

Ausgabe 8.2001

**WebReporting mit JReport: Ein Bericht aus der Praxis**

von Thomas Kestler

# Schritte ins Web

eCommerce auf Hochtouren – alles ist „e“. Doch was, wenn die schönen Versprechungen in die Praxis umgesetzt werden sollen? Der folgende Beitrag schildert die Praxiserfahrungen eines WebReporting-Projektes bei einer großen Bank. Die Aufgabenstellung: eine alte Datawarehouse-Anwendung soll auf Web-„Beine“ gestellt werden.

Um es vorweg zu nehmen, es ist gelungen diese Lösung produktiv in die Praxis umzusetzen. Der Kunde ist sehr zufrieden und die Lösung auch tragfähig und skalierbar. Der Eindruck, dass es leicht war, dies zu erreichen, täuscht aber nur zu sehr. Ein wichtiger Erfolgsfaktor war hierbei mit Sicherheit, dass das Projekt eine (noch) überschaubare Größe hatte.

Doch von Anfang an. Das Altsystem war ursprünglich unter MS-Access 2.0 entwickelt worden und schleppte daher noch etliche Restriktionen aus Version 2.0 mit. Daten und Reports waren auf mehrere *mdb*-Dateien verteilt, die auf die Arbeitsplätze der Sachbearbeiter monatlich per Software-Distribution verteilt wurden. Dieser Softwareversand verursachte erhebliche Kosten. Zudem mussten die Daten zuvor nach Regionen aufgeteilt werden und wurden dann regionsweise versandt. In der Filiale bestand eine Trennung zwischen lokalen Daten und Daten auf dem Filialserver. Der Anwender konnte Daten in die lokale Datenbank replizieren, um diese Daten mit auf dem Notebook zum Kunden nehmen zu können. Nun sollte diese Anwen-

dung abgelöst werden und dabei standen Plattformunabhängigkeit, zentrale Datenhaltung und moderne Webtechnologie als zentrale Eckpunkte im Anforderungskatalog.

Wenn eine solche Altanwendung in eine Webanwendung umgewandelt werden soll, muss zunächst geklärt werden, was wirklich benötigt wird, was mit vertretbarem Aufwand gelöst werden kann und auf was verzichtet werden kann. Ein Dorn im Auge war der Offline-Betrieb der Altanwendung. Die Begriffe Offline und Web vertragen sich nicht (oder nur schlecht). Natürlich hätte man sich vieles ausdenken können: eine lokale Datenbank, eine Java Applikation, Datenreplizierung, automatischer Update der Java Applikation, usw. Nur, damit war die Lösung im vorgegebenen Budget nicht realisierbar und die Folgekosten wären deutlich höher – und diese waren ja die Motivation für die Ablösung. Also wurde diese Forderung gekippt und dadurch ersetzt, dass der Anwender zuvor Reports des zu besuchenden Kunden erzeugt und auf seinem Notebook abspeichert. Ganz klar ein Kompromiss, aber einer, der massiv Kosten sparte. Als Option

besteht auch die Möglichkeit, dass sich der Anwender von außen (vom Kunden) in das System einwählen kann. Man kann aber generell sagen, dass man bei der Ablösung einer Altanwendung durch eine Weblösung sicherlich immer einen gesunden Schnitt abringen muss. Gelingt das nicht, sollte man das Projekt nicht angehen (ich zumindest halte das so).

Eine weitere Anforderung war die Abbildung einer Funktion aus dem Altsystem: dem Szenario-Manager. Dies ist ein Kalkulationswerkzeug, mit dem der Anwender Zahlungsverkehrsszenarien durchspielen und darüber entsprechende Reports erzeugen kann. Da schon zu Beginn Java präferiert wurde, blieb hier aber nur Java 2 als erfolgversprechende Basis. Zudem sollte die Lösung ja auch zukunftssicher sein.

Dies alles führte zu dem Ansatz, die Anwendung als Java-2-Applet zu realisieren. Leider ist die Java2-Unterstützung der Webbrowser immer noch bescheiden. So muss auf jedem (Windows NT) PC das Java 2 PlugIn installiert sein. Nun hat die Bank aber noch vorwiegend OS/2 (Warp 3.5) im Einsatz und dieses unterstützt Java

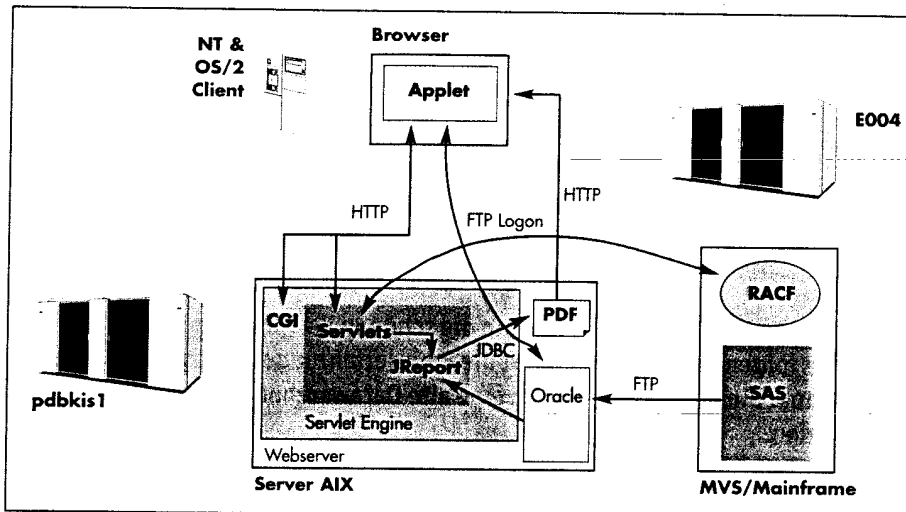


Abb. 1: Prinzipstruktur ursprüngliche Architektur

2 nicht (erst ab Warp 4). Eine Rückportierung auf Java 1.1.8 schien nicht besonders attraktiv und wurde auch nicht in Betracht gezogen. Statt dessen sollten die OS/2-Anwender eine vereinfachte HTML-Schnittstelle erhalten. Zum einen bekommen immer mehr Anwender Windows NT und zum anderen ist der Rollout von Warp 4 bereits absehbar. Für die Zwischenzeit sollte es das HTML-Interface also tun.

Wie sollten die Reports erzeugt werden? Die einfachste Lösung ist die Erzeugung von HTML-Dateien. Dafür existieren ja reichlich Ansätze. Wer aber bereits versucht hat, mit HTML professionelles Reporting zu erstellen, wird schnell am Ende sein. HTML ist eben keine Seitenbeschreibungssprache. Was stand zur Auswahl? Zunächst wurde überlegt, die Reports selbst zu erzeugen und dazu einen kleinen Reportgenerator in Java zu erstellen, der RTF-Dateien erzeugt. RTF ist durchaus dafür geeignet. Der Aufwand hierfür wäre jedoch beträchtlich gewesen. So war es naheliegend, sich auf dem Markt nach bestehenden Reportinglösungen auf Java-Basis umzusehen. Mehrere Produkte wurden evaluiert und die Wahl fiel auf JReport Enterprise Server von JInfonet, ein pure Java Report Generator der als Ausgabeformate u.a. PDF, PS und HTML unterstützt ([www.jreport.de](http://www.jreport.de)).

Die Architektur sah für den Client einen Webbrowser (Netscape) und Acrobat Reader (für die PDF-Dateien) vor. Ein AIX-Server sollte die Datenbank, den Re-

port Server, den Webserver, den Application Server (IBM WebSphere) und ein Servlet beherbergen.

Als zentrale Datenbank wird Oracle 8.0.4 unter AIX eingesetzt. Für die großen Datenmengen der Anwendung eine adäquate Plattform. Da Oracle in der Bank Standard ist, war es auch möglich, sich auf zentrale Features von Oracle zu konzentrieren. Insbesondere kamen deshalb Stored Procedures für die Abbildung der Geschäftslogik in Betracht. Ziel war es, dass sowohl der Java2-Client als auch die Reports möglichst einfach gehalten werden sollten (schließlich könnte es ja auch passieren, dass man das Reporting Tool im laufenden Projekt wechseln muss – was hier nicht der Fall war). Somit wurde ein Großteil der Logik in Stored Procedures integriert. Durch die zentralen Stored Procedures (Functions) war es möglich, ein zentrales Logging zu implementieren. Über eine Parametertabelle kann hierbei der Detailierungsgrad eingestellt werden. Wenn im Betrieb Probleme auftreten, kann anhand einer Logging-Tabelle der Ablauf und aufgetretene Fehler nachvollzogen werden.

Ein zentrales Anwendungs-Servlet wurde als Schnittstelle zwischen dem Frontend (Java 2-Applet, HTML-Dialoge) und dem Report Server entwickelt. Das Servlet hat folgende Aufgaben: Ausführen vorbereitender Stored Procedures vor Ausführen der Reports, Benutzerführung der HTML-Dialoge (für die armen

OS/2 Anwender), und Bereitstellung eines File Viewer Tools auf HTML-Basis. Aufgrund dieser vielfältigen Anforderungen wurde das Servlet dann auch etwas dick, eine Aufteilung in drei Servlets (unter Verwendung zentraler Klassen) wäre in nachhinein sicher besser gewesen, ließe sich aber leicht auch noch nachträglich implementieren. Das File Viewer Tool wurde nötig, weil der Anwender die Möglichkeit bekommen sollte, die auf dem Server erzeugten Reports nachträglich zu verwalten. Dazu wurde eine HTML-Oberfläche realisiert, die vom Servlet verwaltet wird. Das Servlet erzeugt die HTML-Seiten dynamisch aus HTML Template Dateien (ganz klar eine Sache für XML, aber dazu reichte die Zeit nicht mehr). Natürlich darf jeder Benutzer nur seine Dateien sehen und bearbeiten und natürlich führt das Servlet auch ein Session Management durch. Erstaunlich, an was man so alles denken muss. Die Kommunikation zwischen dem Servlet und dem Frontends erfolgt über HTTP POST-Befehle. Das hatte den Vorteil, dass darauf später das HTML-Interface leicht aufgesetzt werden konnte. Im Gegensatz zum GET-Befehl sind beim POST die Parameter an das Servlet nicht sichtbar (Sicherheit). Das Servlet greift über JDBC auf die Oracle Datenbank zu. Da aber stetig andere Sessions auf das Servlet zugreifen, hätte das Servlet für jede Session eine Connection offen halten müssen. Bei bis zu 3000 Anwendern hätte das zu Problemen geführt. JDBC 2.0 führt zwar Connection Pooling ein, dazu benötigt man aber auch einen JDBC 2.0-Treiber für Oracle (nicht für Version 8.0.4). Also eröffnet das Servlet eine anonyme Connection zum Ausführen der Stored Procedures und bei Bedarf eine konkrete Connection für benutzerbezogene Aktivitäten (Identifizierung, Anmeldeprozedur, Passwort ändern). Das Servlet verfügt über ein detailliertes Logging, dessen Detaillierungsgrad per Parameter einstellbar ist. Schließlich protokolliert das Servlet alle ausgeführten Reports in einer Billing-Tabelle für die spätere Kostenabrechnung.

Die Entwicklung der Anwendung fand unter Windows NT statt, in der Annahme, dass gemäß „compile once, run anywhere“ die Integration in die Produktionssysteme

keine größeren Probleme bereiten sollte. Schließlich wurden alle Betriebssystemabhängigkeiten durch Parameter (Pfadnamen etc.) entschärft. Natürlich war von Anfang an klar, dass es ohne Last- und Performancetests auf der Zielplattform in einer produktionsnahen Umgebung nicht geht. Leider verzögerte sich die Bereitstellung dieser Umgebung, sodass die Test erst relativ spät durchgeführt werden konnten. Es traten dann unspezifische Performanceprobleme auf, die trotz großem Aufwand nicht geklärt werden konnten. Die Tests wurden mit Java 1.1.8 und Java 1.2 durchgeführt. Die Ausstattung des AIX-Servers (RS6000 SP2 4 RISC-Prozessoren 110 MHz, 4GB RAM) hätte voll ausreichen müssen. Vergleichstests unter Solaris (Ultra Sparc 10 und E450) brachten hingegen hervorragende Ergebnisse, auch wenn man hier Äpfel mit Birnen vergleicht. Ein Sun-Rechner war jedoch für dieses Projekt nicht zu bekommen, dafür ein NT-Server (Compaq ProLiant 3000, PIII 500 MHz, 256 MB RAM). Da JReport unter Windows NT während der Entwicklung gut lief, wurde letztendlich die

Entscheidung getroffen, JReport Enterprise Server, den Webserver Tomcat und das Anwendungsservlet auf dem NT-Server zu installieren und zu betreiben. Wer hätte das gedacht? JReport läuft unter NT4.0 stabil und mit guter Reaktionszeit.

Der Wechsel in der Architektur brachte aber ein neues Problem mit sich. Das Java 2-Applet greift über JDBC auf den Datenbankserver zu. Dies ist aber nichts anderes als eine Socketverbindung und solche fallen unter die Jurisdiktion des Security Managers (Applet Security). Warum? Weil das Applet vom Webserver geladen wird und dieser nun der NT-Server ist. Die Datenbank liegt auf dem AIX-System, also ein anderer Rechner, mit anderer IP-Adresse. Es gab drei Lösungsansätze:

- Einsatz des Oracle Connection Managers, einer Art Proxy, der auf dem NT-Server laufen sollte und die Anfragen an den AIX-Server weiterleitet oder
- erteilen der Erlaubnis zum Socket-Connect auf den AIX-Server für das Applet, oder
- Verwendung von Signed Applets.

Leider traten beim Einsatz des Oracle Connection Managers Probleme bei Zugriffen über JDBC auf. Also entschieden wir uns für den zweiten Weg und schalteten dem Applet die Verbindung zu Port 1521 (Oracle Listener) auf dem AIX-Server frei. Natürlich hätten wir den Webserver, den Application Server und das Servlet auch auf dem AIX-Server belassen können und nur JReport auf den NT-Server auslagern. Dagegen sprachen aber einige Punkte, vor allem die knappe Zeit.

#### Fazit:

Ja, es ist gelungen, ein Web-Reporting-Projekt mit einer Pure Java Lösung erfolgreich in die Praxis umzusetzen. JReport war den Anforderungen des Enterprise Reporting gewachsen. Die Architektur (Three Tier) erwies sich als tragfähig, flexibel und skalierbar – aber es gab auch Herausforderungen. Und das Projekt hatte nur Erfolg, weil es gelungen war, zu Beginn des Projektes den Anforderungskatalog an die Möglichkeiten einer Weblösung anzupassen. Diese Erfahrung wird auch für die Zukunft wichtig sein. ■



**Noch Fragen?**

**[www.javamagazin.de/forum](http://www.javamagazin.de/forum)**