückgreifen kann.
- Ist die angebotene Software-Lösung wirkliches Multi-Tenancy, was für eine saubere Architektur sprechen würde, oder handelt es sich z. B. um ein „Fake-SaaS“-Angebot? Der Kunde sollte also nachfragen, wie viele Versionen, Instanzen und Infrastrukturen tatsächlich eingesetzt werden und wie die Anpassung für die einzelnen Tenants wirklich geschieht.

3.7 In Verbindung mit BPEL

Wir wollen nun verschiedene Ansätze diskutieren, wie man sich Multi-Tenancy auch im Bereich des Workflow-Management unter Verwendung von BPEL zunutze machen kann. Diese möglichen Ansätze sind hier nach den Auslieferungsmodellen aus Kapitel 2.2 gruppiert.

IaaS

Virtualisierung ist die Umsetzung von Multi-Tenancy auf Hardware-Ebene. Wenn BPEL demnach als Infrastructure as a Service bereitgestellt werden soll, sähe die Umsetzung so aus, dass ein Image bestehend aus Betriebssystem, ggf. lokalem DBMS (inkl. gleichem Datenbankschema), einer Installation der BPEL-Engine und deployten Prozessen erstellt und dieses Image mittels einer Virtualisierungslösung mehrfach instanziiert wird.

Abbildung 3.3 veranschaulicht diesen Sachverhalt. Der Anbieter stellt lediglich die Hardware und das Image zur Verfügung, die Verwaltung, Wartung und der Betrieb

der kompletten Softwareumgebung liegt vollständig in der Hand des Tenants. Auf der anderen Seite bietet das natürlich die größte Flexibilität, denn der Kunde kann z. B. ohne weiteres auch alle benötigten Services in dieser Umgebung installieren.

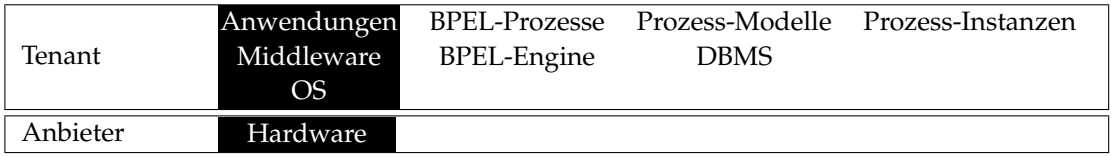


Abbildung 3.3: IaaS [ALMS09]

Bei einem PaaS-Ansatz wird hingegen bis hin zur BPEL-Engine alles vom Anbieter gestellt (siehe Abb. 3.4). Ob ein Tenant eigene Prozesse hochladen kann oder ob vorgegebene Prozesse pro Tenant extra deployt werden, spielt keine Rolle. Man erkennt leicht, dass dieser Ansatz weniger Aufwand für den Tenant bedeutet, dass allerdings auch Einschränkungen beispielsweise hinsichtlich der Prozessnamen auftreten können. *PaaS*

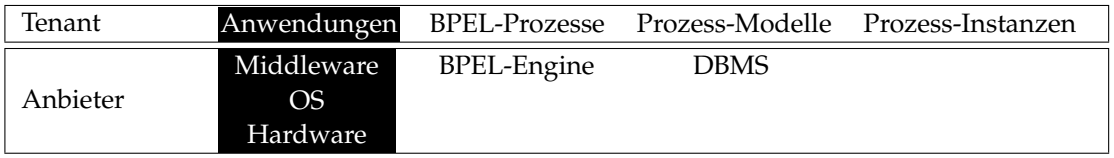


Abbildung 3.4: PaaS [ALMS09]

Als weitere Verschiebung in der Gewichtung zwischen Flexibilität und Aufwand für den Tenant (vgl. Abb. 3.1) wollen wir nun ein SaaS-Szenario betrachten. Wie in Abb. 3.5 dargestellt, könnte jeder Tenant auf die im System verfügbaren Prozesse zugreifen, indem er Nachrichten an die entsprechenden Endpunkte verschickt. Dadurch wird der Prozess instanziiert und die Instanz kann dem Tenant zugeordnet werden. Selbstverständlich muss diese Zuordnung bis zur Terminierung der Prozessinstanz ermöglicht werden. Dem Tenant ist zwar die Möglichkeit verschlossen, eigene Prozessmodelle zu verwenden, allerdings ist es in diesem Fall typisch, das System so weit wie möglich konfigurierbar zu gestalten. Es könnten demnach als Beispiel Endpunkte in der Prozdessdefinition konfigurierbar sein, sodass ein Tenant die Möglichkeit hat, auch eigene Web Services in den Prozess einzubinden. *SaaS*

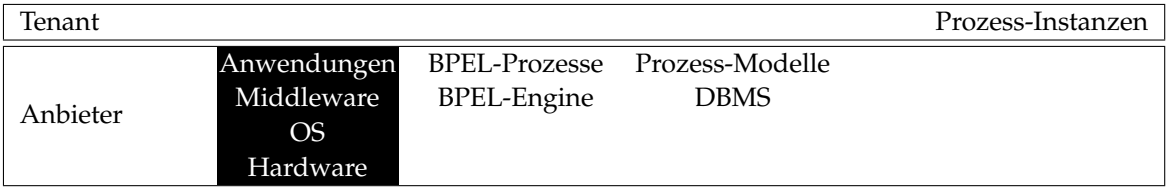


Abbildung 3.5: SaaS [ALMS09]

Kapitel 4

Konzept & Spezifikation

Das konkrete Szenario im praktischen Teil dieser Diplomarbeit ist, dass Geschäftsprozesse mit der BPEL-Engine Apache ODE unter Anwendung von Multi-Tenancy bereitgestellt werden. Die Überlegungen in Kap. 3.7 haben gezeigt, dass es eine Vielzahl an Kombinationsmöglichkeiten von Multi-Tenancy und einer BPEL-Engine gibt. In diesem Kapitel wird nun darauf eingegangen, welche dieser allgemeinen Ansätze tatsächlich mit der Apache ODE umgesetzt werden sollen. Hierfür wird zunächst in Abschnitt 4.1 das Konzept vorgestellt, welches einen groben Überblick des Systems vermitteln soll. Anschließend werden in Abschnitt 4.2 die konkreten Arbeitsabläufe im Umgang mit dem System beschrieben.

4.1 Konzept

Die Idee für das zu entwickelnde System (nachfolgend ODE^{MT} genannt) besteht darin, dass ein Anbieter eine BPEL-Engine für seine Kunden bereitstellen will, damit diese Geschäftsprozesse ausführen können. Bei der Umsetzung sollen unterschiedliche Multi-Tenancy-Ansätze zum Einsatz kommen. Die Anforderungen der Kunden werden heterogen sein, weswegen ODE^{MT} flexibel gestaltet werden soll, d. h. es soll möglich sein, ODE^{MT} im Rahmen von IaaS, PaaS und SaaS zu betreiben. Weiterhin soll eine Konfigurationsmöglichkeit existieren, mit deren Hilfe es möglich ist, zwischen diesen Betriebsmodi umzuschalten. Obwohl ODE^{MT} ein einziges System ist, werden die Ansätze in den Abschnitten 4.1.1 - 4.1.3 getrennt behandelt.

Die Apache ODE besitzt ein Web-Interface, über das sich Prozesse und Prozessinstanzen verwalten lassen (siehe Kap. 2.4.2.7). Dieses soll in ODE^{MT} ebenfalls Verwendung finden. Allerdings ist es nicht sinnvoll, ein ungesichertes Interface in einem Mehrbenutzersystem zu verwenden. Deshalb soll ODE^{MT} eine Benutzerverwaltung und einen Login-Mechanismus unterstützen. Grundsätzlich wird es drei Rollen von Benutzern geben, denen ein auf die entsprechende Rolle zugeschnittenes Interface zur Verfügung steht.

4.1.1 IaaS

Der erste Ansatz besteht darin, ODE^{MT} in einem IaaS-Szenario bereitzustellen. Das bedeutet, dass die modifizierte Apache ODE in ein Image verpackt wird, damit dieses durch eine der unterschiedlichen Virtualisierungslösungen (vgl. Kap. 2.2.1.1) instanziiert werden kann. Bei dieser Lösung handelt es sich um Multi-Instance Multi-Tenancy (siehe Kap. 3.2.3). Der Hypervisor stellt das Multi-Tenancy-System dar und die instanziierten Images sind die Multi-Instance-Komponenten. Letztendlich kann der Tenant auf die ihm zugeordneten virtuellen Maschinen zugreifen und hat völlige Flexibilität, um zusätzliche Software zu installieren und sich das komplette System nach seinen Vorstellungen einzurichten. In ODE^{MT} wird in diesem Fall der IaaS-Modus standardmäßig aktiviert sein, welcher sich vom Funktionsumfang und von der Arbeitsweise nicht von der Apache ODE ohne Multi-Tenancy unterscheiden soll, d. h., die Management-Services und das Web-Interface bleiben quasi unverändert.

Damit ODE^{MT} in einer Produktivumgebung eingesetzt werden kann, müsste zusätzlich ein Framework eingeführt werden, welches automatisiert Images für die verschiedenen Tenants deployen kann, sodass die Tenants selbst die Kontrolle über den Lebenszyklus der virtuellen Maschinen haben. Solch ein Framework geht allerdings über den Rahmen dieser Diplomarbeit hinaus. Deswegen wird hier nur auf bestehende Tools der jeweiligen Anbieter von Virtualisierungslösungen verwiesen (vgl. Kap. 2.2.1 und Kap. 2.2.1.1), mit deren Hilfe die Einrichtung und Verwaltung pro Tenant „manuell“ erfolgen muss. Da eine IaaS-Lösung aber grundlegend für Premium-Kunden angeboten wird, sollte sich hierbei die Anzahl der Tenants in einem überschaubaren Bereich bewegen.

Eine weitere Einschränkung ist, dass wir in dieser Ausarbeitung nicht über die Installation der grundlegenden Komponenten wie Betriebssystem, Apache ODE und Datenbank-Server innerhalb des Images hinaus gehen, weil das Image letztendlich immer auf die konkrete Situation angepasst werden muss. Dabei kann eine Verfeinerung der Konfiguration und Installation zusätzlicher Komponenten notwendig sein. Ein Beispiel dafür sind Sicherheitsanforderungen, wodurch man sich über Verschlüsselung der Nutzdaten Gedanken machen muss. Je nachdem, wie viele Tenants diese Besonderheiten benötigen und wie komplex diese sind, kann es sich lohnen, die Abweichungen vom Grundsystem bereits ins Image zu integrieren oder doch eher ein Anpassungsskript bzw. eine Modifikationsanleitung zur Verfügung zu stellen. Weiterhin steht natürlich die Möglichkeit offen, mehrere vorgefertigte Images zur Verfügung zu stellen, aus denen die Tenants auswählen können (vgl. Tab. 2.1).

Obwohl wir wie bereits angesprochen für IaaS kein Verwaltungsframework implementieren, müssen dennoch einige Aspekte spezifiziert werden. Dazu gehört z. B., dass der Tenant über ein Protokoll wie Secure Shell (SSH) oder Remote Desktop

Protocol (RDP) auf die virtuellen Maschinen zugreifen kann, um diese anzupassen.

4.1.2 PaaS

Wie wir in Kapitel 3.7 gesehen haben, bedeutet BPEL mit Multi-Tenancy im Rahmen von PaaS, dass den Tenants eigens deployte Prozessmodelle zugeordnet sind, diese aber alle von einer Instanz der Engine bereitgestellt werden. Demzufolge handelt es sich bei dieser Variante um native Multi-Tenancy (siehe Kap. 3.2.1).

Konkret soll ODE^{MT} unterstützen, dass ein Tenant sowohl eigene Prozessmodelle über das Web-Interface hochladen als auch Standard-Prozesse aus einer Bibliothek des Anbieters für sich deployen kann.

Das Web-Interface muss demnach so angepasst werden, dass lediglich der Zugriff auf eigene Prozessmodelle, Instanzen, Packages und Statistiken möglich ist. Außerdem wird deutlich, dass der Zugriff des Tenants ausschließlich über das Web-Interface möglich sein darf und er nicht wie beim IaaS-Ansatz aus Abschnitt 4.1.1 Zugriff auf das Installationsverzeichnis von ODE^{MT} bekommt.

Um solch ein Szenario konkret mit der Apache ODE umzusetzen, gibt es verschiedene Ansätze, die nachfolgend diskutiert werden. Da es sich dabei um Modifikationen der Deployment-Vorgehensweise handelt, betrachten wir zunächst das Original.

Das Deployment eines neuen Prozessmodells bei einer ODE-Installation ohne Multi-Tenancy läuft standardmäßig so ab, dass über das Web-Interface ein Zip-Archiv, die sog. *Deployment-Unit*, zum System übertragen wird. Eine alternative Möglichkeit besteht darin, die entpackte Deployment-Unit eigenhändig in das Prozessverzeichnis *<odedir>/processes* zu kopieren. Da der Zugriff auf das Prozess-Verzeichnis allerdings bei PaaS eingeschränkt sein sollte, bleibt nur die Variante, dass der Tenant-Administrator einen Prozess über das Web-Interface hochladen kann. Dabei muss die Engine dieses Prozessmodell eindeutig dem entsprechenden Tenant zuordnen können.

4.1.3 SaaS

Ein SaaS-Ansatz setzt ebenso wie der PaaS-Ansatz aus Abschnitt 4.1.2 native Multi-Tenancy um. Allerdings soll es bei SaaS lediglich Standard-Prozessmodelle geben, die von allen Tenants gleichermaßen genutzt werden. ODE^{MT} muss dazu unterstützen, dass einer Prozessinstanz Informationen zur Verfügung stehen, aus denen sie schließen kann, welcher Tenant diesen Prozess instanziiert hat. Das bedeutet, die Instanz muss wissen, aus welchem Kontext heraus sie aufgerufen wurde. Um dies zu

gewährleisten, soll sich der Tenant authentifizieren, wenn er einen Geschäftsprozess aufruft.

Anschließend könnte man sich überlegen, wozu diese Informationen gebraucht werden können. Eine Möglichkeit wäre, tenant-spezifische Daten aus externen Quellen nachzuladen. Außerdem können die Login-Daten quasi wiederverwendet werden, wenn es um Authentifizierung beim Aufruf externer Web Services geht. Hierfür werden allerdings keine Anforderungen an ODE^{MT} gestellt, da sich solche Lösungen an einer konkreten Situation orientieren müssen.

4.1.4 Sicherheit

Da bei PaaS und SaaS alle Tenants mit einer einzigen ODE-Instanz arbeiten, ist es bei diesen Lösungen wichtig, dass eine Absicherung der Management-API sowohl in der Implementierung der Services und ebenfalls an den Endpoints geschieht, da sonst der ganze Aufwand für Multi-Tenancy wirkungslos gemacht werden kann.

4.2 Spezifikation

In diesem Kapitel werden die konkreten Anwendungsfälle beschrieben, wie die Aktoren aus Abschnitt 4.2.1 mit ODE^{MT} interagieren können.

4.2.1 Aktoren

In dem zu entwickelnden System gibt es drei verschiedene Rollen, die in den nachfolgenden Anwendungsfällen als Aktoren Verwendung finden:

1. *Multi-Tenancy-Administrator*: Hierbei handelt es sich um das technische Personal des Anbieters der Multi-Tenancy-Lösung.
2. *Tenant-Administrator*: Dies ist der Administrator auf Seite des Kunden.
3. *Tenant-Endbenutzer*: Darunter versteht man die letztendlichen Benutzer des Systems auf Seite des Kunden.

4.2.2 Anwendungsfälle PaaS-Modus

Die Anwendungsfälle dieses Abschnitts beziehen sich ausschließlich auf den PaaS-Modus von ODE^{MT}.

Prozess aus Bibliothek deployen

Aktoren: Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Multi-Tenancy“ wird angezeigt (siehe Abbildung C.7).

Regulärer Ablauf:
1. Auswahl des Bibliotheks-Prozesses.
2. Klick auf den Button „Deploy“.

Nachbedingung: Der entsprechende Prozess wurde für den Tenant deployt.

Alternative Abläufe: -

Prozess hochladen

Aktoren: Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Deployment“ wird angezeigt (siehe Abbildung C.4).

Regulärer Ablauf:
1. Der Aktor gibt einen Namen für den Prozess ein.
2. Der Aktor wählt die hochzuladende Deployment-Unit aus.
3. Der Aktor klickt auf den Button „OK“.

Nachbedingung: Der Prozess wurde für den Tenant deployt.

Alternative Abläufe: -

4.2.3 Prozess-Management

In diesem Abschnitt werden Anwendungsfälle beschrieben, die sich allgemein auf das Prozess-Management beziehen. Der Multi-Tenancy-Administrator kann diese Anwendungsfälle in jedem Modus der Engine ausführen, der Tenant-Administrator hingegen nur im IaaS- bzw. PaaS-Modus.

Prozess aktivieren

Aktoren: Multi-Tenancy-Administrator, Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt, die Seite „Processes“ wird angezeigt (siehe Abbildung C.2) und der Prozess ist deaktiviert.

Regulärer Ablauf:
1. Klick auf den Button „Activate“.

Nachbedingung: Der Prozess ist wieder aktiviert.

Alternative Abläufe: -

Prozess deaktivieren

Aktoren: Multi-Tenancy-Administrator, Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt, die Seite „Processes“ wird angezeigt (siehe Abbildung C.2) und der Prozess ist aktiviert.

Regulärer Ablauf:
1. Klick auf den Button „Retire“.

Nachbedingung: Der Prozess ist nun deaktiviert.

Alternative Abläufe: -

Prozessmodell löschen

Aktoren: Multi-Tenancy-Administrator, Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Deployment“ wird angezeigt (siehe Abbildung C.4).

Regulärer Ablauf:
1. Klick auf den Button „Undeploy“ beim entsprechenden Prozess.

Nachbedingung: Der Prozess ist nun gelöscht.

Alternative Abläufe: -

4.2.4 Benutzer-Management

Die Anwendungsfälle dieses Abschnitts drehen sich um das Tenant- und Benutzer-Management und kommen hauptsächlich im PaaS-Modus zum Tragen.

Benutzer anlegen

Aktoren: Multi-Tenancy-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Multi-Tenancy“ wird angezeigt (siehe Abbildung C.6).

Regulärer Ablauf:
1. Eingabe des Benutzernamens.
2. Eingabe des Passworts.
3. Auswahl einer Rolle.
4. Eingabe einer Tenant-ID.
5. Klick auf den Button „Create User“.

Nachbedingung: Der Benutzer ist angelegt. Wenn die Tenant-ID zuvor noch nicht verwendet wurde, ist ebenfalls ein neuer Tenant angelegt.

Alternative Abläufe: -

Benutzer anlegen

Aktoren: Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Multi-Tenancy“ wird angezeigt (siehe Abbildung C.7).

Regulärer Ablauf:
1. Eingabe des Benutzernamens.
2. Eingabe des Passworts.
3. Klick auf den Button „Create User“.

Nachbedingung: Der Tenant-Benutzer ist angelegt.

Alternative Abläufe: -

Benutzer löschen

Aktoren: Multi-Tenancy-Administrator, Tenant-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Multi-Tenancy“ wird angezeigt (siehe Abbildungen C.6 & C.7).

Regulärer Ablauf:

1. Auswahl des zu löschenden Benutzers.
2. Klick auf den Button „Delete User“.

Nachbedingung: Der Benutzer ist gelöscht.

Alternative Abläufe: -

Tenant löschen

Aktoren: Multi-Tenancy-Administrator

Vorbedingung: Der Aktor ist eingeloggt und die Seite „Multi-Tenancy“ wird angezeigt (siehe Abbildung C.6).

Regulärer Ablauf:

1. Auswahl des zu löschenden Tenants.
2. Klick auf den Button „Delete Tenant“.

Nachbedingung: Der Tenant und alle zugehörigen Benutzer sind gelöscht.

Alternative Abläufe: -

Kapitel 5

Entwurf & Implementierung

Das vorliegende Kapitel bildet den Abschluss des praktischen Teils dieser Ausarbeitung. Darin werden zum einen die Entwurfsentscheidungen bei der Entwicklung von ODE^{MT} erläutert und zum anderen in einem groben Überblick die Implementierung beschrieben.

Eine grundlegende Entscheidung im Entwicklungsprozess stellte die Wahl der Code-Basis dar. Der erste Gedanke bei der Modifikation eines quelloffenen Software-Produkts ist natürlich, dass man sich die aktuellste Quelltext-Distribution herunterlädt, um darauf aufzubauen. Durch eingehende Literaturrecherche wurde jedoch immer deutlicher, wie wichtig Sicherheit im Multi-Tenancy-Umfeld ist. Von daher lag es nahe, auf den Quelltext von [Gör09] aufzusetzen, weil darin z. B. eine Absicherungsmöglichkeit der Management-Services bereits integriert war.

5.1 Verwendete Technologien

Die Entwicklung des Prototyps wurde auf einem x86-kompatiblen PC unter Microsoft Windows XP Service Pack 2 durchgeführt. Hierbei kamen Eclipse in der Version 3.5.1 (Codename „Galileo“), Java mit dem Java Development Kit (JDK) 1.5.0 Update 21, Apache Tomcat in der Version 5.5.28 und der MySQL-Server in der Version 5.1 zum Einsatz.

Die Java-Version ist in sofern wichtig, als dass Apache ODE zwar auch mit Java 1.6 ausführbar, allerdings nur mit Java 1.5 kompilierbar ist. Die restlichen Programm-Versionen dürften nicht maßgebend sein.

Der Prototyp ist außerdem unter Ubuntu/Debian auf den Servern von Amazon EC2 getestet worden.

5.2 Datenbank

Damit MySQL benutzt werden kann, muss zunächst der MySQL-Treiber `\Tools\mysql-connector-java-5.1.10-bin.jar` von der beiligenden CD nach `$CATALINA_HOME\common\lib` kopiert werden. In ODE^{MT} wird MySQL für einen JDBCRealm (siehe Kap. 2.3.2) verwendet. Das Datenbank-Schema für diesen Realm (siehe Listing 2.6) haben wir für ODE^{MT} leicht abgeändert, indem wir in der User-Tabelle einen Tenant-Identifier hinzugefügt haben (Listing 5.1; vgl. Kap. 3.2).

```
CREATE TABLE ‘users‘ (
  ‘user_name‘ varchar(32) NOT NULL,
  ‘user_pass‘ varchar(32) NOT NULL,
  ‘user_groupid‘ int(11) NOT NULL DEFAULT ’−1’,
  ‘tenantId‘ varchar(32) NOT NULL,
  PRIMARY KEY (‘user_name‘)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Listing 5.1: User-Tabelle mit Tenant-Identifier für JDBCRealm

Damit der JDBCRealm in ODE^{MT} aktiviert wird, muss eine Datei `\META-INF\context.xml` mit dem Inhalt aus Listing 5.2 erstellt werden. Im Zuge des Deployments der ODE-Engine wird diese Datei nach `$CATALINA_HOME\conf\Catalina\<hostname>\<Anwendungs-Name>.xml` kopiert. Beim erneuten Deployment von ODE^{MT} müsste die `ode.xml` manuell gelöscht werden, falls es Änderungen an der `context.xml` geben sollte, da die Datei nicht überschrieben wird.

```
<?xml version="1.0" encoding="UTF−8"?>
<Context docBase="${catalina.home}/webapps/ode"
        privileged="false" antiResourceLocking="false" antiJARLocking="false">

 <Realm className="org.apache.catalina.realm.JDBCRealm" debug="99"
         driverName="org.gjt.mm.mysql.Driver"
         connectionURL=
            "jdbc:mysql://localhost:3306/<DBNAME>?user=<USER>&amp;password=<PASS>"
         userTable="users"
         userNameCol="user_name"
         userCredCol="user_pass"
         userRoleTable="user_roles"
         roleNameCol="role_name"/>
</Context>
```

Listing 5.2: Definition eines JDBCRealm innerhalb eines <Context>

5.3 Umsetzung von Multi-Tenancy

In diesem Kapitel wollen wir besprechen, wie sich die Konzepte aus Kapitel 4.1 umsetzen lassen. Der erste Gedanke daraus befasst sich mit der Realisierung der drei Betriebsmodi. Dazu ist im Deployment-Deskriptor der Apache ODE eine Umgebungsvariable (siehe Kap. 2.3.1) eingeführt worden, wodurch sich die Engine zwischen den drei in Tab. 5.1 aufgeführten Modi umschalten lässt.

Wert der Variablen `ENGINE-TYPE`	Modus	Kapitel
„iaas“	IaaS-Modus	5.3.1
„paas“	PaaS-Modus	5.3.2
„saas“	SaaS-Modus	5.3.3

Tabelle 5.1: Modus-Konfiguration durch Umgebungsvariable

In den kommenden Abschnitten werden wir uns die Betriebsmodi getrennt voneinander anschauen.

5.3.1 Prototyp IaaS

Die Umsetzung von ODE^{MT} geschieht als EC2-Image. In Kap. 2.2.1.1 haben wir bereits beschrieben, wie ein eigenes Image erzeugt werden kann. Für die Implementierung wurde entschieden, die entsprechenden Komponenten auf der Grundlage eines frei verfügbaren Ubuntu-Image zu installieren und zu konfigurieren. Im Auslieferungszustand ist der `iaas`-Modus aktiviert.

Die Arbeitsweise mit ODE^{MT} weist im Vergleich zur Standard-Engine ohne Multi-Tenancy kaum Unterschiede auf. Die Web Services behalten alle Methoden, da diese durch Einführung von Multi-Tenancy nicht angepasst wurden. Lediglich die Benutzung des Web-Interfaces ist in sofern verändert, als dass sich ein Benutzer einloggen muss und somit eine unterschiedliche Oberfläche für den Anbieter (Rolle „Multi-Tenancy-Administrator“) und den Tenant (Rolle „Tenant-Administrator“) zur Verfügung steht. An der Funktionalität unter der Oberfläche ändert sich nichts, da es sich innerhalb der EC2-Instanz um einen Ein-Benutzer-Betrieb handelt.

5.3.2 Prototyp PaaS

Im Falle von PaaS gibt es einige Entwurfsentscheidungen zu treffen. Die Fragestellung ist, wie man das Deployment regelt, damit Prozessmodelle eindeutig einem Tenant zugeordnet werden können.

Es gibt zwei Wege, wie Apache ODE erfährt, dass ein neues Prozessmodell verfügbar ist. Zum einen existiert eine Komponente namens *DeploymentPoller*, welche Änderungen auf dem Prozess-Verzeichnis überwacht und auf diesem Wege den sog. *ProcessStore* informieren kann, sobald ein neu zu deployender Prozess zur Verfügung steht. Der zweite Weg geht über den *DeploymentService*. Dieser entpackt das hochgeladene Archiv und informiert den *ProcessStore* selbstständig. Der ProcessStore ist es jeweils, der im Anschluss u. a. den BPEL-Compiler (siehe Kap. 2.4.2.1) startet, damit der kompilierte Prozess verwendet werden kann.

Aus diesem Sachverhalt lassen sich zwei Lösungsansätze ableiten, um das Zuordnungsproblem der Prozessmodelle zum Tenant zu lösen.

Die erste Möglichkeit besteht darin, dass mehrere Prozess-Verzeichnisse existieren. Jeder Tenant hätte sein eigenes Verzeichnis, worauf ein eigenständiger Poller lauscht. Das Problem ist allerdings der proportionale Ressourcenverbrauch zur Anzahl der Tenants. Da man bei PaaS (ebenso bei SaaS) auf möglichst hohe Nutzerzahlen bedacht ist, kann dieser Ansatz bereits wieder verworfen werden.

Die zweite Möglichkeit sieht eine Zugehörigkeit über Prozessnamen vor. Dieses Konzept kann sowohl bei Bibliotheks-Prozessen als auch bei eigens hochgeladenen Prozessmodellen greifen, ohne dass der normale Deployment-Vorgang verändert werden muss. Bevor das Prozessmodell durch die Engine kompiliert wird, müssen die tenant-spezifischen Änderungen bereits an der Deployment-Unit durchgeführt sein. Weil es keine Ausschlusskriterien bei diesem Ansatz gibt, haben wir uns bei ODE^{MT} hierfür entschieden. Für die Umsetzung bietet ODE^{MT} zwei Varianten an. Zum einen bietet der MultiTenancyService (siehe Kap. 5.4.1) eine eigene `tenantDeploy`-Methode an, die grob gesehen die gleiche Aufgabe wie die `deploy`-Methode des DeploymentServices übernimmt, allerdings vor dem eigentlichen Deployment das Namensproblem löst. Zum anderen gibt es die `deployLibraryProcess`-Methode, die das Namensproblem ebenfalls vor dem normalen Deployment löst. In ODE^{MT} wurde eine Änderung des Namens mit der Tenant-ID als Präfix umgesetzt.

Es reicht allerdings nicht aus, lediglich den Namen des Prozessmodells zu ändern, vielmehr müssen die verschiedenen Elemente der Deployment-Unit (vgl. Kap. 2.4.2.1) angepasst werden.

Zu allererst darf es keine gleichnamigen Deployment-Units (auch *Packages* genannt) geben. Aus diesem Grund bekommen diese die Tenant-ID als Präfix in den Namen. Über dieses Namensschema kann daraufhin eine Filterung der verschiedenen Packages erfolgen, damit ein Tenant-Administrator im Web-Interface (siehe Kap. 5.4.2) auf der „Deployment“-Seite ausschließlich die eigenen Packages angezeigt bekommt.

Ein Prozess ist über eine in der entsprechenden WSDL-Datei definierte URL erreichbar. Um keine Kollisionen bei den Prozess-Endpoints zu erhalten, muss diese

Adresse angepasst werden. ODE^{MT} stellt die Tenant-ID deshalb auch dem Service-Namen voran (siehe Listing 5.3).

```
<definitions ... targetNamespace=
    "http://www.se.uni-hannover.de/soa08/tutorial/t44DynamicInvokeProcess">
  ...
  <service name="MultilingualHelloWorld">
    <port name="MultilingualHelloWorldPort" binding="tns:MultilingualHelloWorldBinding">
      <soap:address
          location="http://localhost:8080/ode/processes/t44MultilingualHelloWorld"/>
    </port>
  </service>
</definitions>
```

Listing 5.3: WSDL-Anpassung für Tenant-ID „t44“

Über XML-Namespaces und den Target-Namespace kann eine BPEL-Engine unterscheiden, welche Nachricht zu welchem Service gehört, wenn die Nachrichten den gleichen Namen besitzen. In sofern ist es ebenfalls wichtig, dass der Target-Namespace in der WSDL-Datei und im BPEL-Prozess und der korrespondierende XML-Namespace im Deployment-Deskriptor abgeändert werden. Auch hier empfiehlt es sich, den in der Namespace-URL enthaltenen Prozessnamen mit der Tenant-ID als Präfix zu versehen.

Weiterhin hat auch der BPEL-Prozess selbst einen zu ändernden Namen, weil im Web-Interface ebenfalls eine Filterung auf den Seiten „Processes“ und „Instances“ erwünscht ist. Alternativ ließe sich diese Filterung theoretisch ebenfalls über Properties eines Prozessmodells erreichen. Dazu wäre es allerdings notwendig, direkt in den Deployment-Prozess der Apache ODE einzugreifen, welches nach Möglichkeit vermieden werden sollte. Um den Namen des Prozesses zu ändern, muss in der `.bpel`-Datei und im Deployment-Deskriptor `deploy.xml` jeweils das Attribut `name` des `<process>`-Elements angepasst werden. Auch hier verwendet ODE^{MT} die Tenant-ID als Präfix.

Ein komplettes Beispiel eines Deployments für einen Tenant wird in Kapitel A besprochen.

ODE^{MT} unterstützt außerdem den Upload eigener Prozessmodelle durch den Tenant-Administrator. Hierbei werden die Elemente der Deployment-Unit ebenfalls wie bei den Bibliotheksprozessen namentlich abgeändert, sodass eine eindeutige Zuordnung und Filterung von Packages, Prozessmodellen und Instanzen möglich ist.

5.3.3 Prototyp SaaS

In [Gör09] werden verschiedene Wege ermöglicht, wie ein Benutzer-Kontext in eine Prozessinstanz integriert werden kann. Dieser Kontext lässt sich ohne Modifikationen für den Multi-Tenancy-Ansatz bei SaaS verwenden.

Die BPEL-Engine aus [Gör09] ist z.B. in der Lage, die Benutzerinformationen bei der Authentifizierung durch die Middleware entweder in Variablen oder in einem Benutzer-Kontext abzulegen. Weiterhin können auch statische Login-Informationen im Deployment-Deskriptor definiert werden.

5.3.4 Sicherheit

Wie [Gör09] hervorhebt, kann das Web-Interface der Apache ODE lediglich mit Soap 1.1 Nachrichten umgehen, weswegen bislang keine Unterstützung für WS-Security und WS-SecurityPolicy möglich ist. [Gör09] merkt deswegen an, dass das Web-Interface in dieser Form in einer sicherheitsrelevanten Produktivumgebung mit abgesicherten Management-Services nicht benutzt werden kann. In ODE^{MT} besteht dennoch die Möglichkeit, alle Management-Services per WS-Security abzusichern. Dazu muss lediglich der Datei `security.xml` im `webapps`-Wurzelverzeichnis mit Inhalt gefüllt werden.

5.3.5 Benutzerverwaltung

In Kap. 4.2.4 wurden die Anwendungsfälle zur Benutzerverwaltung beschrieben. Dieses dient zum einen der Zugriffskontrolle auf das Web-Interface und zum anderen werden über die Informationen des eingeloggten Users (Rolle, Tenant-ID, ...) die einzelnen Funktionen im Web-Interface verfügbar gemacht oder verborgen.

ODE^{MT} erwartet von jedem Benutzer, dass ihm eine Rolle zugewiesen ist, die in der Liste der Aktoren aus Kapitel 4.2.1 vorkommt („Multi-Tenancy-Administrator“, „Tenant-Administrator“ oder „Tenant-User“).

5.4 Erweiterung der Apache ODE

Die aktuelle stabile Version der Apache ODE ist 1.3.3, welche am 08. August 2009 veröffentlicht wurde. Die Grundlage dieser Ausarbeitung ist allerdings der Quelltext aus [Gör09]. Dort kam die Entwicklerversion vom 19. November 2008 (Revision 718763) zum Einsatz.

Es wurde nach Möglichkeit vermieden, bestehenden Code zu ändern, weswegen insgesamt nur wenige Änderungen an bestehenden Klassen der Apache ODE durchgeführt wurden. Als Erweiterungen der Engine sind der Multi-Tenancy-Service und die Multi-Tenancy-Funktionalität des Web-Interfaces zu nennen.

5.4.1 MultiTenancyService

In Anlehnung an die bestehenden Web Services der Apache ODE (siehe Beschreibung in Kap. 2.4.2.6), wurde in dieser Arbeit ein weiterer Service mit dem Namen „MultiTenancyService“ eingeführt, der alle Operationen zur Verfügung stellt, die Multi-Tenancy betreffen. Nachfolgend werden diese Operationen aufgelistet und erläutert.

createUser: Erwartet als Parameter eine User-ID, ein Passwort und eine Rolle (siehe Kap. 4.2.1) und liefert einen Bool'schen Wert zurück, ob der User angelegt werden konnte.

deleteUser: Erwartet als Parameter einen Benutzernamen und liefert einen Bool'schen Wert zurück, ob der User gelöscht werden konnte.

deleteTenant: Erwartet als Parameter eine Tenant-ID und liefert einen Bool'schen Wert zurück, ob der Tenant und alle dazugehörigen Benutzer gelöscht werden konnten.

deployLibraryProcess: Erwartet als Parameter einen Bibliotheks-Prozessnamen und eine Tenant-ID und liefert einen Bool'schen Wert zurück, ob der entsprechende Bibliotheks-Prozess erfolgreich deployt werden konnte.

getLibraryProcesses: Der Aufruf erfolgt ohne Parameter und es wird eine Liste von Bibliotheks-Prozessnamen zurückgeliefert.

getTenantIds: Der Aufruf erfolgt ohne Parameter und es wird eine Liste der Tenant-IDs zurückgeliefert.

getUserList: Erwartet als optionalen Parameter einen Rollennamen und gibt eine Liste der registrierten Benutzernamen zurück. Falls eine Rolle angegeben wird, werden nur Benutzer mit dieser Rolle zurückgeliefert.

listDeployedTenantPackages: Erwartet als Parameter eine Tenant-ID und gibt eine Liste der Packagenamen zurück, die der entsprechende Tenant deployt hat.

tenantDeploy: Erwaret als Parameter genau wie die Methode `deploy` des DeploymentService einen Namen und ein Zip-Archiv, deployt das Package allerdings tenant-spezifisch.

5.4.2 Web-Interface

Das Web-Interface bestand bislang aus reinen HTML-Seiten, deren Content-Bereich dynamisch durch das Yahoo! User Interface Library Version 2 (YUI2) geladen und teilweise auch periodisch aktualisiert wurde. Um die Benutzerinformationen durch die Authentifizierung von Tomcat in die Seite integrieren zu können, wurden die HTML-Seiten bei ODE^{MT}komplett durch JSPs ersetzt. Die JavaScript-Bibliotheken wurden in sofern erweitert, als dass die Schnittstellen des MultiTenancyService (siehe Abschnitt 5.4.1) nach JavaScript abgebildet wurden, damit der Service aus dem Web-Interface aufgerufen werden kann. Hierzu muss der Service in der WSDL-Datei ein SOAP1.1-Binding verfügbar sein, denn YUI2 kann lediglich SOAP1.1-Nachrichten verschicken.

In Tabelle 5.2 ist kompakt dargestellt, welcher Rolle bei welchem Modus welche Funktionen zur Verfügung stehen.

	Multi-Tenancy-Administrator	Tenant-Administrator	Tenant-User
Home	*alle Modi:* Übersicht über die Anzahl Prozesse und Instanzen	*paas:* Übersicht über eigene Prozesse und Instanzen *iaas/saas:* -	
Processes	*alle Modi:* Übersicht über alle Prozessmodelle	*paas:* Übersicht über eigene Prozessmodelle *iaas:* Übersicht über eigene (=alle) Prozessmodelle *saas:* -	*alle Modi:* -
Instances	*alle Modi:* Übersicht über alle Prozessinstanzen	*paas:* Übersicht über eigene Prozessinstanzen *iaas:* Übersicht über eigene (=alle) Prozessinstanzen *saas:* -	*alle Modi:* -
Deployment	*alle Modi:* Übersicht über alle deployten Packages *iaas/saas:* Möglichkeit zum Hochladen von Packages	*paas:* Übersicht über eigene deployte Packages & Möglichkeit zum Hochladen von Packages *iaas:* Übersicht über eigene (=alle) deployte Packages & Möglichkeit zum Hochladen von Packages *saas:* -	*alle Modi:* -
Security	*alle Modi:* Sicherheitseinstellungen für alle Services	*paas:* Sicherheitseinstellungen für eigene Services *iaas:* Sicherheitseinstellungen für eigene (=alle) Services *saas:* -	*alle Modi:* -
Multi-Tenancy	*alle Modi:* Interface zum Anlegen und Löschen von Tenants	*paas:* Interface zum Anlegen und Löschen von Tenant-Usern & Deployment von Prozessen aus Bibliothek *iaas/saas:* -	*alle Modi:* -

Tabelle 5.2: Funktionen des Web-Interfaces

Kapitel 6

Zusammenfassung

Fassen wir die Ergebnisse dieser Ausarbeitung einmal zusammen. Wir haben am Anfang mit der Begriffserklärung von Multi-Tenancy begonnen und anschließend erklärt, dass wir BPEL und Multi-Tenancy miteinander verknüpfen wollen. In Kapitel 2 haben wir dann die Grundlagen aus dem Kontext von Multi-Tenancy und von BPEL angeschaut. Wir haben über Services und eine Service-orientierte Architektur gesprochen, haben uns verschiedene Auslieferungsmodelle angesehen, sind noch näher auf Virtualisierung eingegangen bis wir letztlich eine Analyse der Apache ODE durchgeführt haben.

Mit diesem Wissen war es dann möglich, Multi-Tenancy genauer zu untersuchen. Dort haben wir gesehen, dass es verschiedene Grade von Multi-Tenancy gibt, die sich beispielsweise am verwendeten Datenmodell orientieren. Außerdem wurde deutlich, dass die zuvor betrachteten Auslieferungsmodelle als gute Kategorisierungen dienen können und man BPEL-Engines in unterschiedlichen Szenarien auf verschiedenen Wegen an den Tenant bringen kann.

Mit diesem Wissen ging es dann in den praktischen Teil der Ausarbeitung, in welchem wir die erarbeiteten Konzepte konkret auf die Apache ODE beziehen konnten. Mit ODE^{MT} haben wir dann gezeigt, dass sich ein Prototyp entwickeln lässt, der für IaaS, PaaS und SaaS gerüstet ist. Auf diese Weise können Geschäftsprozesse in BPEL unter Verwendung von Multi-Tenancy zur Verfügung gestellt werden, womit wir das Ziel erreichen konnten: ODE^{MT} ist eine „multi-tenant-fähige BPEL-Engine“.

Anhang A

Beispiel-Prozess „DynamicInvoke“

Der *DynamicInvokeProcess* ist ein beispielhafter, in BPEL modellierter Geschäftsprozess. Der Ablauf beginnt mit einer eingehenden Nachricht, welche einen Parameter enthält, der die Sprache für einen später auszugebenden „Hello World!“-String enthält. Zwischendrin werden *if-else*-Konstrukte verwendet, um eine Variable abhängig von der Sprache mit einer entsprechenden Endpunkt-Adresse zu belegen. Diese Variable wird danach ausgelesen und der gewünschte „HelloWorldService“ aufgerufen. Als Sprachen akzeptiert der Prozess „ger“ für Deutsch, „esp“ für Spanisch, „fra“ für Französisch und bei allen anderen Werten wird Englisch verwendet.

Die Deployment-Unit dieses Prozesses für Apache ODE umfasst die folgenden Dateien, welche anschließend, teilweise leicht gekürzt, abgedruckt sind.

Datei	Funktion
deploy.xml	Deployment-Deskriptor
DynamicInvokeProcess.bpel	Prozess-Definition
DynamicInvokeProcessArtifacts.wsdl	Beschreibung des Web Services, über welchen der Prozess aufgerufen werden kann.
HelloWorldService.wsdl	Beschreibung des HelloWorld-Services, der auf den Servern der Universität Hannover ausgeführt wird.

Tabelle A.1: Inhalt der Deployment-Unit „DynamicInvoke“

Nachfolgend werden wir die einzelnen Bestandteile der Deployment-Unit im Quelltext betrachten. Darin werden Teile grau hinterlegt sein, um zu zeigen, was verändert werden muss, damit der DynamicInvokeProcess in einem PaaS-Szenario pro Tenant deployt werden kann (vgl. Kap. 5.3.2).

In Listing A.1 lässt sich der Deployment-Deskriptor nachlesen. Darin ist erkennbar, welche Web Services zur Verfügung gestellt (`<provide>`) und welche externen Services aufgerufen werden (`<invoke>`).

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy xmlns="http://www.apache.org/ode/schemas/dd/2007/03"
  xmlns:DynamicInvokeProcess=
  "http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess"
  xmlns:HelloWorldService=
  "http://www.se.uni-hannover.de/soa08/tutorial/HelloWorldService">
  <process name="DynamicInvokeProcess:DynamicInvokeProcess">
    <process-events generate="all"/>
    <provide partnerLink="client">
      <service name="DynamicInvokeProcess:MultilingualHelloWorld"
        port="MultilingualHelloWorldPort"/>
    </provide>
    <invoke partnerLink="HelloWorldServicePartnerLink">
      <service name="HelloWorldService:HelloWorldService" port="HelloWorldServicePort"/>
    </invoke>
  </process>
</deploy>
```

Listing A.1: Deployment-Deskriptor von DynamicInvokeProcess

In Listing A.2 handelt es sich um die XML-Repräsentation der grafischen Darstellung des Prozesses aus Abbildung 2.7.

```
<!-- DynamicInvokeProcess [Generated by the Eclipse BPEL Designer] -->
<bpel:process name="DynamicInvokeProcess"
  targetNamespace=
  "http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess"
  suppressJoinFailure="yes"
  xmlns:tns="http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess"
  xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
  xmlns:ns="http://www.se.uni-hannover.de/soa08/tutorial/HelloWorldService"
  xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref">

  <!-- Import the client WSDL -->
  <bpel:import
    namespace="http://www.se.uni-hannover.de/soa08/tutorial/HelloWorldService"
    location="HelloWorldService.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />
  <bpel:import
    namespace="http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess"
    location="DynamicInvokeProcessArtifacts.wsdl"
    importType="http://schemas.xmlsoap.org/wsdl/" />

  <!-- ================================================================
       PARTNERLINKS
       List of services participating in this BPEL process
       ================================================================ -->
  <bpel:partnerLinks>
    <!-- The 'client' role represents the requester of this service. -->
```

```
27    <bpel:partnerLink name="client"
28      partnerLinkType=" tns:DynamicInvokeProcess "
29      myRole="DynamicInvokeProcessProvider" />
30    <bpel:partnerLink name="HelloWorldServicePartnerLink"
31      partnerLinkType="tns:HelloWorldPartnerLinkType"
32      partnerRole="HelloWorldProvider" />
33  </bpel:partnerLinks>
34
35  <!-- ================================================================
36       VARIABLES
37       List of messages and XML documents used within this BPEL process
38       ================================================================ -->
39  <bpel:variables>
40    <!-- Reference to the message passed as input during initiation -->
41    <bpel:variable name="input"
42      messageType="tns:DynamicInvokeProcessRequestMessage" />
43
44    <!-- Reference to the message that will be returned to the requester -->
45    <bpel:variable name="output"
46      messageType="tns:DynamicInvokeProcessResponseMessage" />
47    <bpel:variable name="HelloWorldServicePartnerLinkResponse"
48      messageType="ns:getMessageResponse" />
49    <bpel:variable name="HelloWorldServicePartnerLinkRequest"
50      messageType="ns:getMessageRequest" />
51    <bpel:variable name="HelloWorldEndpoint"
52      element=" sref:service -ref" />
53  </bpel:variables>
54
55  <!-- ================================================================
56       ORCHESTRATION LOGIC
57       Set of activities coordinating the flow of messages across the
58       services integrated within this business process
59       ================================================================ -->
60  <bpel:sequence name="main">
61    <!-- Receive input from requester.
62         Note: This maps to operation defined in DynamicInvokeProcess.wsdl
63    -->
64    <bpel:receive name="receiveInput" partnerLink="client"
65      portType=" tns:DynamicInvokeProcess "
66      operation="process" variable="input"
67      createInstance="yes"/>
68
69    <!-- Generate reply to synchronous request -->
70    <bpel:if name="ChooseLanguage" xmlns:http="urn:http:namesapce">
71      <bpel:condition><![CDATA[$input.payload/tns:input='ger']]></bpel:condition>
72      <bpel:assign validate="no" name="German">
73        <bpel:copy>
74          <bpel:from>
75            <bpel:literal xml:space="preserve">
76              <sref:service-ref
77                xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref">
```

```
78                <EndpointReference
79                  xmlns="http://www.w3.org/2005/08/addressing">
80                  <Address>
81 http://hobbes.se.uni-hannover.de:8080/HelloWorldServices/services/
       GermanHelloWorldService
82                  </Address>
83                </EndpointReference>
84              </sref:service-ref>
85            </bpel:literal>
86          </bpel:from>
87          <bpel:to variable="HelloWorldEndpoint"></bpel:to>
88        </bpel:copy>
89      </bpel:assign>
90      <bpel:elseif>
91        <bpel:condition><![CDATA[$input.payload/tns:input='esp']]></bpel:condition>
92        <bpel:assign validate="no" name="Spanish">
93          ...
94        </bpel:assign>
95      </bpel:elseif>
96      <bpel:elseif>
97        <bpel:condition><![CDATA[$input.payload/tns:input='fra']]></bpel:condition>
98        <bpel:assign validate="no" name="French">
99          ...
100       </bpel:assign>
101     </bpel:elseif>
102     <bpel:else>
103       <bpel:assign validate="no" name="English">
104         ...
105       </bpel:assign>
106     </bpel:else>
107   </bpel:if>
108   <bpel:assign validate="no" name="AssignEndpoint">
109     <bpel:copy>
110       <bpel:from variable="HelloWorldEndpoint"></bpel:from>
111       <bpel:to partnerLink="HelloWorldServicePartnerLink"></bpel:to>
112     </bpel:copy>
113   </bpel:assign>
114   <bpel:invoke name="DynamicHelloWorldInvoke"
115     partnerLink="HelloWorldServicePartnerLink" operation="getMessage"
116     portType="ns:HelloWorldServicePortType"
117     outputVariable="HelloWorldServicePartnerLinkResponse">
118   </bpel:invoke>
119   <bpel:assign validate="no" name="AssignOutput">
120     <bpel:copy>
121       <bpel:from>
122         <bpel:literal xml:space="preserve">
123           <tns:DynamicInvokeProcessResponse
124             xmlns:tns="http://www.se.uni-hannover.de/soa08/tutorial/
                  DynamicInvokeProcess"
125             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
126             <tns:result></tns:result>
```

```
127             </tns:DynamicInvokeProcessResponse>
128           </bpel:literal>
129         </bpel:from>
130         <bpel:to variable="output" part="payload"></bpel:to>
131       </bpel:copy>
132       <bpel:copy>
133         <bpel:from part="parameters" variable="HelloWorldServicePartnerLinkResponse">
134           <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
135             <![CDATA[message]]>
136           </bpel:query>
137         </bpel:from>
138         <bpel:to part="payload" variable="output">
139           <bpel:query queryLanguage="urn:oasis:names:tc:wsbpel:2.0:sublang:xpath1.0">
140             <![CDATA[tns:result]]>
141           </bpel:query>
142         </bpel:to>
143       </bpel:copy>
144     </bpel:assign>
145     <bpel:reply name="replyOutput" partnerLink="client"
146       portType="tns:DynamicInvokeProcess" operation="process" variable="output" />
147   </bpel:sequence>
148 </bpel:process>
```

Listing A.2: DynamicInvokeProcess.bpel

Abschließend ist in Listing A.3 der Web Service beschrieben, über den sich der BPEL-Prozess aufrufen lässt.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
3    xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
4    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
5    xmlns:tns=
6    "http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess"
7    xmlns:vprop="http://docs.oasis-open.org/wsbpel/2.0/varprop"
8    xmlns:wsdl="http://www.se.uni-hannover.de/soa08/tutorial/HelloWorldService"
9    name="DynamicInvokeProcess"
10   targetNamespace=
11     "http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess">
12
13 <!-- ================================================================
14      TYPE DEFINITION - List of types participating in this BPEL process
15      The BPEL Designer will generate default request and response types
16      but you can define or import any XML Schema type and use them as part
17      of the message types.
18      ================================================================ -->
19   <plnk:partnerLinkType name="HelloWorldPartnerLinkType">
20     <plnk:role name="HelloWorldProvider" portType="wsdl:HelloWorldServicePortType"/>
21   </plnk:partnerLinkType>
22
23   <import location="HelloWorldService.wsdl"
```

```
24    namespace="http://www.se.uni-hannover.de/soa08/tutorial/HelloWorldService"/>
25  <types>
26    <schema xmlns="http://www.w3.org/2001/XMLSchema"
          attributeFormDefault="unqualified"
27      elementFormDefault="qualified"
28      targetNamespace=
29      " http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess ">
30      <element name="DynamicInvokeProcessRequest">
31        <complexType>
32          <sequence>
33            <element name="input" type="string" />
34          </sequence>
35        </complexType>
36      </element>
37
38      <element name="DynamicInvokeProcessResponse">
39        <complexType>
40          <sequence>
41            <element name="result" type="string"/>
42          </sequence>
43        </complexType>
44      </element>
45    </schema>
46  </types>
47
48 <!-- ================================================================
49     MESSAGE TYPE DEFINITION - Definition of the message types used as
50     part of the port type defintions
51     ================================================================ -->
52  <message name="DynamicInvokeProcessRequestMessage">
53    <part element="tns:t44DynamicInvokeProcessRequest" name="payload"/>
54  </message>
55  <message name="DynamicInvokeProcessResponseMessage">
56    <part element="tns:DynamicInvokeProcessResponse" name="payload"/>
57  </message>
58
59 <!-- ================================================================
60     PORT TYPE DEFINITION - A port type groups a set of operations into
61     a logical service unit.
62     ================================================================ -->
63
64 <!-- portType implemented by the DynamicInvokeProcess BPEL process -->
65  <portType name=" DynamicInvokeProcess ">
66    <operation name="process">
67      <input message="tns:DynamicInvokeProcessRequestMessage"/>
68      <output message="tns:DynamicInvokeProcessResponseMessage"/>
69    </operation>
70  </portType>
71
72 <!-- ================================================================
73     PARTNER LINK TYPE DEFINITION
```

```
74      ========================================================================== -->
75    <plnk:partnerLinkType name="DynamicInvokeProcess">
76      <plnk:role name="DynamicInvokeProcessProvider"
77        portType="tns:DynamicInvokeProcess"/>
78    </plnk:partnerLinkType>
79
80    <binding name="MultilingualHelloWorldBinding"
81      type="tns:DynamicInvokeProcess">
82      <soap:binding style="document"
83        transport="http://schemas.xmlsoap.org/soap/http" />
84      <operation name="process">
85        <soap:operation
86          soapAction=
87          "http://www.se.uni-hannover.de/soa08/tutorial/DynamicInvokeProcess/process"
              />
88        <input>
89          <soap:body use="literal" />
90        </input>
91        <output>
92          <soap:body use="literal" />
93        </output>
94      </operation>
95    </binding>
96    <service name="MultilingualHelloWorld">
97      <port name="MultilingualHelloWorldPort" binding="tns:MultilingualHelloWorldBinding">
98        <soap:address
99          location="http://localhost:8080/ode/processes/MultilingualHelloWorld"/>
100     </port>
101   </service>
102 </definitions>
```

Listing A.3: DynamicInvokeProcessArtifacts.wsdl

Anhang B

Installationsanleitung

Nachfolgend wird beschrieben, wie die Ergebnisse des praktischen Teils dieser Arbeit kompiliert werden können. Die entsprechenden Quelltexte sind auf der beiliegenden CD im Ordner `\Implementation\src` zu finden.

Um die Apache ODE aus dem Quelltext kompilieren zu können, muss zunächst das JDK von Sun Microsystems[1] in der Version 1.5 installiert werden. Die Apache ODE läuft zwar auch mit Java 1.6, allerdings sind Teile des Projekts nicht damit übersetzbar. Anschließend muss sichergestellt werden, dass die Umgebungsvariable `JAVA_HOME` gesetzt ist und auf das Wurzelverzeichnis der JDK-Installation verweist. Die Kompilierung und das Packaging geschieht mittels *Apache Buildr*[2]. Eine Voraussetzung dafür ist, dass Ruby[3] installiert wurde. Wenn dieses der Fall ist, kann Buildr durch den Befehl `gem install buildr -v <VERSION>` installiert werden. Die in dieser Arbeit verwendete Version ist 1.3.1. Buildr verwendet ein sog. *Rakefile*, ein Ruby-Skript mit einem bestimmten Aufbau. Im Übersetzungsvorgang werden weiterhin Subversion-Kommandos abgesetzt, weswegen zwangsläufig ein Kommandozeilen-Client für Subversion installiert sein muss. Unter Windows bietet sich hierfür z. B. SlikSVN[4]) an. Um den Build-Vorgang zu starten wechselt man in einer Shell in den Trunk-Ordner der Apache ODE und für den Befehl `buildr _1.3.1_ TEST=no ode:package` aus. Das „1.3.1“ steht hierbei wiederum für die eingesetzte Version von Buildr.

Um Apache ODE weiterhin modifizieren zu können, lassen sich durch den Befehl `buildr eclipse` Projekt-Dateien für Eclipse erstellen. Hierbei scheint Buildr allerdings keine `.project`-Datei für `ode-bpel-scripts` anzulegen, weswegen man sich von dieser Datei eine Kopie aus dem Trunk des SVN-Repositories der Apache ODE besorgen sollte.

Weiterhin müssen folgende Einstellungen in Eclipse vorgenommen werden:

[1] `http://java.sun.com`
[2] `http://buildr.apache.org/`
[3] `http://rubyinstaller.org/`
[4] `http://www.sliksvn.com`

- Man muss `M2_REPO` als „classpath variable“ einrichten und diese auf das lokale Maven2-Repository zeigen lassen. Solche Variablen lassen sich im Dateimenü `Window > Preferences > Java > Build Path > Classpath Variables` setzen.
- Es muss sichergestellt werden, dass auch wirklich das JDK 1.5 ausgewählt ist, noch bevor man die Projekte importiert. Weiterhin muss die Datei `tools.jar` als externe JAR-Datei im Build-Path der dazugehörigen JRE liegen.

Selbst wenn man in Eclipse kompiliert, muss für einige Elemente, wie z. B. JACOB Channel Interfaces, zusätzlich `buildr` aufgerufen werden, weil das Build-System von Eclipse nicht alle Komponenten erfasst und der Build-Vorgang speziell für das Rakefile konzipiert wurde.

Anhang C

Screenshots

Nachfolgend sind die Screenshots des Web-Interfaces (siehe Kap. 5.4.2) abgebildet, um sich einen visuellen Eindruck der Oberfläche verschaffen zu können.

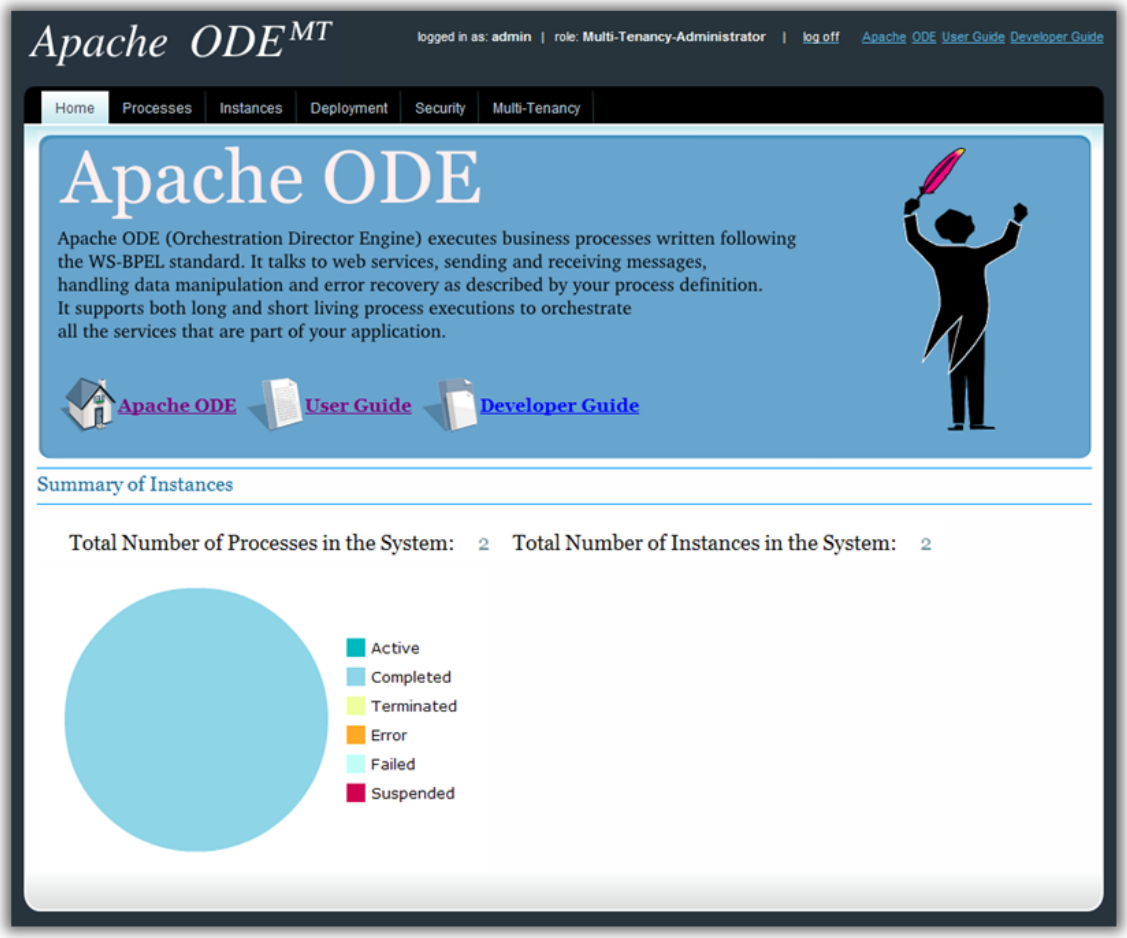


Abbildung C.1: Web-Interface Seite „Home“

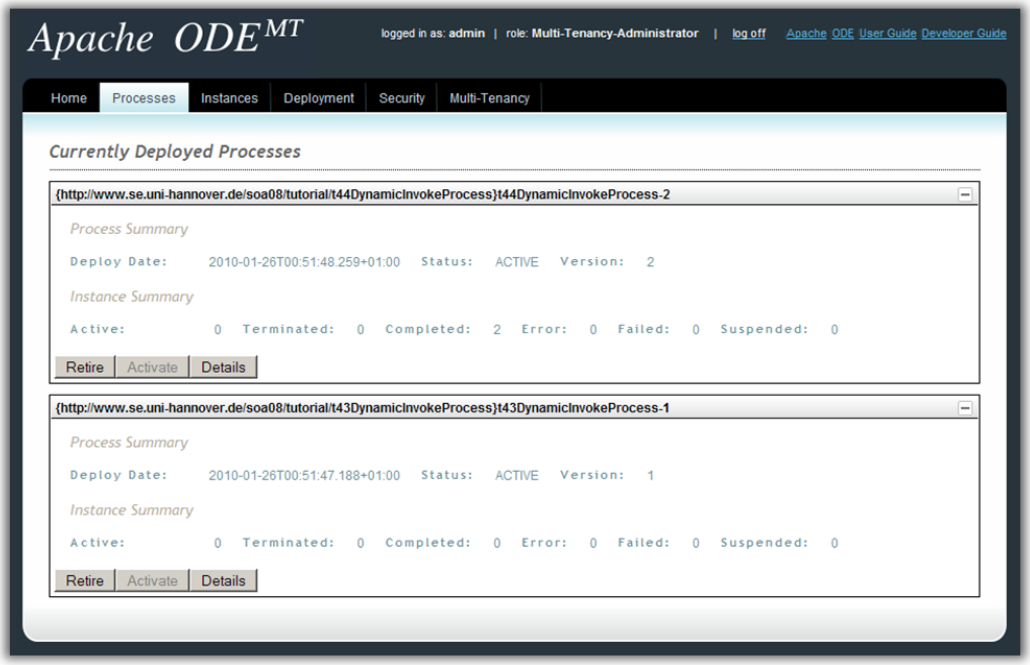


Abbildung C.2: Web-Interface Seite „Processes“

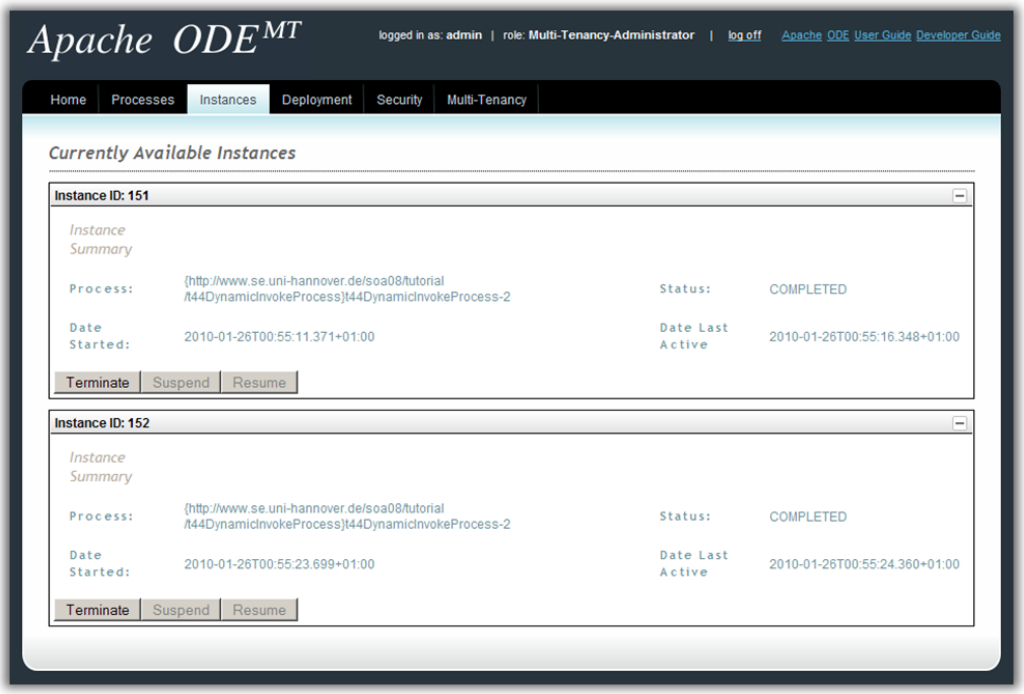


Abbildung C.3: Web-Interface Seite „Instances“

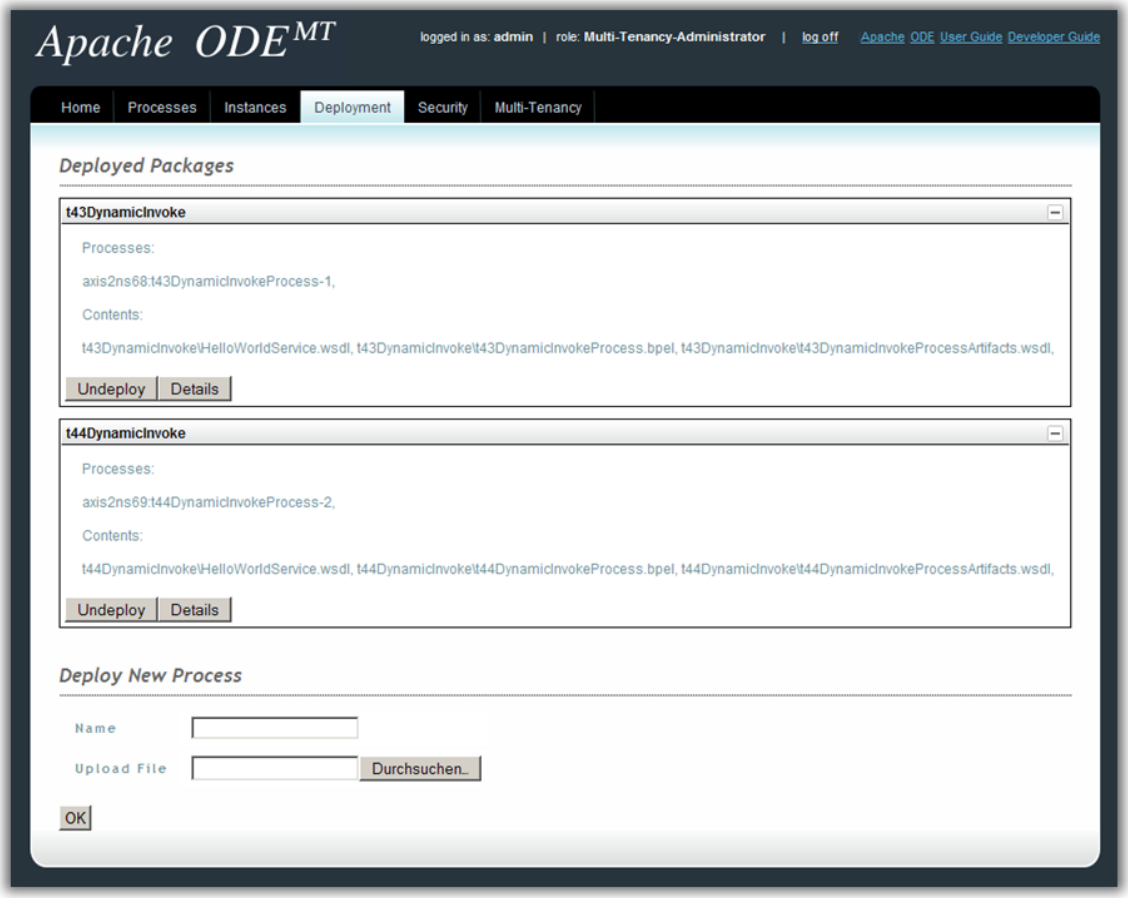


Abbildung C.4: Web-Interface Seite „Deployment“

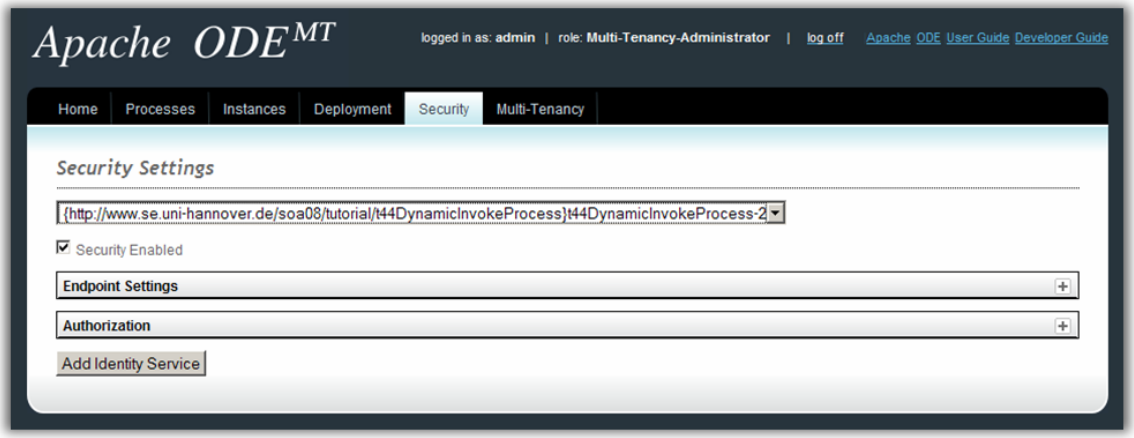


Abbildung C.5: Web-Interface Seite „Security“

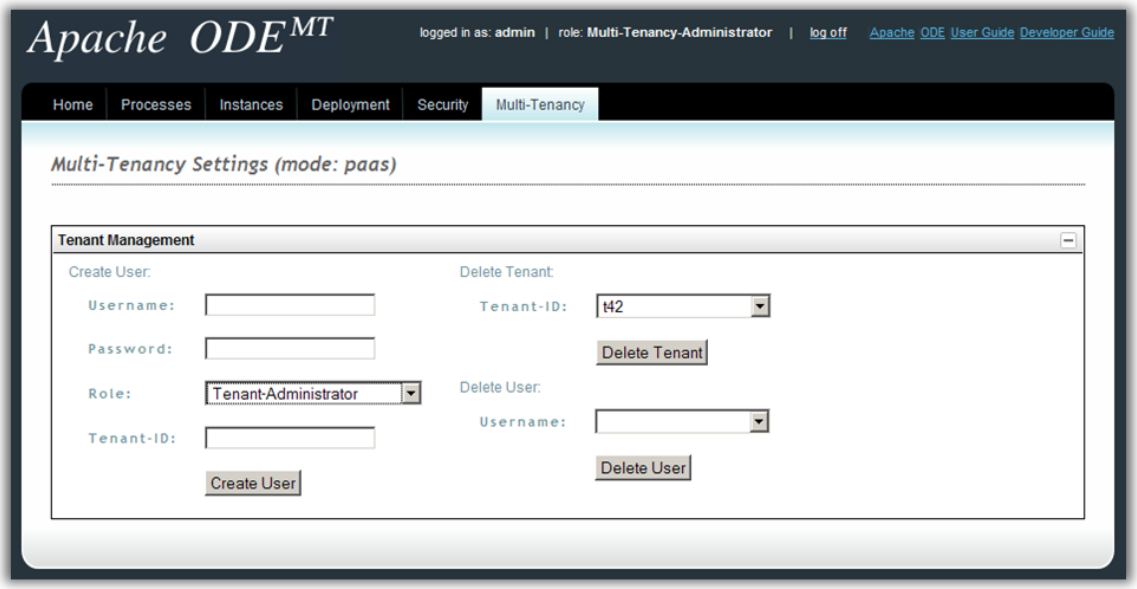


Abbildung C.6: Web-Interface Seite „Multi-Tenancy“ für Multi-Tenancy-Administrator

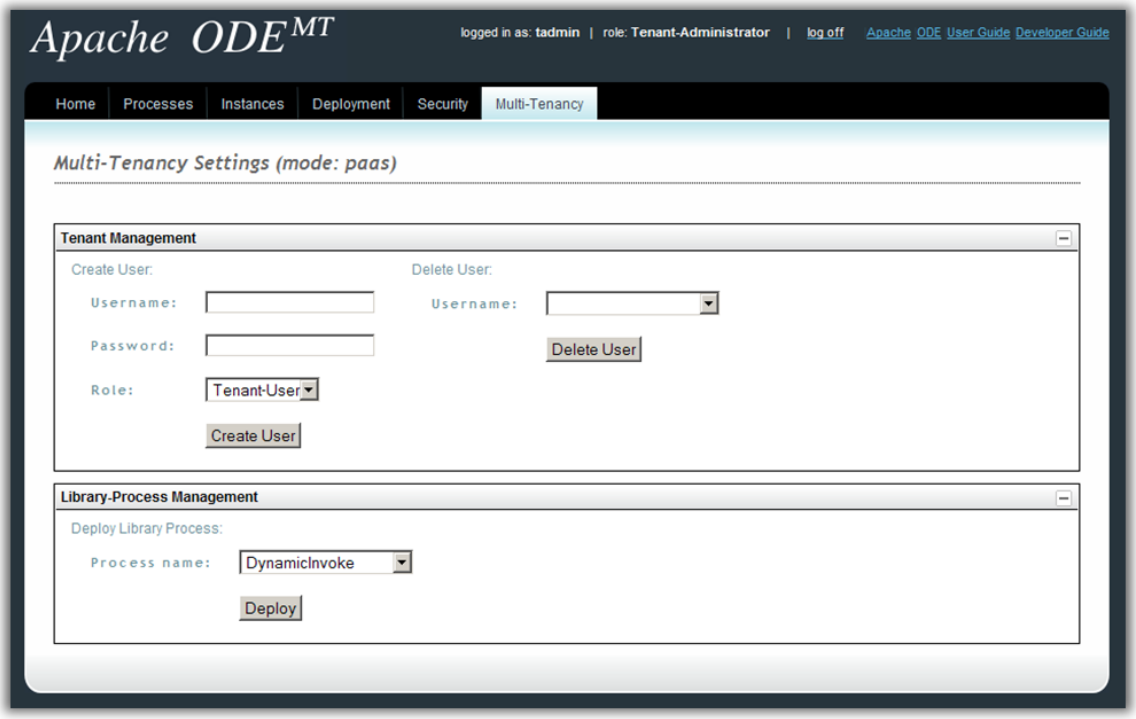


Abbildung C.7: Web-Interface Seite „Multi-Tenancy“ für Tenant-Administrator

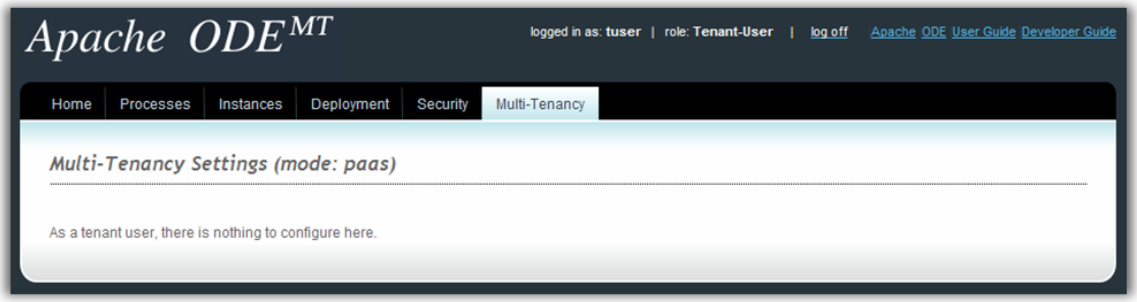


Abbildung C.8: Web-Interface Seite „Multi-Tenancy“ für Tenant-User

Abbildung C.9: Web-Interface Login-Seite

Literaturverzeichnis

[AAA+06] ALVES, Alexandre ; ARKIN, Assaf ; ASKARY, Sid ; BLOCH, Ben ; CURBERA, Francisco ; GOLAND, Yaron ; KARTHA, Neelakantan ; STERLING ; KÖNIG, Dieter ; MEHTA, Vinkesh ; THATTE, Satish ; RIJN, Danny van d. ; YENDLURI, Prasad ; YIU, Alex: *Web Services Business Process Execution Language Version 2.0*. OASIS Committee Draft, May 2006

(Zitiert auf Seite 6)

[ADC05] ABELS, Tim ; DHAWAN, Puneet ; CHANDRASEKARAN, Balasubramanian: An Overview of Xen Virtualization. (2005)

(Zitiert auf Seiten 12, 13, and 111)

[ADH+03] AALST, W. M. P. d. ; DONGEN, B. F. ; HERBST, J. ; MARUSTER, L. ; SCHIMM, G. ; WEIJTERS, A. J. M. M.: Workflow mining: a survey of issues and approaches. In: *Data Knowl. Eng.* 47 (2003), Nr. 2, S. 237–267. `http://dx.doi.org/http://dx.doi.org/10.1016/S0169-023X(03)00066-1`. – DOI http://dx.doi.org/10.1016/S0169–023X(03)00066–1. – ISSN 0169–023X

(Zitiert auf Seite 50)

[AFG+09] ARMBRUST, Michael ; FOX, Armando ; GRIFFITH, Rean ; JOSEPH, Anthony D. ; KATZ, Randy H. ; KONWINSKI, Andrew ; LEE, Gunho ; PATTERSON, David A. ; RABKIN, Ariel ; STOICA, Ion ; ZAHARIA, Matei: Above the Clouds: A Berkeley View of Cloud Computing / EECS Department, University of California, Berkeley. Version: Feb 2009. `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`. 2009 (UCB/EECS-2009-28). – Forschungsbericht

(Zitiert auf Seiten 45 and 109)

[AGL98] AGRAWAL, Rakesh ; GUNOPULOS, Dimitrios ; LEYMANN, Frank: Mining Process Models from Workflow Logs. In: *In Sixth International Conference on Extending Database Technology*, 1998, S. 469–483

(Zitiert auf Seite 50)

[ALMS09] ANSTETT, Tobias ; LEYMANN, Frank ; MIETZNER, Ralph ; STRAUCH, Steve: Towards BPEL in the Cloud: Exploiting Different Delivery Models for the Execution of Business Processes. In: *SERVICES '09: Proceedings of the 2009 Congress on Services - I*. Washington, DC, USA : IEEE Computer Society, 2009. – ISBN 978–0–7695–3708–5, 670–677

(Zitiert auf Seiten 8, 16, 51, 62, 65, and 111)

[And04] ANDERSON, Chris: *The Long Tail*. Version: October 2004. `http://www.wired.com/wired/archive/12.10/tail.html`, Abruf: 07. 12. 2009

(Zitiert auf Seite 15)

[AWM04] AALST, Wil van d. ; WEIJTERS, Ton ; MARUSTER, Laura: Workflow Mining: Discovering Process Models from Event Logs. In: *IEEE Transactions on Knowledge and Data Engineering* 16 (2004), Nr. 9, S. 1128–1142. `http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/TKDE.2004.47`. – DOI http://doi.ieeecomputersociety.org/10.1109/TKDE.2004.47. – ISSN 1041–4347

(Zitiert auf Seite 50)

[BBF+08] BARTEL, Mark ; BOYER, John ; FOX, Barb ; LAMACCHIA, Brian ; SIMON, Ed ; EASTLAKE, Donald (Hrsg.) ; REAGLE, Joseph (Hrsg.) ; SOLO, David (Hrsg.) ; HIRSCH, Frederick (Hrsg.) ; ROESSLER, Thomas (Hrsg.): *XML Signature Syntax and Processing (Second Edition)*. World Wide Web Consortium, Recommendation REC-xmldsig-core-20080610, June 2008

(Zitiert auf Seite 51)

[BDF+03] BARHAM, Paul ; DRAGOVIC, Boris ; FRASER, Keir ; HAND, Steven ; HARRIS, Tim ; HO, Alex ; NEUGEBAUER, Rolf ; PRATT, Ian ; WARFIELD, Andrew: Xen and the art of virtualization. In: *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles*. New York, NY, USA : ACM, 2003. – ISBN 1–58113–757–5, S. 164–177

(Zitiert auf Seite 12)

[Bit08] BITZER, Frank: *Management Framework for Amazon EC2*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Diplomarbeit, November 2008. `http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2841&engl=0`. – 77 S.

(Zitiert auf Seite 9)

[CFH+05] CLARK, Christopher ; FRASER, Keir ; HAND, Steven ; HANSEN, Jacob G. ; JUL, Eric ; LIMPACH, Christian ; PRATT, Ian ; WARFIELD, Andrew: Live migration of virtual machines. In: *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation*. Berkeley, CA, USA : USENIX Association, 2005, S. 273–286

(Zitiert auf Seite 12)

[Cle07] CLER, Charlie: *IBM System p: Improving Server Utilization*. Version: 2007. http://www-03.ibm.com/support/techdocs/atsmastr.nsf/WebIndex/WP101145

(Zitiert auf Seite 38)

[Erl07] ERL, Thomas: *SOA Principles of Service Design (The Prentice Hall Service-Oriented Computing Series from Thomas Erl)*. Upper Saddle River, NJ, USA : Prentice Hall PTR, 2007. – ISBN 0132344823

(Zitiert auf Seite 6)

[Gör09] GÖRIG, Heiko: *Kontextbezogene Zugriffskontrolle für BPEL (Engines)*, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, Diplomarbeit, April 2009. http://www.informatik.uni-stuttgart.de/cgi-bin/NCSTRL/NCSTRL_view.pl?id=DIP-2866&engl=0. – 129 S.

(Zitiert auf Seiten 26, 34, 75, 80, and 111)

[GSH+07] GUO, Chang J. ; SUN, Wei ; HUANG, Ying ; WANG, Zhi H. ; GAO, Bo: A Framework for Native Multi-Tenancy Application Development and Management. In: *E-Commerce Technology, IEEE International Conference on, and Enterprise Computing, E-Commerce, and E-Services, IEEE International Conference on* 0 (2007), S. 551–558. http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/CEC-EEE.2007.4. – DOI http://doi.ieeecomputersociety.org/10.1109/CEC–EEE.2007.4. ISBN 0–7695–2913–5

(Zitiert auf Seiten 43, 46, 49, 52, and 63)

[Gun08] GUNTEN, Andreas von: Plattformen als Service. In: *Infoweek.ch* n/a (2008), Nr. 2

(Zitiert auf Seite 62)

[Ham07] HAMILTON, James: On designing and deploying internet-scale services. In: *LISA'07: Proceedings of the 21st conference on Large Installation System Administration Conference*. Berkeley, CA, USA : USENIX Association, 2007. – ISBN 978–1–59327–152–7, S. 1–12

(Zitiert auf Seite 46)

[IDS02] IMAMURA, T. ; DILLAWAY, B. ; SIMON, E.: *XML Encryption Syntax and Processing, W3C Candidate Recommendation, Dec.˜2002*. http://www.w3.org/TR/xmlenc-core/, 2002

(Zitiert auf Seite 51)

[JA07] JACOBS, Dean ; AULBACH, Stefan: Ruminations on Multi-Tenant Databases. In: KEMPER, Alfons (Hrsg.) ; SCHÖNING, Harald (Hrsg.) ; ROSE, Thomas (Hrsg.) ; JARKE, Matthias (Hrsg.) ; SEIDL, Thomas (Hrsg.) ; QUIX, Christoph (Hrsg.) ; BROCHHAUS, Christoph (Hrsg.): *BTW* Bd. 103, GI, 2007 (LNI). – ISBN 978–3–88579–197–3, S. 514–521

(Zitiert auf Seiten 40, 41, and 109)

[Kal08] KALISKI, Burt: Multi-tenant Cloud Computing: From Cruise Liners to Container Ships. In: *Trusted Infrastructure Technologies Conference, Asia-Pacific /Trusted Infrastructure Technologies Conference, Third Asia-Pacific* 0 (2008), S. 4. `http://dx.doi.org/http://doi.ieeecomputersociety.org/10.1109/APTC.2008.16`. – DOI http://doi.ieeecomputersociety.org/10.1109/APTC.2008.16. ISBN 978–0–7695–3363–6

(Zitiert auf Seite 3)

[Mic06a] MICROSOFT CORPORATION: *Architecture Strategies for Catching the Long Tail*. Version: April 2006. `http://msdn.microsoft.com/en-us/library/aa479069.aspx`, Abruf: 25. 08. 2009

(Zitiert auf Seiten 14, 15, 16, 38, 41, 61, and 111)

[Mic06b] MICROSOFT CORPORATION: *Multi-Tenant Data Architecture*. Version: June 2006. `http://msdn.microsoft.com/en-us/library/aa479086.aspx`, Abruf: 04. 10. 2009

(Zitiert auf Seiten 43, 51, and 109)

[Mik08] MIKE HADLOW: *Multi-tenancy part 1: Strategy*. Version: November 2008. `http://mikehadlow.blogspot.com/2008/11/multi-tenancy-part-1-strategy.html`, Abruf: 06. 09. 2009

(Zitiert auf Seite 62)

[SOA09] SOA, Open: *Service Component Architecture Specifications*. Version: April 2009. `http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications`, Abruf: 20. 12. 2009

(Zitiert auf Seite 27)

[SS01] SMITH, S. W. ; SAFFORD, D.: Practical server privacy with secure coprocessors. In: *IBM Syst. J.* 40 (2001), Nr. 3, S. 683–695. – ISSN 0018–8670

(Zitiert auf Seite 51)

[Ste08] STERN, Allen: *Update From Amazon Regarding Friday's S3 Downtime*. Version: February 2008. http://www.centernetworks.com/amazon-s3-downtime-update, Abruf: 16. 12. 2009

(Zitiert auf Seite 52)

[SUN03] SUN MICROSYSTEMS: *Core J2EE Pattern Catalog - Data Access Objects*. Version: April 2003. http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html, Abruf: 17. 12. 2009

(Zitiert auf Seite 29)

[THW05] TEN-HOVE, Ron ; WALKER, Peter: *Java Business Integration (JBI) 1.0, Final Release*. Version: August 2005. http://www.jcp.org/en/jsr/detail?id=208, Abruf: 20. 12. 2009

(Zitiert auf Seite 27)

[WCL+05] WEERAWARANA, Sanjiva ; CURBERA, Francisco ; LEYMANN, Frank ; STOREY, Tony ; FERGUSON, Donald F.: *Web Services Platform Architecture : SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005 http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0131488740. – ISBN 0131488740

(Zitiert auf Seiten 6, 7, and 111)

[Xen10] XEN PROJECT: *Users' Manual Version 3.3*. Version: January 2010. http://bits.xensource.com/Xen/docs/user.pdf, Abruf: 04. 01. 2009

(Zitiert auf Seite 12)

Tabellenverzeichnis

2.1 Kosten für EC2 Instanz-Stunden in Irland 9

3.1 Speicherverbrauch von Schemainstanzen nach [JA07] 41
3.2 Erweiterung durch feste Anzahl zusätzlicher Spalten (vgl. [Mic06b]) 43
3.3 Erweiterung des Datenmodells mittels Schlüssel-/Wert-Paaren (vgl. [Mic06b]) 43
3.4 Service-Ausfälle prominenter Internet-Dienstleister [AFG+09] 45
3.5 Bewertungskriterien für Multi-Tenancy-Systeme 54
3.6 Übersicht über Eigenschaften der Grade von Multi-Tenancy 57

5.1 Modus-Konfiguration durch Umgebungsvariable 77
5.2 Funktionen des Web-Interfaces 83

A.1 Inhalt der Deployment-Unit „DynamicInvoke“ 87

Abbildungsverzeichnis

1.1 IT-Stack und Multi-Tenancy 4

2.1 Grundlegender Aufbau einer Service-orientierten Architektur (vgl. [WCL+05]) 7
2.2 Browser-Plugin „ElasticFox“ für Mozilla Firefox 10
2.3 Xen-Architektur (vgl. [ADC05]) 13
2.4 Graphische Veranschaulichung des sog. „Long Tail“ [Mic06a] 15
2.5 Einfluss von SaaS im Bezug auf den sog. „Long Tail“ [Mic06a] 16
2.6 Architektur von BPEL 23
2.7 Beispielprozess „DynamicInvoke“ 25
2.8 Architektur der Apache ODE (vgl. [Gör09]) 26
2.9 ODE-Statistik im Web-Interface 34

3.1 Wahl eines Reifegrades für Multi-Tenancy [Mic06a] 61
3.2 Wahl des Auslieferungsmodells [ALMS09] 62
3.3 IaaS [ALMS09] 65
3.4 PaaS [ALMS09] 65
3.5 SaaS [ALMS09] 65

C.1 Web-Interface Seite „Home“ 97
C.2 Web-Interface Seite „Processes“ 98
C.3 Web-Interface Seite „Instances“ 98
C.4 Web-Interface Seite „Deployment“ 99
C.5 Web-Interface Seite „Security“ 99
C.6 Web-Interface Seite „Multi-Tenancy“ für Multi-Tenancy-Administrator 100
C.7 Web-Interface Seite „Multi-Tenancy“ für Tenant-Administrator . . . 100
C.8 Web-Interface Seite „Multi-Tenancy“ für Tenant-User 101
C.9 Web-Interface Login-Seite 101

Verzeichnis der Listings

2.1 Definition einer Umgebungsvariablen im Deployment-Deskriptor 20
2.2 Zugriff auf Umgebungsvariable einer Web-Applikation 20
2.3 Absicherung von Ressourcen bei Apache Tomcat 21
2.4 Rollen und Benutzer beim MemoryRealm 21
2.5 Definition eines JDBCRealm 22
2.6 Datenbankschema für einen JDBCRealm 22
2.7 Beispiel-Angabe eines Namespaces 32

3.1 Zugriff auf tenant-spezifische Daten mithilfe eines Views 42

5.1 User-Tabelle mit Tenant-Identifier für JDBCRealm 76
5.2 Definition eines JDBCRealm innerhalb eines <Context> 76
5.3 WSDL-Anpassung für Tenant-ID „t44“ 79

A.1 Deployment-Deskriptor von DynamicInvokeProcess 88
A.2 DynamicInvokeProcess.bpel 88
A.3 DynamicInvokeProcessArtifacts.wsdl 91

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

(Andreas Buchholz)