

# Konzept und Support für das Testen von Services

Jorge Marx Gómez, Universität Oldenburg und Daniel Lübke, Leibniz Universität Hannover



Jorge Marx Gómez ist Professor für Wirtschaftsinformatik an der Universität Oldenburg.



Daniel Lübke ist wissenschaftlicher Mitarbeiter im Fachgebiet Software Engineering an der Universität Hannover.

Service-orientierte Architekturen (SOA) sind ein aufstrebender Architekturstil für die Organisation von Geschäftsanwendungen und gesamten Anwendungslandschaften. Hierbei wird die Geschäftslogik als ein Satz von verteilten Softwarekomponenten, den sogenannten Services, angeboten. Solche Services können über standardisierte Schnittstellen angesprochen werden. Somit ist gewährleistet, dass sie sich flexibel und aus anderen Softwareanwendungen ansprechen lassen. Die am häufigsten verwendeten Standards für die Bereitstellung und den Aufruf von Services sind Webservices [10].

**In diesem Beitrag lesen Sie:**

- welche Bedeutung das Testen von Softwareanwendungen hat,
- welche Auswirkungen SOA-Anwendungen auf das Testen haben können,
- wie sinnvoll bei einer SOA eine Testautomatisierung ist.

Allgemein können Services sowohl von bestehenden Anwendungen angeboten werden, als auch von externen Dienstleistern. So bieten z.B. Google und eBay viele ihrer Dienste als Webservices an.

Das Haupteinsatzgebiet für SOA sind geschäftliche Softwareanwendungen. Diese Anwendungen sind oftmals geschäftskritisch. Ausfälle oder Fehlfunktionen können dementsprechend verheerende Folgen für das Unternehmen haben. Dies führt dazu, dass das Testen von diesen Systemen wichtiger Bestandteil des Entwicklungsprozesses sein muss. Allerdings wird das Testen von Anwendungen, die nach SOA-Prinzipien strukturiert sind, durch folgende Faktoren komplizierter als das Testen von „normalen“ Softwaresystemen:

- Services sind verteilte Softwarekomponenten, d.h. Tests müssen weitere Fehlerfälle im Falle von nicht verfügbaren Services und Netzwerkverbindungen beinhalten. Zudem sind entfernte Aufrufe deutlich langsamer als lokale Aufrufe, so dass gerade automatisierte Testsuiten länger für die Ausführung benötigen.
- Services können von Externen zur Verfügung gestellt werden, d.h.

Services können nicht einfach für das Testen beeinflusst werden, es können manchmal keine Testdaten verwendet werden und die Anbindung ist langsamer als mit internen Services.

- Services sind evtl. kostenpflichtig, d.h. jeder Serviceaufruf aus einem Test muss bezahlt werden. Daher sind bei solchen Services die Anzahl der Aufrufe zu minimieren.
- SOA möchte flexibel sein, d.h. Software wird oft geändert und muss dementsprechend oft getestet werden. Hierbei rentieren sich automatisierte Testsuiten schneller, aber Verantwortliche müssen sich mehr Gedanken um die Ausführungsfrequenz von nicht-automatisierbaren Tests machen.

Diese Herausforderungen erzwingen die Übernahme und Anpassung von klassischen Softwareteststrategien auf SOA-Anwendungen.

## Verwendung verschiedener Teststufen

In Softwareprojekten wird ausgehend von den (Benutzer-)Anforderungen die Software entworfen, in Einheiten zer-

Tabelle 1: Teststufen für verschiedene Entwicklungsprodukte

	Servicekomposition	Service (z. B. Java/.NET)
<b>Unit-Test</b>	Komposition, in der alle Services durch Mocks ersetzt sind	Objekte, bei denen alle Abhängigkeiten durch Mocks ersetzt sind
<b>Integrationstest</b>	Komposition, in der sukzessive die Services aktiviert werden	Objekte, bei denen sukzessive die Abhängigkeiten aktiviert werden
<b>Systemtest</b>	Komposition mit allen Services und vollständiger Konfiguration/Deployment	Service, mit allen Produktivklassen und Konfiguration/Deployment

teilt und implementiert. Analog dazu kann auf jeder dieser Ebenen getestet werden. Jede implementierte Einheit kann einzeln in sogenannten Unit Tests getestet werden. Hierbei werden alle anderen Einheiten ausgeblendet und gegebenenfalls durch einfache Ersatzimplementierungen (sogenannten Mocks) ersetzt. Unit Tests sollen Fehler in der Funktionsweise der einzelnen Einheiten finden.

Sind alle einzelnen Module getestet, so kann man diese miteinander kombinieren und zusammen testen. Diese Teststufe heißt Integrationstest. Sie dient dazu, Fehler in der Zusammenarbeit verschiedener Softwareeinheiten zu finden [1].

Wenn alle Einheiten zusammengestellt sind, liegt das ganze System vor und kann getestet werden. Diese Teststufe heißt Systemtest und dient dazu Fehler bezüglich der Benutzeranforderungen im endgültigen System zu finden.

In SOA-Projekten gibt es verschiedene Entwicklungsartefakte, die beim Testen zu berücksichtigen sind. Hauptsächlich sind dies Servicekompositionen und Serviceimplementierungen. Hierbei sind die verschiedenen Einheitstypen, wie in Tabelle 1 dargestellt, zu beachten.

### Aufstellen einer Teststrategie

Die Kenntnis um die verschiedenen Teststufen alleine ist aber nicht für das Testen ausreichend. Es stellt sich zwangsläufig die Frage, welche Aspekte auf welcher Stufe getestet werden sollen und welche Tests auf welcher Stufe operieren sollen. Hierbei muss zu aller erst festgelegt werden, welche Aspekte für die Firma als Ganzes und für das Projekt im Speziellen relevant sind. Mögliche Aspekte sind hierbei u.a.:

- Tests der Funktionalität einzelner Services
- Tests der Datentransferobjekte
- Tests des Deployments
- Tests der Autorisierung und Authentifizierung
- Tests bei Fehlen von Services
- Test auf Interoperabilität mit früheren Versionen
- Tests der Funktionalität des Systems

Je nach Umfeld können neue Aspekte hinzukommen oder andere Aspekte gewählt werden. Wichtig ist hierbei, dass Zuständigkeiten festgelegt werden (wer testet was und wann?) und die Art der Teststufe für den jeweiligen Test vorgegeben wird (z.B. Datentransferobjekte werden in Komponententests getestet).

### Toolsupport

Wenn die SOA-Landschaft oft umgestaltet werden soll, z.B. um eine neue Version eines Services anzubieten oder um Geschäftsprozesse anzupassen, ist es nötig das Testen zu optimieren um möglichst schnell die neue Version der Software in Betrieb nehmen zu können. Eine Option dazu ist die Automatisierung von Tests, was oftmals mit funktionalen Tests für einzelne Dienste aber auch für Servicekompositionen praktikabel möglich ist. Automatisierte Tests können schnell immer wieder ausgeführt werden. Werden Tests oft wiederholt zahlt sich der zusätzliche Aufwand für die Automatisierung aus. Hierzu werden Werkzeuge benötigt, die die eigene Teststrategie unterstützen und auf die Besonderheiten von Service-orientierten Architekturen Rücksicht nehmen. Dabei ist es vor allem wichtig, einzelne Services durch Serviceimitationen auszutauschen, um danach Fehlersituationen simulieren zu können. Ersteres erlaubt die Isolation von Komponenten für Kompo-

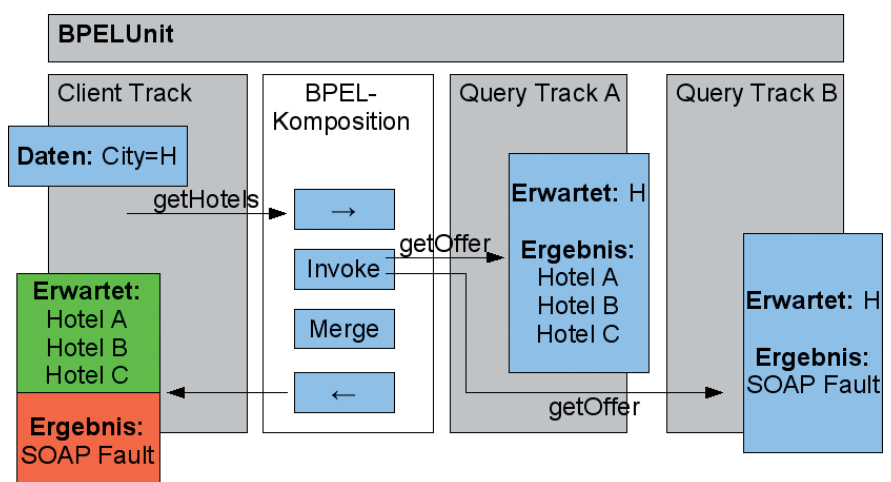
nententests und letzteres erlaubt das Testen der Fehlerbehandlungsroutinen für Servicekompositionen.

Ein solches Werkzeug ist BPELUnit [2], welches das Erstellen und automatisierte Ausführen von Tests für BPEL-Kompositionen aber auch für einzelne Webservices erlaubt. Hierbei erlaubt BPELUnit die Definition Eingabe- und Solldaten für Webservices. Für BPEL-Kompositionen kann BPELUnit zusätzlich das Deployment steuern als auch benutzte Webservices durch Mocks ersetzen. Diese Mocks können sowohl empfangene Daten überprüfen als auch Fehler in Form von SOAP-Faults an die Komposition senden.

BPELUnit ist als Open-Source-Projekt unter <http://bpelunit.org> verfügbar. Es besteht aus einem Core, der über verschiedene Runner angesprochen werden kann. Somit kann BPELUnit in diverse Entwicklungsumgebungen integriert werden. Mitgeliefert werden Runner für Eclipse [9] (und darauf basierende Entwicklungsumgebungen wie ActiveBPEL Designer), ANT (für automatisierte Builds) und ein Kommandozeilen-Client.

Mit solchen Tools ist es möglich BPEL-Kompositionen in ihrer Gesamtheit als auch isoliert zu testen. Ein Beispiel in Anlehnung an [2] ist in Bild 1 gezeigt: Es zeigt eine BPEL-Komposition, die verschiedene Informationsservices abfragt und die Ergebnisse zusammenführt. Für den Aufrufer der

Bild 1: Testszenario für eine BPEL-Komposition



BPEL-Komposition gibt es den „Client-Track“. Dieser schickt nun Testdaten zu der BPEL-Komposition, die zwei Webservices aufruft, um Hotels zu ermitteln. Die Webservices sind in diesem Fall durch Mocks ersetzt, so dass keine externen Webservices aufgerufen werden müssen. Diese Mocks haben ihre eigenen Tracks. Wird ein Mock aufgerufen, so werden die Eingabedaten überprüft. In diesem Fall muss die Stadt, für die Hotels gesucht werden soll, dieselbe Stadt sein, mit der die Komposition aufgerufen wurde. Danach sendet der Mock Daten zurück. Der erste Mock liefert dabei drei beliebige Hotels, während der Zweite einen Fehler simuliert. Erwartet wird in einem solchen Szenario, dass die BPEL-Komposition die Hotels des ersten Webservices zurückliefert. In diesem Fall wurde allerdings die Fehlerbehandlung nicht korrekt implementiert, so dass die BPEL-Komposition selber einen Fehler zurück liefert.

Auf diese Art und Weise besteht mit BPELUnit die Möglichkeit einen Komponententest zu schreiben, der die BPEL-Komposition komplett isoliert und alle Abhängigkeiten mittels der Werkzeugunterstützung eliminiert. So können auch erst später im Projekt die Informationsservices implementiert oder von externen Anbietern integriert werden.

## Testen von Föderierten ERP-Systemen

Ein Föderiertes ERP-System (FERP-System) ist ein ERP-System dessen betriebliche Anwendungsfunktionen variabel, dynamisch oder adaptiv zu unterschiedlichen voneinander unabhängigen Softwareanbietern zugeordnet werden können [5]. Die Gesamtfunktionalität des Systems wird von einem Ensemble standardisierter Teilsysteme bereitgestellt, das für den Endanwender als ganzheitliches Gesamtsystem (ERP-System) erscheint [6]. Das FERP Exchange System (FERP ES) baut hierbei auf den FERP Prototypen FERP X ONE auf [7]. Dieser Prototyp wurde in einer Projektgruppe „Föderierte ERP-

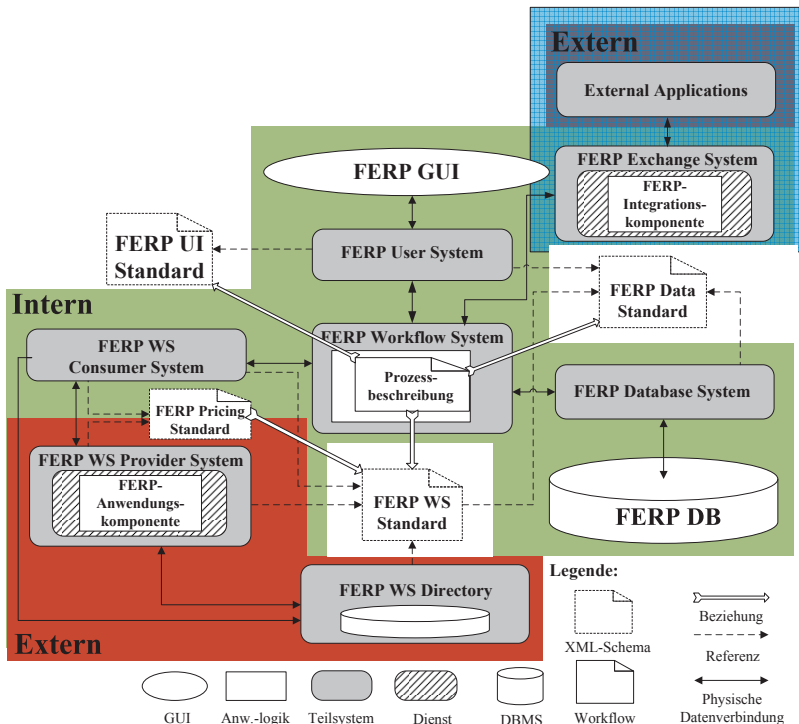


Bild 2: FERP Referenzarchitektur mit FERP ES und Testumgebung

Systeme auf der Basis von Web Services und Peer-to-Peer-Systemen“ entwickelt [8]. Das FERP ES dient als Integrationsschnittstelle für die Integration von externen Anwendungen und Föderierten ERP-Systemen. Darüberhinaus ermöglicht FERP ES mit Hilfe explizit für die gewünschten Tests modellierten Workflows, beliebige Daten durch Web Services als Schnittstelle auf dem FERP-System zu extrahieren und durch eine gewünschte Anwendung beziehungsweise Logik am anderen Ende zu verarbeiten. Als Beispiel könnte die Abfrage aller Kunden-Stammdaten genommen werden. Das Testprogramm lässt sich diese Daten über das FERP ES ermitteln und kann dann beispielsweise eine automatische Überprüfung auf Vollständigkeit durchführen. Als Ergebnis lassen sich dann alle Kundennummern der Stammdatensätze protokollieren, welche den Testkriterien nicht entsprechen um so eine manuelle Korrektur vorzunehmen.

Bild 2 zeigt die allgemeine Web Service basierende FERP-Referenzarchitektur mit dem erweiterten FERP Exchange System. Der interne Bereich

stellt eine Basisinstallation bei einem Anwenderunternehmen dar, bestehend aus grafischer Benutzeroberfläche, Datenbankmanagementsystem, Workflowmanagementsystem und einem System zum Aufruf von Webservices. Der untere externe Bereich kennzeichnet externe Teilsysteme zum Angebot von Webservices und zur Organisation (Publizieren und Suchen) von Webservice-Angeboten. Der obere externe Bereich zeigt hierbei den Bereich der Testplattform, auf den sich der Fokus konzentriert. Für Tests bezüglich der Korrektheit von Daten im FERP-System werden keine Kenntnisse der Teile des FERP User Systems oder des FERP Consumer Systems benötigt, da das FERP ES die einzige Schnittstelle von Außen nach Innen darstellt und ein direkter Zugriff auf diese Systeme nur vom FERP-System selbst durch Workflows vorgenommen werden kann. Statt einer externen Business-Anwendung wird daher lediglich eine entsprechende Anwendung mit zugehöriger Testlogik benötigt.

Nachfolgend eine grobe Übersicht eines möglichen Testablaufes, wobei

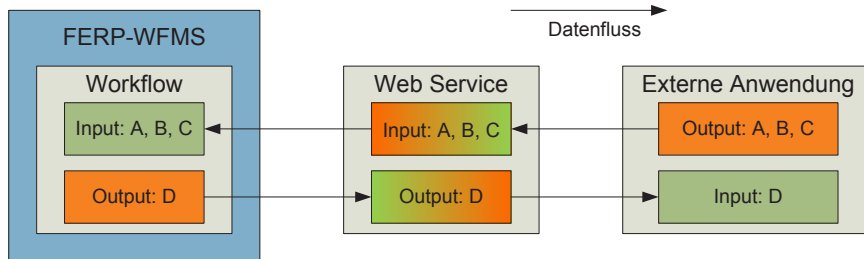


Bild 3: Ablauf des Datenflusses

hier vorausgesetzt wird, dass die nötigen Workflows bereits vorhanden sind. In Bild 3 ist weiterhin der Datenfluss zur Veranschaulichung dargestellt:

1. Die externe Anwendung, in diesem Fall die Testlogik, spricht einen Web Service für die entsprechenden Tests an und übergibt ihm die Datenanfrage.
2. Der Web Service leitet diese Anfrage an das FERP Exchange System weiter
3. Das FERP Exchange System ruft den oder die entsprechenden Workflows auf und übergibt diesem die Abfragedaten.
4. Das Workflow Management System verarbeitet den Workflow und gibt das Ergebnis an das FERP Exchange System zurück.
5. Das FERP Exchange System leitet die Ergebnisse des Workflows an den Web Service weiter.
6. Der Web Service übergibt diese dem Testprogramm, welches diese Daten nun analysieren kann.

Technische Änderungen am FERP ES oder dem FERP X ONE sind für die Umsetzung einer Testumgebung bei erster Betrachtung nicht notwendig. Lediglich die Testlogik in Form einer externen Anwendung mit Web Service Schnittstelle muss hierfür noch entwickelt werden. Diese muss in der Lage sein, Benutzeraktionen zu emulieren, um auch semi-automatische Workflows testen zu können. Hier bietet sich eine Plattform an, welche durch eine Art Plug-In-Lösung durch verschiedene Testscenarien erweitert werden kann. Weiterhin müssen

für diese Tests notwendige Workflows modelliert werden. Da zum Beispiel ein Datenbank-Ladevorgang immer gleich ist, müsste hierfür lediglich, auf den ersten Blick zumindest, nur ein einzelner Workflow erstellt werden. Dieser kann beispielsweise Kundendaten, Bestelldaten oder Abrechnungsdaten auslesen, je nach dem, welcher Input ihm geboten wird.

#### Literatur

- [1] Hardt, D.: Testen von serviceorientierten Architekturen: Teststrategie und Implementierung, Masterarbeit, Leibniz Universität Hannover, 2007, [http://www.se.uni-hannover.de/documents/studthesis/MSc/Dennis\\_Hardt-Testen\\_von\\_serviceorientierten\\_Architekturen-Teststrategie\\_und\\_Implementierung.pdf](http://www.se.uni-hannover.de/documents/studthesis/MSc/Dennis_Hardt-Testen_von_serviceorientierten_Architekturen-Teststrategie_und_Implementierung.pdf).
- [2] Lübke, D.: Unit Testing BPEL Compositions, in Test and Analysis of Web Services (Luciano Baresi und Elisabetta di Nitto), Springer Verlag, 2007.
- [3] Mayer, P.: BPELUnit, <http://bpelunit.org>, 2007.
- [4] Gronau, N.: Enterprise Resource Planning und Supply Chain Management, Architektur und Funktionen, München, 2004.
- [5] Marx Gómez, J., Rautenstrauch, C.: Architektur einer mittelstandsfähigen ERP-Lösung auf Basis von Web-Services und Peer-to-Peer-Netzen: ERP Management - Zeitschrift für unternehmensweite Anwendungssysteme, 1/2005, pp. 38-40.
- [6] Brehm, N., Marx Gómez, J., Rautenstrauch, C.: An ERP-solution based on Web-Services and Peer-to-Peer-Networks for SMEs: International Journal of Information Systems and Change Management (IJISCM), Volume 1 - Issue 1 - 2006, pp. 99-111.
- [7] FERP, Projektgruppe: Projektbericht Föderierte ERP-Systeme auf der Basis von Web Services und P2P-Systemen. (2007), 04.

- [8] Mesick, T.: Integration von externen Anwendungssystemen und Föderierten ERP-Systemen - Individuelles Projekt an der Abteilung Wirtschaftsinformatik I der Universität Oldenburg. (2007) 1.
- [9] Eclipse Entwicklungsumgebung: [www.eclipse.org](http://www.eclipse.org), letzter Zugriff am 09.01.2008
- [10] W. Dostal, M. Jeckle, I. Melzer und B. Zengler: Service-orientierte Architekturen mit Web Services, Elsevier Spektrum Akademischer Verlag, 2005.

#### Schlüsselwörter

Service-orientierte Architekturen, Servicekompositionen, Webservices, Teststrategie, Unit-Test, Integrationstest, Systemtest, Mocks, BPEL-Kompositionen, Föderiertes ERP-System

#### Challenges for Testing services

Service-oriented Architecture (SOA) strives to better support business processes. Changes in the business processes can be quickly incorporated into the service compositions. However, the compositions as well as the services themselves have to be tested in order to spot errors in their behaviour. Therefore, methods and tools that were used for testing software have to be adapted to the SOA field. This article presents the challenges faced when testing SOA applications and presents the BPELUnit framework for automating tests. Finally, a concept for testing FERP systems is given as an example.

#### Keywords:

Test, SOA, BPELUnit, FERP

#### Kontakt

Prof. Dr.-Ing. Jorge Marx Gómez  
 Universität Oldenburg  
 Department für Informatik  
 Abt. Wirtschaftsinformatik I  
 Ammerländer Heerstr. 114-118  
 26129 Oldenburg  
 Tel. 0441/ 798-4470  
[jorge.marx.gomez@uni-oldenburg.de](mailto:jorge.marx.gomez@uni-oldenburg.de)