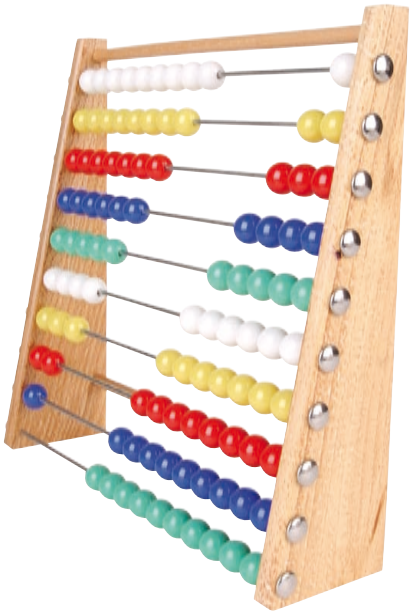
Softwareindustrialisierung mit dem Java-basierten Banking Framework

Do you agree?

VON MATTHIAS SCHORER UND DIETER PETERS

Frameworks sind aus der heutigen Java-Entwicklung nicht mehr wegzudenken. Dieser Artikel beleuchtet, welchen Weg die Fiducia IT AG bei ihrer Java-Anwendungsentwicklung verfolgt und welche Produkte mit dem Fiducia eigenen Framework JBF (Java-based Banking Framework) entwickelt werden.



Uns wird ab und zu die Frage gestellt, warum wir den Aufwand betreiben, für unsere Softwareentwickung ein eigenes Framework zu verwenden, statt „einfach" kommerzielle oder frei verfügbare Lösungen zu wählen. Die Antwort ist einfach: weil es sich lohnt, eine in Form von Frameworks implementierte Basis und Toolchain für Softwareentwicklung einzusetzen. Unter dem Label JBF ist eine mittlerweile über viele Jahre gereifte und bewährte Kombination aus Architektur, Programmiermodellen und Design Patterns, generischen Softwarekomponenten, APIs und aufeinander abgestimmten Laufzeitkomponenten entstanden. Die Komponenten basieren je nach Einsatzzweck auf Java-Standards oder Open-Source-Produkten und werden durch Eigenentwicklungen vereinfacht und ergänzt. Das Framework unterstützt so optimal die Anforderungen und das Umfeld: Große und sehr große Softwareentwicklunsprojekte im Zusammenspiel mit mainframebasierten Kernbankensystemen und die Einbettung in eine äußerst große und verteilte IT-Infrastruktur.

Als die Fiducia IT AG sich 1998 entschloss, für Neuentwicklungen außerhalb des Mainframe Java einzusetzen, gab es weder Applicaton Server, EJBs noch generische Persistenzlayer wie z.B. JDO. Zusätzlich musste das mainframebasierte Kernbanksystem angesteuert werden, das nur über SNA kommuniziert sowie die in Banken verwendeten Peripheriegeräte wie Sparbuchdrucker und automatische Kassentresore. Dieses Umfeld war der damals noch recht jungen Java-Welt fremd, daher bedurfte es der Adaption und der Schaffung geeigneter Lösungen für die Entwicklung und den Betrieb von Java-Anwendungen.

agree – die treibende Kraft

Nach mehreren Fusionen sah man sich vor der Herausforderung, vier verschiedene Backend- und Frontend-Systeme auf nur ein System zusammenzuführen. Gleichzeitig waren vom Markt geforderte Funktionalitäten, vor allem im Bereich Bankvertrieb, zu implementieren. Als Lösung entwickelte die Fiducia das neue Kernbanksystem agree mit der Arbeitsoberfläche agree BAP (Bankenarbeitsplatz). Gleichzeitig sollte das Client-Betriebssystem OS/2 auf Windows XP migriert werden.

Da eine solche Migration bei mehr als 100.000 Arbeitsplätzen und mehr als 10.000 Geldausgabeautomaten nicht als „big bang" stattfinden kann, profitierte Fiducia davon, dass sie schon bei der Client-Entwicklung 1998 auf Java gesetzt hatte. Der JBF-basierte Rich Client ermöglichte ihr den Betrieb sowohl auf OS/2 als auch auf Windows XP. Unterschiede der JVMs auf den Plattformen, wie z.B. das Fokus-Handling, wurden hierbei durch den JBF-Gui-Layer, genannt XBF, abgefangen. Bis zu 640 Entwickler haben seit 2003 gleichzeitig an agree gearbeitet und erzeugten mittlerweile 22 Millionen Lines of Code in 84.300 Java-Klassen.

Aufgrund der gigantischen Anzahl an Klassen und den mehr als 800 JAR-Files, in denen das Frontend agree BAP ausgeliefert wird, wurden bisher unentdeckte Fehler sowohl in der JVM als auch in Java Web Start gefunden. Ein von der Fiducia entwickelter Fix für Java Web Start [2], der in der JVM ab der Version 1.4.9 ausgeliefert wurde, verminderte sowohl den Speicherverbrauch als auch die Ladezeit von Java Web Start um 50 Prozent. Die Vorgehensweise bei der Entwicklung wurde von Forrester als „Best Practice" veröffentlicht. Darin wurde besonders positiv herausgestellt, dass sowohl die gesamte Softwareentwicklung, ganz gleich ob für Filiale (agree BAP), SB-Lösung (agree SB) oder Homebanking als auch deren Betrieb auf Basis einer durchgängigen Technologie, nämlich JBF stattfindet.

Die aus der Entwicklung von agree gewonnenen Erfahrungen flossen in die

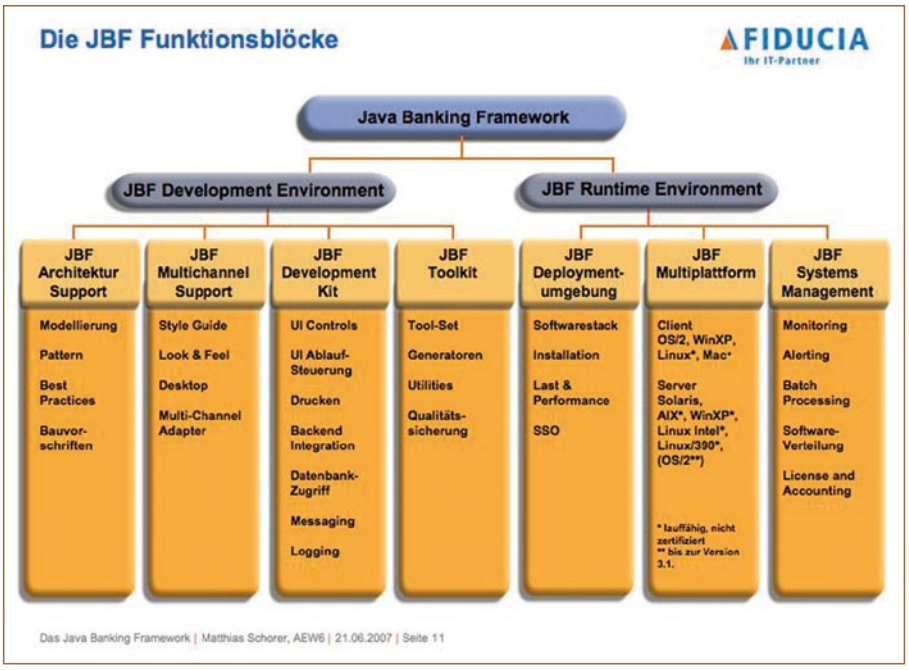


Abb. 1: JBF-Funktionsblöcke

agree-Standardarchitektur (aSA) ein. Sie enthält Vorgaben für den Komponentenschnitt und gibt u.a. Richtlinien für das Datenbankdesign vor. Im Juni 2007 wurden die letzten Partnerbanken der Fiducia auf agree migriert. In Zukunft soll die Weiterentwicklung von agree und seiner technischen Plattform in die Hand genommen werden.

Während sich agree mit der Konsolidierung der Backend- und Frontend-Systeme beschäftigte, geht es bei dem neuen Projekt um die Verminderung der Abhängigkeit vom Mainframe. Gründe hierfür sind unter anderem der schwindende Skill im Mainframe-Bereich als auch Marktanforderungen, die mit dem bestehenden System nur schwer umsetzbar sind. Mit der Weiterentwicklung von agree verlagert die Fiducia Massendatenverarbeitung vom Mainframe in die dezentrale Welt von Unix und Java. Das ehrgeizigste Teilprojekt ist dabei die Implementierung von SEPA (Single Euro Payments Area) [3]. SEPA löst ab 1. Januar 2008 den bisher gebräuchlichen Auslandszahlungsverkehr in Europa ab. Ab 2010 wird dann der komplette Zahlungsverkehr der Partnerbanken mit bis zu 25 Millionen Buchungen am Tag über SEPA abgewickelt.

Das JBF-Framework

Wie aus Abbildung 1 ersichtlich, besteht JBF aus vielen Komponenten. Das Java Framework besteht aus einem Client- und einem Server Framework. Das weiter oben schon erwähnte XBF ist hierbei nicht nur der GUI Layer, sondern hat auch einen Serveranteil, die so genannten XBF Services, die im JASS – dem JBF Application Server System – laufen (Abb. 2). XBF ist der Grundstock, mit dem der Anwendungsentwickler arbeitet.

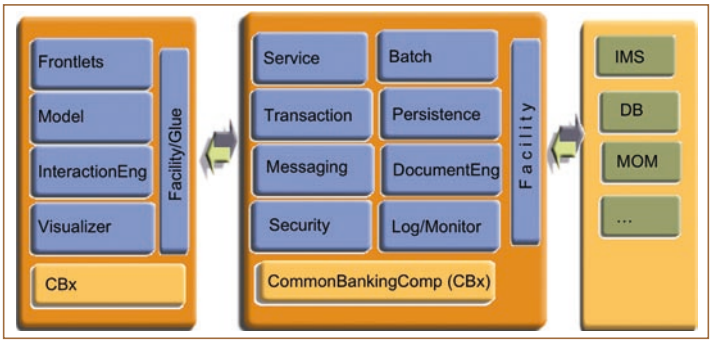


Abb. 2: JBF Client-/Serverarchitektur

XBF bietet dem Entwickler nicht nur GUI Controls (Abb. 3), sondern wesentlich mehr. XBF stellt den Anwendungsrahmen agree BAP und auch die notwendigen Visualizer zur Verfügung. Dies ist erforderlich, um über alle Projekte hinweg ein einheitliches Aussehen und Handling zu erreichen.

Auf der einen Seite wird der Entwickler in seinen Möglichkeiten eingeschränkt, auf der anderen Seite wird er entlastet und kann sich auf die fachlichen Inhalte seines Projekts konzentrieren. Diese werden auf der Client-Seite als XBF Frontlets und auf der Serverseite als XBF Services implementiert. Ein Frontlet ist eine Interaktionskomponente, d.h. eine Komponente, mit der ein Benutzer interagieren kann. Einfach ausgedrückt, realisiert ein Frontlet einen Dialogschritt. Mehrere Frontlets können zu einer Dialogfolge kombiniert werden. Auf diese Weise kann die gesamte Benutzerschnittstelle einer Anwendung mithilfe einzelner, klar voneinander abgegrenzter Bausteine erstellt werden. Die eigentliche Erzeugung, Visualisierung und Steuerung der Frontlets übernehmen die jeweils dafür spezialisierten Framework-Module.

Ein Frontlet ist grundsätzlich nach dem MVC-Muster aufgebaut. Es besteht im Wesentlichen aus einem Modell, einer View und Commands, die hier die Aufgaben des Controllers übernehmen. Das Modell kapselt den internen Zustand eines Frontlets. Die View dient zur Visualisierung des Modells gegenüber dem Benutzer. Die Commands implementieren die Interaktionslogik und werden in der Regel durch Benutzerinteraktionen ausgelöst. Sie modifizieren das interne Modell und steuern den Interaktionsfluss, indem sie beispielsweise weitere Frontlets anfordern. Die Frontlets der gesamten Anwendung bilden dabei eine Baumstruktur. Die eigentliche Darstellung der einzelnen Frontlets und deren Beziehungen untereinander obliegen dem Visualizer, der durch das Framework implementiert wird. Ein Visualizer ist in XBF ein Konzept, welches kanalspezifisch umgesetzt wird. Konkret gibt es aktuell Unterstützung für die beiden Kanäle Swing und HTML.

Ein XBF Service ist ein Dienst, welcher einen Teil der Business-Logik einer Anwendung implementiert. Ein Service hat einen systemweit eindeutigen Namen, über den dieser angefordert werden kann. Er hat fachlich definierte Ein-/Ausgabeschnittstellen, die *Request* und *Response* genannt werden. Schließlich hat er auch eine Implementierung, die die eigentliche Fachlogik enthält.

Ein Service kann sowohl synchron als auch asynchron ausgeführt werden. Bei synchroner Ausführung hält die aktuelle Programmausführung beim Aufrufer an, bis der Service antwortet, danach wird die Programmausführung fortgesetzt und die Antwort kann ausgewertet werden. Bei asynchroner Ausführung wird die Programmausführung nicht unterbrochen. Die Ausführung des Services erfolgt dabei in einem eigenen Thread. Die Service-Ergebnisse können in diesem Fall also nicht unmittelbar ausgewertet werden.

Entwicklung mit JBF

Die Entwicklung mit JBF teilt sich in einen dreistufigen MDA-Prozess. Im ersten Teil des Prozesses erfolgt eine Modellierung der unterschiedlichen Datenmodelle und Services innerhalb des Rational Software Architects (RSA). Für jeden Service wird hierbei das Request- und Response-Interface definiert, das dann wieder die definierten Datenmodelle verwendet. Für die Datenmodelle werden die einzelnen Properties und Datentypen definiert sowie ihre Abhängigkeiten untereinander.

Bei Datenmodellen unterscheidet man nach Transferobjekten und Datenobjekten. Erstere werden für die Kommunikation zwischen Modulen genutzt, letztere nur innerhalb eines Moduls. Für den GUI Layer werden Interaktionsmodelle als das Modell des MVC Patterns erstellt.

Als zweite Phase des Prozesses schließt sich die Generierung der notwendigen Artefakte an. Zum einen werden die notwendigen Sourcecode-Artefakte, zum anderen XML-Konfigurationen für die Serviceregistrierung als XML-Dateien für die Anbindung der Persistenzschicht erzeugt. Innerhalb der Persistenzkonfiguration wird auch das Mapping zwischen Datenobjekt und Datenbankschema definiert. Die so generierten Artefakte werden durch den Entwickler in der dritten Phase mit Businesslogik versorgt. Hierbei werden die Codeabschnitte in den Artefakten durch geschützte Bereiche vor einem Überschreiben bei einem erneuten Generieren bewahrt.

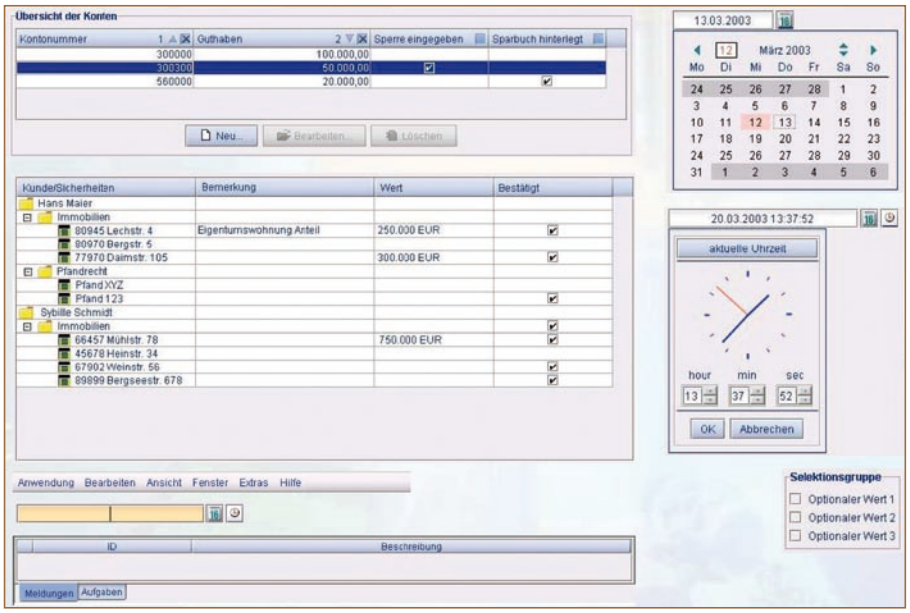


Abb. 3: Beispiele für UI Controls

Die Erstellung der GUIs erfolgt ein wenig abweichend vom vorhergehend beschriebenen Prozess. Das Design der einzelnen „Masken" erfolgt mit einem speziellen, eigengeschriebenen Plug-in innerhalb des RSA. Dabei erzeugt der GUI Builder direkt Sourcecode. Über ein weiteres Plug-in, dem Frontlet Builder, wird der Controller erzeugt und über einen Binding Builder die Verknüpfung zwischen dem Datenmodell und den Feldern der Oberfläche definiert. Durch den eingesetzten MDA-Ansatz und die Unterstützung mit Modellierungswerkzeugen und Generatoren wird dem Entwickler viel Arbeit abgenommen. Dadurch kann sich der Entwickler auf die Implementierung seiner Fachlogik konzentrieren.

Testautomatisierung

640 Entwickler, 22 Millionen Codezeilen, 84 300 Java-Klassen erfordern ein sorgfältiges und automatisiertes Testen. Daher werden den Projekten gleich mehrere Werkzeuge für Tests an die Hand gegeben. Mit dem *End2End-Lasttest-Bundle*, einem auf JUnit-basierenden Framework, kann der Entwickler Tests für seine Anwendungsfälle schreiben und über Property-Dateien verschiedene Szenarien durchspielen. Vorgeschrieben ist mindestens ein solcher Test pro Anwendungsfall bzw. Service. Das Framework wurde so konzipiert, dass diese Testfälle später für die Lasttests mittels Loadrunner [4] genutzt werden können. Die Lasttests finden in einer produktionsnahen Umgebung statt, und es werden aus der Praxis bekannte Lastprofile aufgesetzt.

Service Request und Response

Service Request und Response Java Interfaces, die von Framework Basis-Interfaces abgeleitet werden müssen. Die Implementierung wird zur Laufzeit vom Framework erzeugt.

Codebeispiele:

```
public Interface ServiceBeispielRequest extends
                                        ServiceRequest {
 public void setKriterium(String kriterium);
 public String getKriterium();
}

public Interface ServiceBeispielResponse extends
                                    DefaultServiceResponse {
 public void setErgebnis(int ergebnis);
 public int getErgebnis();
}
```

Über das Basisinterface wird das Statusattribut geerbt. Laut Namenskonvention beginnen die Interface-Namen immer mit Service und enden auf Request bzw. Response.

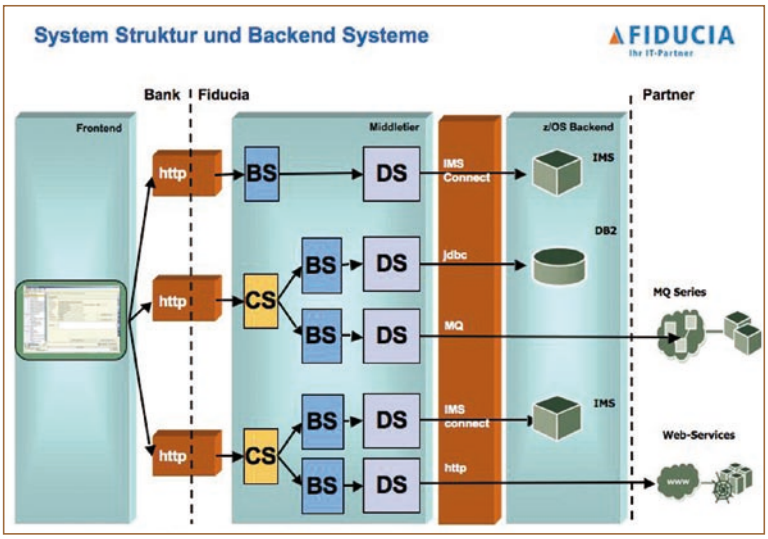


Abb. 4: Die Schichten von agree

Durch Nutzung der von den Entwicklern geschriebenen fachlichen Tests wird erreicht, dass die erhaltenen Daten auch realistisch sind. Alle JUnit-Tests werden zusätzlich im Build-Portal für Integrationstests genutzt. Oberflächentests werden mit Quicktest durchgeführt.

Service Implementierung

Für die Implementierung eines Service steht die Basisklasse ServiceCommand zur Verfügung, in der Implementierungsklasse muss nur noch die Methode *execute* ausgearbeitet werden.

Codebeispiel:

```
public class ServiceBeispiel implements ServiceCommand
{
public void execute(ServiceCommandEnvironment env)
{
 // Request und Response aus dem Environment holen
 ServiceBeispielRequest request =
            (ServiceBeispielRequest) env.getRequest();
 ServiceBeispielResponse response =
           (ServiceBeispielResponse) env.getReponse();

 // sinnvoller fachlicher Code
 int ergebnis = 0;
 if ("42".equals(request.getKriterium())) {
  ergebnis = 23;
 }

 // Ergebnisse in den Response setzen
 response.setErgebnis(ergebnis);
  response.setSatus(Status.OK);
 }
}
```

Auch bei der Implementierungsklasse eines Service gelten Namenskonventionen: der Klassenname beginnt immer mit „Service".

Betrieb und Deployment

agree wird auf mehr als 250 Sun V440 Servern mit Solaris betrieben. Als Basis für das JBF Application Server System (JASS) dient Apache Tomcat 4 in Kombination mit dem Apache-Webserver. Die Server sind zu sog. Portalen mit maximal 16 Servern zusammengefasst. Das Loadbalancing wird über MOD_JK gesteuert. Ein Portal zeichnet sich dadurch aus, dass alle in ihm enthaltenen Server den identischen Softwarestack haben. Auf einem Portal werden bis zu 100 Banken bzw. 6 000 Arbeitsplätze betrieben. Mit dieser Infrastruktur werden 35 Millionen Serviceaufrufe pro Tag allein im Filialgeschäft verarbeitet. Zusätzlich gibt es noch mehrere Java-Web-Start-Downloadportale, über die die Banken mit neuer Client-Software versorgt werden. Die JBF-Entwickler haben einige Änderungen am Java-Web-Start-Verfahren [2] durchgeführt, um die Software für den Massenbetrieb tauglich zu machen. So kann die Fiducia zentral steuern, welche Bank welchen Softwarestand bekommt und ob die Software in der Nacht „preloaded" werden soll oder ob sie unter Tags ad-hoc versorgt wird. Ebenso einfach kann man im Fehlerfall wieder auf eine vorhergehende Version schwenken.

Fazit und Ausblick

Die Frage, ob es sinnvoll ist, ein eigenes Framework einzusetzen, kann die Fiducia definitiv mit ja beantworten. Hätte die Fiducia zum Beispiel von Anfang an auf EJB in der damaligen Version 1.0 gesetzt, dann hätte bereits zweimal sehr viel Aufwand in das notwendige Refactoring gesteckt werden müssen, um die Anwendung konform zur jeweilig gültigen EJB-Spezifikation zu machen. Stattdessen konnten die Ressourcen für die Erweiterung von agree eingesetzt werden. Ebenso macht es Sinn, die eingesetzte Open-Source-Software durch eine eigene Framework-Schicht zu kapseln. Denn hierdurch wird es einfacher, Basiskomponenten auszutauschen. Als Beispiel sei hier der theoretisch mögliche Austausch der heute auf JDO-basierenden OR-Mapping-Schicht genannt. Eine zukünftige Version von JBF könnte hier auch auf Hibernate setzen, ohne dass die Anwendungen angepasst werden müssten. JBF wird aber nicht nur innerhalb der Fiducia eingesetzt, sondern auch von Partnern im genossenschaftlichen Finanzverbund. Natürlich lässt sich mit JBF nicht nur Software für den Finanzbereich entwickeln. JBF ist flexibel genug, um als Basis für beliebige, große Softwaresysteme zu dienen, bei denen es auf hohe Skalierungsfähigkeit und schnelle Implementierung ankommt. Die Fiducia hat sich weiter entschlossen, in unregelmäßigen Abständen JBF Tools für die Java Community bereitzustellen, zum Beispiel das Tool „JnlpRunner" [5] mit dem es möglich ist, Java-Web-Start-basierte Anwendungen zu laden und zu debuggen.

Dieter Peters ist Berater bei der syngenio AG. Seit 2000 entwickelt er mit Java und war lange Jahre als JBF-Coach bei der Fiducia IT AG tätig.

Matthias Schorer ist Java-Evangelist und kam 1998 zur Fiducia München, wo er die Entwicklung der Java Banking Frameworks leitete. Seit 2003 ist er Technischer Chefarchitekt und verantwortet die Architektur von agree und Horizon sowohl auf der Entwicklungs- als auch auf der Betriebsseite.

Links & Literatur

[1] www.jbfone.de

[2] developers.sun.com/learning/javaoneonline/j1sessn.jsp?sessn=TS-3212&yr=2006&track=desktop

[3] de.wikipedia.org/wiki/Single_Euro_Payments_Area

[4] www.mercury.com/us/products/performance-center/loadrunner/

[5] www.jbfone.de/jnlprunner