Wolfram Kusterer

(E-Mail: wolfram.kusterer@parasoft.com)
ist als Geschäftsführer der Parasoft Deutschland GmbH verantwortlich für den Vertrieb von Software und Dienstleistungen in Deutschland, Österreich und der Schweiz. Er ist seit 20 Jahren in der IT und davon mehr als 10 Jahre in verschiedenen Funktionen im Consulting tätig.

End-to-End Test und Validierung

Qualitätssicherung in einem integrierten Geschäftssystem – ein praxisorientiertes Beispiel

Das Testen und Validieren von mehrschichtigen integrierten Systemen ist anspruchsvoll. Solche Systeme sind komplex, die Ursache ihrer Fehler genau zu bestimmen, ist schwierig, und eine Testumgebung aufzubauen, mit der man das Gesamtsystem angemessen testen kann, ist meist langwierig und kostspielig. Dieser Artikel behandelt verschiedene Test- und Validierungstechniken, mit denen sich Probleme in einem beispielhaften Geschäftssystem, das aus verschiedenen Komponenten besteht, identifizieren und diagnostizieren lassen. Die Systemarchitektur und die Testanforderungen in dem Beispiel basieren auf Erfahrungen, die Parasoft während der Arbeit mit einer breiten Kundenbasis in verschiedenen Branchen gesammelt hat. Wie einfach die empfohlenen Techniken automatisiert werden können, zeigen wir anhand einer beispielhaften Umsetzung mit Parasoft SOAtest. Die vorgestellten Praktiken können sowohl die Fehlerdiagnose maßgeblich beschleunigen als auch die Zeit und die Ressourcen reduzieren, die zur Erzeugung von wiederverwendbaren und reproduzierbaren Tests erforderlich sind.

Vorstellung des Szenarios

Nehmen wir einmal an, dass unsere Organisation für das „Parabank“ Online-Banking-System verantwortlich ist, das allen Kunden erlaubt, neue Konten zu eröffnen und Geld zwischen den Konten zu transferieren. Das System kann über das Internet mit Browsern genutzt werden sowie mit anderen Systemen, die eine Verbindung mit ATM-Terminals, mobilen Anwendungen oder anderen Quellen besitzen. Wir nennen dieses System im folgenden OBA (Online Banking Application).

Systemarchitektur

OBA ist auf einem JBoss Application Server gehostet und kommuniziert über TIBCO EMS mit einem Kontosystem. Der TIBCO Bus vermittelt die Kundentransaktionen zwischen der Online-Banking-Anwendung und dem Mainframe Backend, wo die Kontodaten verwaltet werden. Die Transaktionen beinhalten Überweisungen, Einzahlungen, Abhebungen, etc.. Die Broker Middleware stellt Performance und Verlässlichkeit zwischen den zwei Schichten bereit, sie ermöglicht außerdem die ESB-Transaktionen.

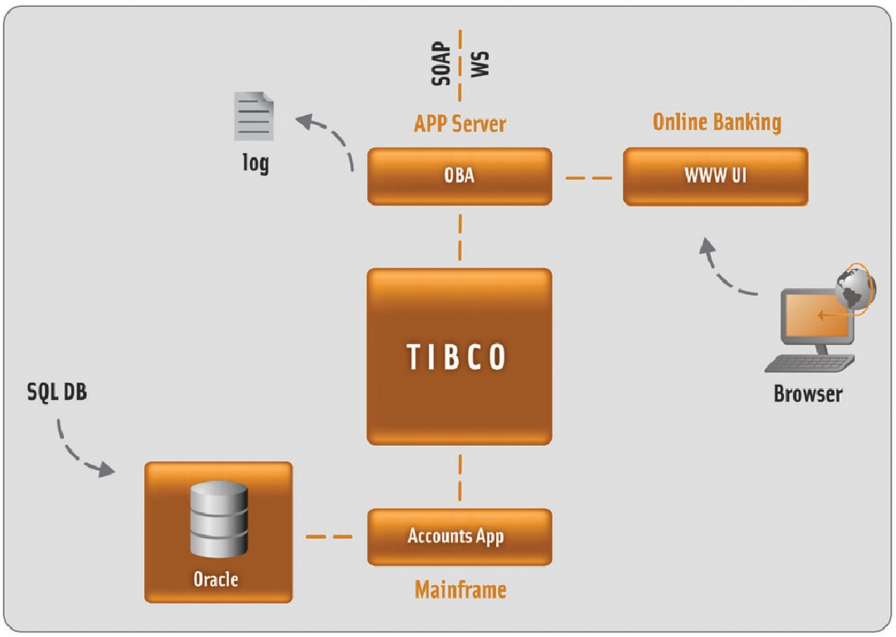

Abbildung 1: Systemarchitektur

Das Mainframe Backend verbindet sich mit einer Oracle-Datenbank, welche die Kundenkontendaten enthält. OBA umfasst

außerdem einen Webservice mit SOAP Interface, das von anderen externen Anwendungen (z. B. das ATM Netzwerk der Bank, eine iPhone Anwendung, Terminals der Zweigstellen, etc.) genutzt wird.

Solche mehrschichtigen integrierten Systeme sind heutzutage in Unternehmen sehr üblich und sie sind typischerweise weitaus komplexer als in diesem Beispiel. Dennoch sollte dieses Beispiel die Schwierigkeiten verdeutlichen, die beim Aufbau einer Testumgebung auftreten, welche es ermöglicht, verschiedene Komponenten anhand von realistischen Use Case Szenarien zu testen und zu validieren. Und obwohl in diesem Beispiel TIBCO eingesetzt wird, gelten dieselben Konzepte und Lösungen, die wir in diesem Artikel untersuchen, für WebMethods, WebSphere, Sonic oder jede andere nachrichtenorientierte Middleware oder ESB-Plattform.

Use Case Szenario

Nun wollen wir das folgende beispielhafte Use Case Szenario für OBA untersuchen:

- Einzahlung von $ 500 über die OBA Web Service SOAP Schnittstelle (unter der Annahme, dies wurde von einem ATM oder einer anderen externen Anwendung initiiert).
- Login von einem Web-Browser zur Prüfung des aktuellen Kontostands.
- Logout.

Vom Testen her betrachtet, führt solch ein einfaches Use Case Szenario oft zu weiteren aufwendigen Testanforderungen mit vielen Schritten wie z. B.:

- Ausführung eines Reset-Scripts, um die Datenbank in den Ausgangszustand zu setzen.
- Abgleich der Kontoinhalte unter Nutzung einer Datenbankschnittstelle.
- Nutzung eines browserbasierten HTTP POST Tools:
- Einfügen des SOAP XML Template für den Transfer.
- Editieren des Templates, um den gewünschten Betrag ($ 500), einzutragen.
- Abschicken.
- Login von einem Web-Browser zur Prüfung des neuen Kontostands.
- Logout.
- Wenn die Einzahlung fehlschlägt:
- Überprüfung der Server Log Files der Anwendung:
- FTP Fenster auf dem OBA Server öffnen.

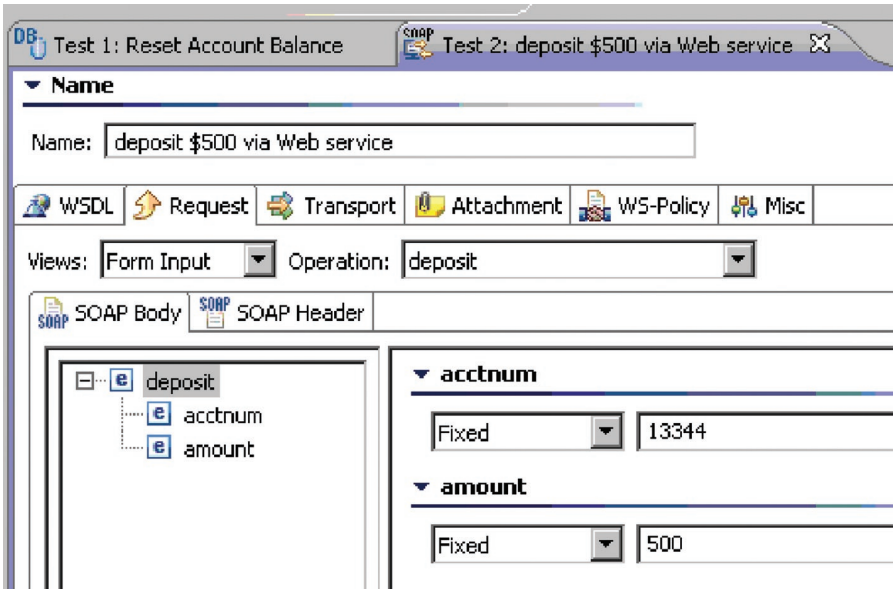

Abbildung 2: Konfiguration der Einzahlung

- Die Activity Logdatei abrufen.
- Den neuen Transaktionseintrag überprüfen.
- Überprüfung des Einzahlungsauftrags auf dem Bus:
- Login auf dem ESB Server.
- Die Mitteilung im Message Browser finden.

Anforderungen

Diese zusätzlichen Schritte können den QS-Ingenieur erheblich belasten, denn

- es sind diverse Tools und Methoden erforderlich, um die Funktionalität des Systems an den verschiedenen Stellen zu überprüfen;
- es ist ziemlich zeitaufwendig, dies alles wiederholt zu tun. Also müsste ein Kompromiss gefunden werden zwischen der Häufigkeit, mit der solche Regressionstests durchgeführt werden, sowie der erforderlichen Zeit und dem Aufwand, um diese auszuführen;
- es ist schwierig, dieses Vorgehen zu skalieren. Es können nur einige solcher Szenarien auf diese Art und Weise getestet werden, was ein Qualitätsrisiko mit Folgen in Bezug auf Sicherheit, Verlässlichkeit und Konformität darstellt.

Orchestrierung von Test und Validierung

Der Einsatz von Parasoft SOAtest ist ein Weg, um diese verschiedenen Test- und Validierungsaktivitäten über mehrere Schnittstellen und Protokolle zu orchestrieren. In diesem Artikel wird SOAtest verwendet, um zu demonstrieren, wie wesentliche Teststrategien automatisiert werden können.

Datenbank-Initialisierung

Mit dem DB-Tool definieren wir einen Setup Test, der automatisch eine Reihe von Abfragen ausführt und die Datenbank in einen Zustand versetzt, in dem man die Transaktion starten kann. Unter anderem wird der Kontostand auf den Ausgangsbetrag gesetzt.

Einzahlungstransaktion

Danach nutzen wir das SOAP Client Tool, um die $ 500 einzuzahlen. Mit SOAtest können Tests automatisch aus der WSDL, aus Traffic Logs, SOA Registries oder anderen Quellen erzeugt werden. Die Request-Parameter werden in einem XML-Baum dargestellt, mit dem eine visualisierte, nachvollziehbare Konfiguration möglich ist. In diesem Fall benötigen wir nur die entsprechende Kontonummer und den Dollarbetrag, wie in Abbildung 2 angezeigt.

Überprüfung des Kontostands

Anschließend wollen wir überprüfen, dass das Guthaben richtig auf dem Konto gebucht wurde. Wir könnten dazu einen Webbrowser nutzen und uns manuell einloggen, um den Saldo zu prüfen. Wir wollen dieses jedoch zu einem reproduzierbaren, automatischen Schritt machen. Deshalb lassen wir SOAtest die Browser Aktionen in einem Login Szenario aufnehmen, um sie dann im End-to-End-Einzahlungs-Testszenario wenn nötig ablaufen zu lassen.

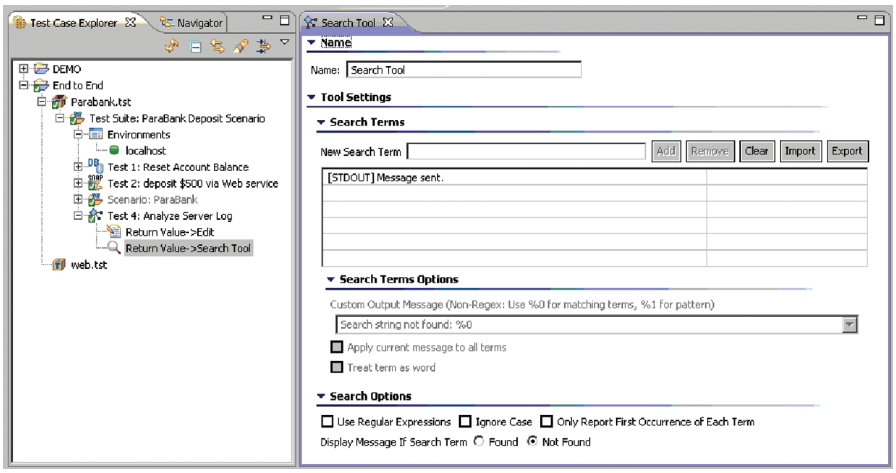

Abbildung 3: Automatisierte Suche in der Logdatei

Problemdiagnose

Angenommen, der Kontostand wird nicht wie erwartet aktualisiert und zeigt nach der Einzahlung nicht den neuen Saldo an. Wie können wir ermitteln, wo die Ursache des Problems liegt?

Überprüfung der Logdatei

Wir wissen, dass OBA einen „message sent"-Eintrag aufzeichnet, wenn ein Einzahlungsauftrag abgearbeitet wird. Damit wird kenntlich gemacht, dass OBA eine JMS-Message erstellt und an das Konto-Backend gesendet hat. Wieder könnten wir uns auf dem Server einloggen und die Logs kontrollieren, aber auch hier bevorzugen wir es, einen Prozess zu haben, den wir zukünftig einfach wiederholen können.

Wir automatisieren diesen Prozess mit dem FTP-Tool, das dafür benutzt werden kann, über SFTP/FTP Dateien zu kopieren oder Befehle auszuführen. Danach verwenden wir das Search Tool, um den Output des FTP-Tools (d. h. die Logdatei) nach der Zeichenkette „Message sent" zu durchsuchen. Ist die Zeichenkette vorhanden, zeigt sie an, dass die Anwendung via JMS einen Einzahlungsauftrag an das Backend gesendet hat.

Nachdem wir das Szenario erneut laufen lassen, stellen wir fest, dass der „Message sent" Eintrag wie erwartet erstellt wurde. Dennoch ist keine Aktualisierung des Kontostandes über die OBA-Webseite sichtbar. In diesem Fall überprüfen wir als nächsten logischen Schritt, ob OBA wirklich eine JMS Message auf dem TIBCO Bus zur Übertragung an das Konto-Backend eingestellt hat.

Überwachung der Transaktionsmeldungen auf dem Bus

Zur Überprüfung, ob OBA den Einzahlungsauftrag richtig im Bus einstellt, nutzen wir den Event-Monitor. Damit verfolgen wir die Meldungen, die über den TIBCO Bus gehen. Dazu fügen wir ein Event-Monitor-Tool zur Testsuite hinzu, wählen die TIBCO Option aus und fügen die benötigten Message Provider jars (Java Archives) zur SOAtest classpath-Konfiguration hinzu. Dies wird auch für verschiedene andere Anbieter wie WebMethods, WebSphere oder Sonic unterstützt.

Wenn wir das Szenario nun mit dem Event-Monitor laufen lassen, der die TIBCO Queue überwacht, stellen wir fest, dass keine JMS-Message in die Queue eingestellt wurde.

Monitoring der Anwendung auf Code-Ebene

Was tun wir als nächstes, nun da wir wissen, dass keine JMS-Message für den Einzahlungsauftrag eingestellt wurde? Es gibt offensichtlich ein Problem in der Anwendung. Sie sendet die Message nicht und die Log-Dateien zeigen keine Probleme an. Um das Problem zu beleuchten, können wir die Anwendung auf Code-Ebene verfolgen und überprüfen, ob der Code für das Einstellen der JMS-Message zur Laufzeit richtig aufgerufen wird.

Der erste Schritt zur Konfiguration eines solchen Monitorings besteht darin, das Startup-Skript der Anwendung so zu verändern, dass es die Parasoft Java Monitoring jars enthält. Dazu ist keine Codeänderung nötig, aber es muss ein zusätzliches Argument zur Kommandozeile hinzugefügt werden. Das Java Run-time Monitoring System verbindet sich dann mit dem JVM Instrumentation API, um verschiedene Ereignisse bei der Programmausführung zu verfolgen und diese auf den Desktop des Testers zurückzumelden. So können wir, immer wenn das Test-Szenario ausgeführt wird, sehen, welche Methoden in der Anwendung aufgerufen werden.

Diese Ereignisverfolgung zur Laufzeit kann als ein wertvolles Tool zur Identifizierung von Problemen und deren Ursachen dienen. Manchmal können Entwickler Anwendungen von ihrem Desktop aus testen, wo der Quellcode in einem IDE wie Eclipse verfügbar ist. Oft ist dies jedoch sehr schwierig, z. B. wenn Anwendungen in einer anderen Umgebung eingesetzt und mit einer Konfiguration betrieben werden, die sich von der Entwicklungsumgebung unterscheidet. Darüber hinaus fehlt QS-Ingenieuren dieser Zugang in der Regel völlig.

Um das Java Runtime Monitoring für die OBA zu konfigurieren, fügen wir die folgenden Zeilen in das Server Startup Script ein:

```
set MONITOR= [Parasoft Java monitoring jar location]
set PARASOFT_AGENT_OPTS=-
javaagent:"%MONITOR%\MONITOR.jar=nobrowser,port=5091",start,filterin=transaction:customer,trace=transaction:customer
set PARASOFT_BOOTCLASSPATH_OPTS=-
Xbootclasspath/a:"%MONITOR%\MONITOR.jar"
REM and then the Java startup command to use these variables:
„%JAVA%" %PARASOFT_AGENT_OPTS% %PARASOFT_BOOTCLASSPATH_OPTS% %JAVA_OPTS%
„-Djava.endorsed.dirs=%JBOSS_ENDORSED_DIRS%"
-classpath „%JBOSS_CLASSPATH%" org.jboss.Main %*
```

Obwohl in diesem Beispiel ein JBOSS-Server verwendet wird, kann dieses Monitoring ebenso bei WebLogic, WebSphere und anderen bekannten Servern eingesetzt werden.

Der Server wird dann wie üblich gestartet, ist nun aber während der Ausführung des Szenarios für das Monitoring bereit. Im Szenario fügen wir dazu ein weiteres Event-Monitoring-Tool zur Testsuite hinzu und wählen diesmal die "Instrumented Java Application" Option aus. Dann lassen wir das Szenario laufen und beobachten die verschiedenen Java-Events, die in den vorgesehenen Klassen auftreten.

Die Ergebnisse des Monitorings enthalten detaillierte Informationen über die Parameterwerte, welche an die verschiede-

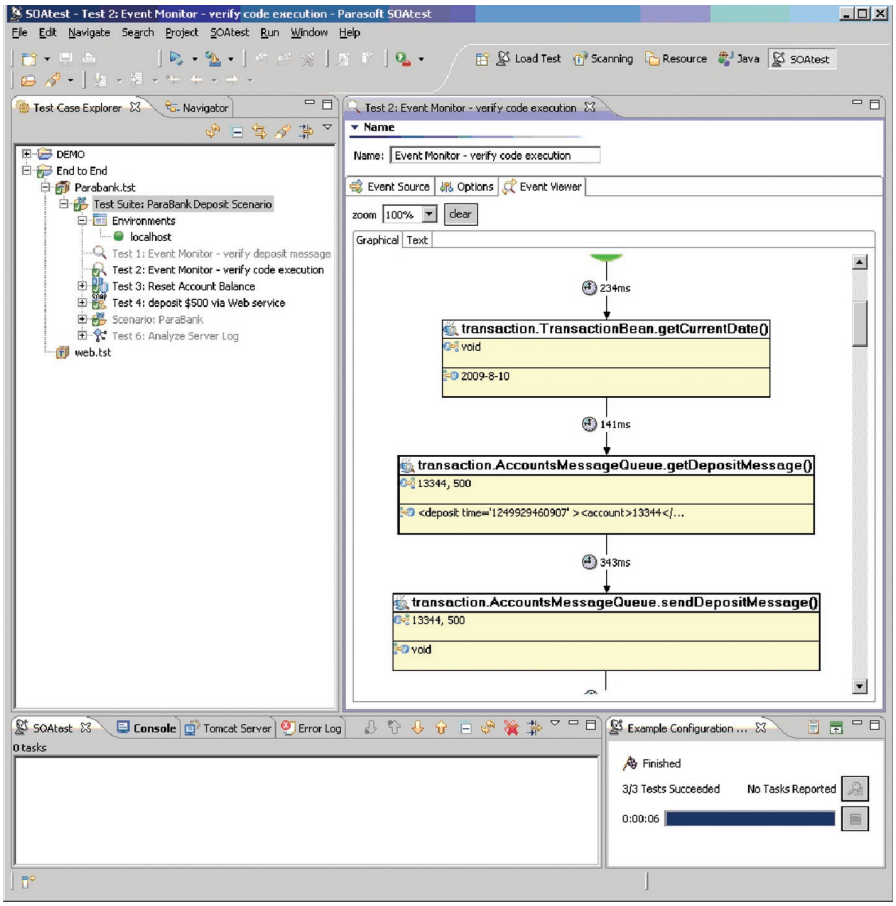

Abbildung 4: Java Event-Verfolgung

nen Methoden übergeben werden, sowie deren Rückgabewerte. Solche Details sind hilfreich, um die Fehlerursache oder wenigstens die Fehlerstelle im Code zu identifizieren. Selbst wenn der zuständige QS-Ingenieur keine Kenntnisse über den Code und dessen Design hat, kann er auf diesem Weg dennoch die betroffenen Klassen und Methoden identifizieren.

Die nächsten Schritte

Generierung von Entwicklertests zur Problemreproduktion

Wenn die fehlerhaften Klassen und Methoden vom QS-Ingenieur identifiziert sind, steht als nächstes die Fehlerbehebung an. Dazu ist es notwendig, den aufgetretenen Fehler in der Entwicklungsumgebung zu reproduzieren, was, wie bereits erwähnt, nicht einfach ist. Um dem Entwicklerteam die Fehlersuche zu erleichtern, kann man die betroffenen Klassen und Methoden mit dem Parasoft Jtest Tracer verfolgen, um anschließend JUnit-Tests zu erstellen, welche das Problem in der Entwicklungsumgebung reproduzierbar machen. Dazu sind einige Schritte notwendig, die hier nur kurz wiedergegeben werden.

Der erste Schritt ist, den Server für das Zusammenspiel mit der Tracer Bibliothek zu konfigurieren. Wiederum ist keine Codeänderung nötig, aber das JVM-Startup-Skript benötigt einige zusätzliche Argumente, wie z. B.:

```
-agentlib:pmt=monitor=transaction.*,nostart,port=1234
```

Als Nächstes fügt der QA-Ingenieur den "Jtest Tracer Client" zum SOAtest Parabank Testszenario hinzu und lässt das Szenario wie gewohnt laufen. Dabei wird nun eine "trace out"-Datei an einem vordefinierten Ort erzeugt. Diese Datei wird an den Entwickler übergeben.

Der Entwickler verwendet in seiner gewohnten Entwicklungsumgebung die bereitgestellte "trace out"-Datei zur Generierung von JUnit-Tests mithilfe von Parasoft Jtest. Diese JUnit-Tests reproduzieren dann die Ereignisse des Parabank-Testszenarios auf Java-Code-Ebene. Dabei werden externe Codeabhängigkeiten wie JDBC- oder JMS-Aufrufe durch Stubs ersetzt, sodass das Verhalten der Anwendung aus der QS-Umgebung, wo das Problem auftritt, in der Entwicklungsumgebung nachgebildet werden kann. Da die Java-Methoden mit den realen Werten aus der QS-Umgebung aufgerufen werden, wird dem Entwickler die Problemanalyse wesentlich erleichtert.

Sobald das Problem vom Entwickler identifiziert und auf Java-Codeebene gelöst worden ist, wird in der Testumgebung das Parabank-Testszenario nochmals ausgeführt, um zu verifizieren, dass das Problem tatsächlich behoben wurde.

Erstellung einer Regression-Testsuite

Ein Schlüsselelement in jeder Teststrategie ist die Fähigkeit, das Verhalten eines Systems in einer Serie von wiederholbaren Tests zu erfassen. Die regelmäßige Ausführung dieser Testreihe stellt sicher, dass sich das erwartete Verhalten einer Anwendung gegenüber den vorgegebenen Spezifikationen nicht negativ verändert.

Nun haben wir für den Use Case „Einzahlung" ein funktionierendes End-to-End Testszenario, welches die Überprüfung auf Browserebene, der Server Logs, der Codeausführung sowie der JMS-Meldungen einschließt und für das auf jeder dieser Ebenen in den unterschiedlichen Schritten entsprechende Erfolgskriterien definiert werden können. Solche Erfolgskriterien stellen sicher, dass, wann immer dieser Test zukünftig ausgeführt wird, all diese Punkte im System überprüft und wir alarmiert werden, wenn Abweichungen vom gewünschten Systemverhalten festgestellt werden. Somit steht eine Regression-Testsuite zur Verfügung, die periodisch und automatisiert ablaufen kann.

Das Verhalten der Anwendung sichtbar machen

Um die Regressionstests in einer Testumgebung lauffähig zu machen, welche das Mainframe-Backendsystem nicht enthält, müssen die Messages, die von der OBA gesendet werden, so erfasst und verifiziert werden, als ob das originale Backendsystem verfügbar wäre. Mit anderen Worten müssen wir es nachbilden oder virtuell abbilden, sodass Entwicklung und Test der OBA durchgeführt werden können, auch wenn das Backendsystem nicht verfügbar ist.

Es gibt verschieden Möglichkeiten, wie wir eine solche Emulation mit SOAtest konfigurieren können. Zum Beispiel kann man das Backend innerhalb derselben Testsuite durch Stubs ersetzen oder eine weitere Funktionalität von SOAtest nutzen und einen SOAtest-Server konfigurieren, der sich wie das Backend verhält.

Zusammenfassung

Dieser Artikel hat gezeigt, wie man verschiedene Test- und Validierungstechniken integriert verwenden kann, um den End-to-End Test und die Validierung für einen Use Case zu automatisieren. Wir haben beschrieben, wie

- automatische Überprüfung auf mehreren Ebenen – auf Browserebene, in den Server-Logs, bei der Codeausführung und auf JMS-Ebene – im Zusammenspiel genutzt werden kann, um funktionale Defekte aufzudecken und zu untersuchen
- Event-Monitoring verwendet werden kann, um die Ereignisse zwischen unterschiedlichen Prozessen, welche durch Tests getriggert werden, sichtbar zu machen und zu verfolgen; somit wird eine schnelle Problemdiagnose direkt aus der Testumgebung ermöglicht
- das „Tracing“ von Testfällen aus einer laufenden Anwendung heraus es möglich macht, schnell und einfach Testfälle zu erstellen, welche der Entwicklung helfen, die entdeckten Fehler zu reproduzieren und zu beheben.

Diese Techniken können alle mit Parasoft SOAtest angewendet werden, um eine vollautomatisch laufende Überprüfung direkt aus der Testumgebung zu ermöglichen. Und das auch dann, wenn Teile des Systems unvollständig, noch in der Entwicklung, instabil, nicht ausführbar oder aus anderen Gründen nicht für die Tests verfügbar sind. Man kann umfangreichere und häufigere Tests mit den vorhandenen Ressourcen durchführen und letztlich sicherstellen, dass konforme, verlässliche und sichere Anwendungen termin- und kostengerecht entwickelt und ausgeliefert werden.

Für weitere Informationen, z. B. über Jtest Tracer, die Erstellung von Regressionstests oder zur Visualisierung von Anwendungen und Komponenten, wenden Sie sich an Parasoft Deutschland GmbH, Mähringer Weg 45, 89075 Ulm, Tel. 0731 880309-0, Email: info-de@parasoft.com, oder besuchen Sie das Parasoft End-to-End System Testing Center auf unserer Webseite.