SOA Testing

Tobias Bosch
OPITZ CONSULTING GmbH
München

Schlüsselworte

SOA, SOA Suite, OSB, Testing, Mock

Einleitung

SOA-Anwendungen sind verteilte Systeme und verwenden viele Schnittstellen, um zu kommunizieren: HTTP, JMS, FTP, Email, usw. SOA-Tests sind daher im Gegensatz zu klassischen Softwaretests ungleich komplexer. Sie müssen an beliebige Stellen des Verbindungsnetzes Anfragen abschicken und diese an anderen Stellen wieder abfangen können. Dabei definiert der Testfall, an welchen Stellen beides geschehen soll.

Der Vortrag stellt einen neuen Testansatz für SOA-Tests in der Oracle SOA-Suite unter Verwendung des Oracle Service Bus vor.

Was ist SOA?

Die Serviceorientierte Architektur stellt ein Architekturmodell dar. Sie hat das Ziel, die IT besser und flexibler an den Geschäftsprozessen eines Unternehmens auszurichten. Dadurch kann ein Unternehmen schneller auf Änderungen am Markt reagieren. Neben der Flexibilität sollen mit SOA die langfristigen IT-Kosten gesenkt werden. Eine SOA legt den Fokus der IT statt auf die Technik hin zu den Geschäftsprozessen und den beteiligten Daten, die über mehrere Applikationen und Abteilungen hinweg verwendet werden. SOA erreicht diese Ziele durch die Verwendung von sog. Services als das primäre Mittel, um Anwendungslogik zu realisieren. Services werden durch den sog. Service Provider zur Verfügung gestellt und vom sog. Service Konsumenten verwendet. Services können aus anderen Services zusammengesetzt werden (sog. Komposition). Prozesse können durch die Verwendung von Services automatisiert werden (sog. Orchestrierung) (siehe [1]).

Ein Enterprise Service Bus (ESB) wird in einer SOA eingesetzt, um Services zu entkoppeln und zu virtualisieren (siehe Abbildung 1). Er steht immer zwischen Service Provider und Service Consumer, und erlaubt Routing, Nachrichtentransformation und Protokolltransformation. Der Oracle Service Bus (OSB) ist ein ESB und eine Kernkomponente der Oracle SOA-Suite.

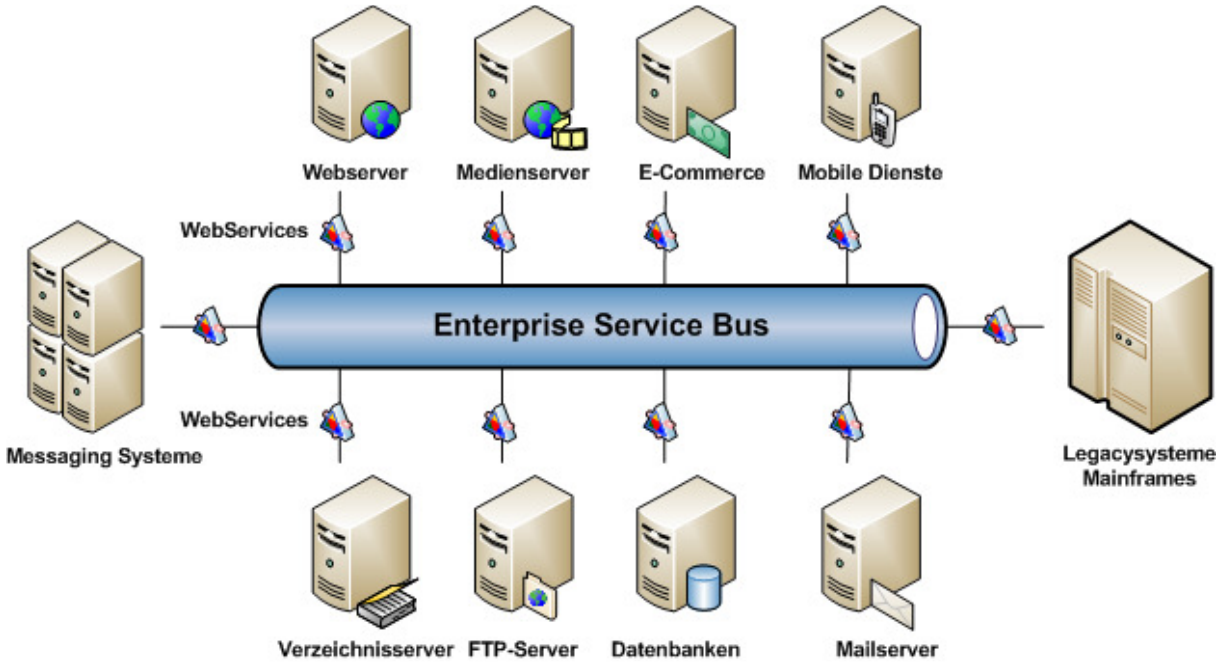

Abb. 1: Enterprise Service Bus

Warum Testen in einer SOA?

Softwaretests haben das Ziel, Fehler in einer Software aufzudecken. Sie prüfen, ob die Software sich unter bestimmten Bedingungen auf eine bestimmte Art und Weise verhält.

Je nachdem, was getestet wird, werden verschiedene Teststufen unterschieden. Der Komponenten- oder auch Unittest testet die korrekte Funktion der grundlegenden Bausteine einer Software. In einer objektorientierten Umgebung ist dies beispielsweise eine Klasse, in einem ServiceBus eine Nachrichten-Transformation oder ein Routing. Komponententests beziehen sich jeweils nur auf eine einzelne Komponente in Isolation. Falls eine Komponente andere Komponenten verwendet, müssen diese während des Tests durch Stellvertreter-Objekte ersetzt werden (sog. Mock-Objekte). Diese Stellvertreter simulieren ein bestimmtes Verhalten und speichern die an sie gesendeten Nachrichten für eine spätere Prüfung im Test.

Integrationstests sind die nächsthöhere Teststufe. Sie prüfen das korrekte Zusammenspiel von mehreren Komponenten. Sie gehen von der korrekten Funktionsweise der einzelnen Komponenten aus. Auch hier kann die Verwendung von Mock-Objekten Sinn machen, um die am Test beteiligten Komponenten einzugrenzen und so Fehler schneller lokalisieren zu können.

Auf oberster Ebene gibt es die Abnahme- oder auch Akzeptanztests. Sie werden durch den Kunden nach dem Abschluss der Entwicklung durchgeführt und prüfen das Verhalten des gesamten Systems gegen alle funktionalen und nicht-funktionalen Anforderungen.

Je früher ein Fehler während des Software-Entwicklungsprozesses gefunden wird, desto weniger kostet es, ihn zu beheben (siehe [2]). Daher lohnt sich der Einsatz von Tests bereits in der Entwicklungsphase. Insbesondere ist der Einsatz von automatisierten, regelmäßig durchgeführten Tests sinnvoll (sog. Continous Integration), da dadurch Nebenwirkungen von Modifikationen in bereits getesteten Teilen der Software schnell gefunden werden können.

Warum sind Tests für eine SOA so wichtig? Erstens ist SOA ist ein verteiltes System. Das heißt, sie besteht aus mehreren Systemen, die wiederum aus vielen verschiedenen Teilen bestehen. Ein Fehler in einer SOA kann daher viele verschiedene Ursachen haben. Softwaretests helfen, die genaue Fehlerursache zu lokalisieren. Zum zweiten ist es eine grundlegende Idee von SOA, die Software leicht anpassen zu können und so schnell auf Änderungen im Geschäftsmodell zu reagieren. Softwaretests helfen hier, um nach einer Änderung Nebenwirkungen auf andere Funktionen der SOA zu finden. Je mehr diese Tests automatisiert sind, desto schneller kann die Änderung produktiv verwendet werden. Zum dritten bildet eine SOA die zentralen Geschäftsprozesse eines Unternehmens ab. Das System muss eine hohe Qualität aufweisen, da Fehler zur Behinderung dieser Prozesse führen. Softwaretests erlauben die Sicherstellung dieser Qualität, da sie Fehler frühzeitig aufdecken und sicherstellen, dass Änderungen keine Auswirkungen auf bereits getestete Teile der SOA haben.

Unittests in der Oracle SOA-Suite

Eine SOA verbindet unterschiedliche Service-Anbieter und –Konsumenten durch Transformations- und Orchestrierungslogiken. Die Oracle SOA-Suite setzt für Transformationen auf XQuery und XSLT. Für die Orchestrierung wird BPEL und BPMN (in Composite Applications) sowie ein eigener XML-Dialekt (in OSB Anwendungen) eingesetzt.

Die Oracle SOA-Suite unterstützt keine automatischen Tests für XQuery oder XSLT. Ein möglicher Ausweg ist die direkte Verwendung der XQuery- und XSLT-Engines von Oracle in einem JUnit-Test [3]. Die Verwendung von JUnit erlaubt die einfache Integration der Tests in Build-Systeme, Continous Integration-Systeme und Entwicklungsumgebungen (z.B. Eclipse).

BPEL- und BPMN-Prozesse können mit Hilfe der SOA-Suite Tests für Composite Applications getestet werden. Diese legen Eingangsdaten für die durch das Composite implementierten Service-Interfaces fest. Referenzierte Services können durch Mocks ersetzt werden, die feste Nachrichten zurückgeben. Prüfungen können für alle eingehenden und ausgehenden Nachrichten festgelegt werden. Composite Tests müssen vor der Ausführung zusammen mit dem Composite auf den Server deployt werden. Die Ausführung kann per Ant automatisiert werden und so in Entwicklungsumgebungen und Continous Integration-Systemen verwendet werden.

Für das Testen der Orchestrierung von Services im OSB existiert keine Testautomatisierung von Oracle. Da es sich hier um ein Oracle spezifisches Format handelt, existieren auch keine Tools von Drittanbietern. In diesem Artikel wird an späterer Stelle ein neues Verfahren vorgestellt, mit dessen Hilfe Unittests inkl. der Unterstützung von Mock-Services im OSB möglich werden (siehe Abschnitt Automatische Tests mit Hilfe des OSBs).

Integrationstests in der Oracle SOA-Suite

Oracle unterstützt in den Tests für Composite Applications auch Integrationstests. Im Gegensatz zu Unittests (s.o.) werden dazu die ausgehenden Services nicht durch einen Mock ersetzt. Dieses Vorgehen ist für alle Composite Applications, unabhängig von den darin verwendeten Protokollen, möglich. Die Composite-Tests der SOA-Suite erlauben es jedoch nicht, Mock-Services in Composites außerhalb des durch den Test aufgerufenen Composites einzusetzen. Das ist aber notwendig, wenn ein SOA-Suite Projekt aus mehreren Composites besteht (z.B. ein Composite für einen BPEL-Prozess und

ein separates Composite für den Mediator) und die gesamte Verarbeitung in der SOA-Suite isoliert von angrenzenden Systemen getestet werden soll.

Neben den Tests für Composite Applications gibt es auch Tools von Drittherstellern wie SoapUI [4] oder das Citrus Test-Framework [5]. Diese rufen bereitgestellte Services auf und prüfen deren Antwortverhalten. Allerdings unterstützen sie zum Teil nur eine kleine Auswahl an Protokollen (SoapUI unterstützt z.B. nur JMS, HTTP und JDBC). Sie erlauben es jedoch nicht, Mock-Services abhängig vom Testfall in den zu testenden Service automatisch zu integrieren.

Nötig ist also ein Testansatz, der für alle Protokolle der Oracle SOA-Produkte verwendet werden kann, und der es erlaubt, vom Testfall abhängige Mocks dynamisch in den zu testenden Service zu integrieren. Im Folgenden wird ein solcher Testansatz vorgestellt (siehe Abschnitt Automatische Tests mit Hilfe des OSBs).

Automatische Tests mit Hilfe des OSBs

Wie in der Einleitung erwähnt, dient ein ESB dazu, die Services innerhalb einer SOA voneinander zu entkoppeln. Das bedeutet, dass jeder Serviceaufruf zunächst zum ESB gelangt, bevor der ESB den Aufruf dann zum konkreten Service weiterleitet. Damit ist ein ESB der ideale Ort für die Implementierung von SOA-Tests: Er kann jeden zu testenden Service aufrufen, und jeden Serviceaufruf abfangen, der von dem aufgerufenen Service durchgeführt wird. Nachfolgend wird dies anhand einer Implementierung für den OSB und JUnit beschrieben:

Der OSB erlaubt es, jeden beliebigen Service aufzurufen, wobei der aufzurufende Service durch eine XQuery-Expression angegeben werden kann (sog. Dynamic Routing). Damit ist es möglich, einen Meta-Service (im folgenden ServiceInvoker genannt) für den OSB zu implementieren, der als Eingabeparameter den Namen des aufzurufenden Service sowie die Eingabeparameter für diesen Service erhält. In der Implementierung kommuniziert der JUnit-Test über SOAP/HTTP mit dem ServiceInvoker im OSB. Somit ist es möglich, über JUnit jeden beliebigen Service in der SOA aufzurufen und dessen Ergebnis zu prüfen.

Um ausgehende Serviceaufrufe des OSBs abzufangen und durch eine Mock-Implementierung zu ersetzen, wird ein weiteres Feature des OSBs verwendet: Die dynamische Konfiguration via JMX. Dadurch ist es u.a. möglich, das zu verwendende Protokoll und die Zieladresse jeder beliebigen Servicereferenz festzulegen. In der Implementierung werden Mock-Services im JUnit-Test definiert und zunächst als HTTP-WebServices zur Verfügung gestellt. Anschließend werden die Servicereferenzen, die durch einen Mock ersetzt werden sollen, im OSB via JMX konfiguriert. Als Transportmedium wird HTTP gesetzt und als Zieladresse die Adresse des jeweiligen Mock-Services des JUnit-Tests angegeben. Ein solcher JUnit-Test kann damit jeden beliebigen Serviceaufruf des OSBs abfangen. Es ist dabei allerdings zu beachten, dass der JUnit-Test die Änderungen an der OSB-Konfiguration am Ende wieder rückgängig machen muss. Abbildung 2 fasst die Zusammenhänge zwischen JUnit-Test und OSB zusammen.

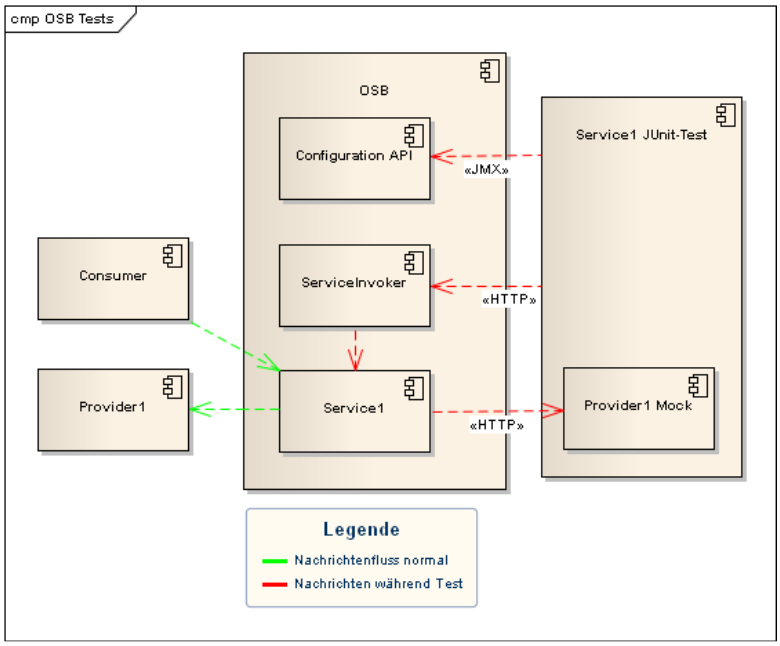

Abb. 2: SOA-Tests mit dem OSB

Durch geeignete Hilfsklassen kann der nötige Code im JUnit-Test sehr kurz gehalten werden. Die Implementierung der Mock-Services kann als anonyme Klassen innerhalb der Testmethode erfolgen, die den Service aufruft. Dadurch kann die Testmethode mit dem Mock direkt kommunizieren. Das folgende Listing stellt einen Beispiel-Test dar. Dieser ersetzt den Service `Provider1` (Zeile 5) durch eine Mock-Implementierung, die den empfangenen Request speichert und immer den Wert `<testresponse>hallo</testresponse>"` zurückgibt (Zeile 6-9). Anschließend ruft der Test den Service `Service1` mit dem Request `<testrequest>hallo</testrequest>` auf (Zeile 12-14). Schließlich prüft der Test, ob der Mock den richtigen Request empfangen hat (Zeile 15) und das Ergebnis des `Service1`-Serviceaufrufs dem Ergebnis des Mocks entspricht (Zeile 16).

```
1  public class Service1Test extends OsbTestBase {
2    private String receivedReq;
3    @Test
4    public void testService1() {
5      mockBusinessService("App/Provider1", new Delegate() {
6        public String serviceCallReceived(String serviceName,
7                                String requestStr) throws Exception {
8          receivedRequest = requestStr;
9          return "<testresponse>hallo</testresponse>";
10       }
11     });
12     String receivedResp = invokeProxyService(
13                "App/Service1",null,
14                "<testrequest>hallo</testrequest>");
15     assertEquals("<testrequest>hallo</testrequest>",receivedReq);
16     assertEquals("<testresponse>hallo</testresponse>",receivedResp);
17   }
18 }
```

Zusammenfassung

Gerade in einer SOA ist die Durchführung von Softwaretests unerlässlich. Eine SOA besteht aus vielen verschiedenen Teilen. Softwaretests erleichtern die Erkennung und Lokalisierung von Fehlern. Die regelmäßige Ausführung von Tests in einer SOA erlaubt weiterhin, unbeabsichtigte Nebenwirkungen einer Änderung auf andere Funktionen der SOA schnell zu finden. Damit ist die Vision einer SOA realisierbar: Schnelle Anpassung der Software an neue Anforderungen unter Beibehaltung einer für kritische Geschäftsprozesse geeigneten hohen Qualität.

Es gibt verschiedene Oracle-unabhängige Tools für Tests in einer SOA. Diese unterstützen zum Teil nicht alle Protokolle und unterstützen auch keine automatisch konfigurierten Mock-Services. Oracle bietet als Testtools in einer SOA lediglich die Tests für Composites an. Diese können zurzeit jedoch nicht mit dem OSB verwendet werden. Außerdem ist die Definition von Mock-Services auf das zu testende Composite beschränkt.

Dieser Artikel beschreibt einen Testansatz unter Verwendung des OSBs und JUnit, der den Aufruf von beliebigen Services und das Abfangen von beliebigen Serviceaufrufen an beliebigen Stellen in einer SOA ermöglicht. Damit werden jegliche Arten von Unit- und Integrationstests auf eine einheitliche Art und Weise möglich.

Quellen

[1] Thomas Erl (2009), SOA Design Patterns, Prentice Hall.

[2] Kaner, Cem; James Bach, Bret Pettichord (2001). Lessons Learned in Software Testing: A Context-Driven Approach. Wiley.

[3] XMLUnit, http://xmlunit.sourceforge.net/

[4] SoapUI, http://www.soapui.org/

[5] Citrus Test-Framework, http://www.citrusframework.org/

Kontaktadresse

Tobias Bosch
OPITZ CONSULTING München GmbH
Weltenburger Straße 4
D-81677 München

Telefon:	+49(0)89-680098-1456
Fax:	+49(0)89-680098-4400
E-Mail	tobias.bosch@opitz-consulting.com
Internet:	www.opitz-consulting.com