Das Tippspiel mit JVx

Die Challenge war für uns die ideale Möglichkeit, das Enterprise Application Framework JVx offiziell der breiten Masse vorzustellen. Ein bis dato unbekanntes Framework tritt gegen bekannte Größen der Java-Welt an. Gestatten, JVx mein Name!

von René Jahn

Auf der Suche nach zeitgemäßen Möglichkeiten zur Bekanntmachung von JVx stießen wir eher zufällig auf die W-JAX-Challenge. Das Thema „Tippspiel für die WM 2010" weckte aber sofort den Entwicklerinstinkt und formierte ein Projektteam mit Roland Hörmann, Martin Handsteiner und René Jahn. Die Tatsache, dass nur noch drei der möglichen sechs Wochen für die Umsetzung zur Verfügung standen, löste zwar kurzfristig Bedenken aus, doch wer nicht wagt, der nicht gewinnt.

Das Framework war von Anfang an klar, denn einerseits wollten wir es vorstellen und andererseits ermöglichte es uns die Umsetzung der gewünschten Anforderungen sowie unsere eigenen Vorstellungen von einem Tippspiel. Mit JVx entwickeln wir im Normalfall individuelle Datenbankanwendungen für mittlere und große Unternehmen in einer Drei-Schichten-Architektur. Es ermöglicht die Erstellung von Technologie mit unabhängigen User Interfaces, z. B. Java-Desktop, Java-Webstart/Applet, Android, und beinhaltet alle notwendigen Komponenten für die einzelnen Schichten. Unser User Interface sollte auf jeden Fall den Komfort einer Desktopanwendung bieten, z. B. Load-on-Demand von Daten, Listen mit Scrolling ohne zu Pagen u. v. m. Deshalb setzten wir auf einen MDI-Swing-Desktop, der sowohl im Browser als auch mit Webstart verwendbar ist. Durch das Framework wird das MVC-Konzept bereits umgesetzt und es entfällt somit die zeitaufwändige Modellanbindung. Das Framework verwendet außerdem eine einheitliche Modellimplementierung für alle UI-Komponenten wie Tabellen, Tree, Editoren, Bildanzeige, Checkboxen u. v. m. Da es sich um ein aktives, generisches Modell handelt, müssen die UI-Komponenten die Aktualität der angezeigten Daten nicht mehr selbst berücksichtigen, denn das Modell informiert bei Bedarf.

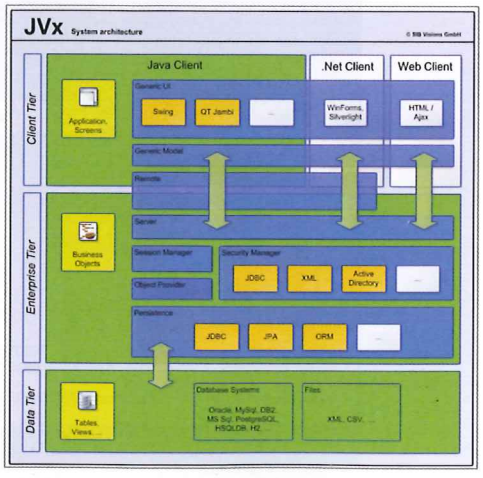

Abb. 1: Überblick der JVx-Architektur

Der Server von JVx ist unabhängig vom Kommunikationsprotokoll und modular aufgebaut, bietet ein speicheroptimiertes Session Handling und übernimmt die Verwaltung aller Businessobjekte bzw. die Businesslogik. Aufgrund von austauschbaren Authentifizierungsverfahren sind verschiedenste Authentifizierungswege umsetzbar. Das Framework enthält bereits Implementierungen für die Verwendung mit Datenbanken und XML. Doch es spricht auch nichts gegen die Anmeldung an ein Active Directory (eine übliche SSO-Lösung in Unternehmen). Als weiteres Extra gibt es den Zugriff auf beliebige Daten (XML, CSV, Sockets usw.) und Datenbanken (Oracle, MySql, Derby usw.) in Form einer Persistence-Implementierung. Abbildung 1 gibt einen Überblick der JVx-Architektur und legt zugleich die Architektur der Tippspielanwendung fest.

Beim User Interface entschieden wir uns für Swing, weil damit die Anwendung sowohl als Web-Start-Applikation als auch im Browser als Applet eingesetzt werden kann. Bei Bedarf könnte das UI auch gegen eine andere Implementierung ausgetauscht werden. Die komplette Businesslogik wird am Server in Form von Java-Objekten bereitgestellt. Die Kommunikation zwischen Client und Server erfolgt über HTTP(S) und wird vom Framework bereits abstrahiert. Das tatsächliche Kommunikationsprotokoll ist somit transparent und kann gegebenenfalls getauscht werden, wie z. B. RMI.

Für den Zugriff auf die Datenbank, im speziellen Derby, verwendeten wir das Persistence-API von JVx, weil es einen minimalen Speicherverbrauch garantiert und besonders effizient arbeitet. Nachdem die Technologie geklärt war, mussten die Rahmenbedingungen für die Umsetzung definiert werden. Dabei gab es nur hinsichtlich der Projektdokumentation Abstriche. Folgende Tools kamen zum Einsatz:

- Das firmenübliche Ticketing-System auf Basis von FlySpray für die Definition der Anforderungen
- Versionierung mit Subversion
- Eclipse als einheitliche Entwicklungsumgebung
- JUnit, Subclipse, Checkstyle, FindBugs, Emma als Eclipse-Plug-ins für die Qualitätssicherung
- Derby-UI und QuantumDB als Eclipse-Plug-ins für die Bearbeitung der Derby-Datenbank
- PlSql Developer für die Bearbeitung der Oracle-Datenbank
- Ant für den automatischen Build

Im nächsten Schritt definierten wir im Team die Anforderung an das Tippspiel und organisierten uns selbstständig, sprich, jeder übernahm die Verantwortung für einen Bereich der Entwicklung. Dazu zählten Datenbankdesign, Businesslogik, Layout, Dokumentation, Qualitätssicherung und der Build-Prozess. Auf diese Art und Weise gab es kaum Überschneidungen und wir verloren keine Zeit. Das Um und Auf dabei war, dass sich jeder auf den anderen verlassen konnte.

Von Beginn an war für uns klar, dass wir nicht nur ein Tippspiel für die WM 2010 entwickeln wollten, sondern ein Tippspiel für alle möglichen Fußball-Events wie Champions League, UEFA-Cup usw. Außerdem mussten einfach gewisse Extras wie private Tippgruppen, Tipp auf den Turniersieger, Benutzeradministration mit Rollenverwaltung, aber auch Mehrsprachigkeit im Umfang enthalten sein. Als Krönung sollte das System auch noch vollautomatisch arbeiten. Einfach ein neues Turnier erstellen, die Mannschaften in Gruppen einteilen, den allgemeinen Spielplan festlegen und die Ergebnisse der einzelnen Partien eingeben. Den Rest übernimmt das System.

Es ging uns also nicht nur darum zu zeigen, wie effizient und komfortabel mit JVx entwickelt werden kann, sondern wie mit JVx komplexe und alltägliche Anforderungen innerhalb kürzester Zeit umgesetzt werden können.

Die Implementierung starteten wir mit dem Ziel, insgesamt zwölf Masken zu erstellen, wobei ein registrierter Benutzer nur sieben davon verwenden darf. Die restlichen Masken dienen ausschließlich der Administration und werden über die Rollenverwaltung an Administratoren vergeben. Eines bereits vorweg: Wir hätten uns mit weniger zufrieden geben sollen, denn es wurde so richtig knapp. Doch später mehr dazu.

Effizienz und Einfachheit

Da es sich bei JVx um ein relativ neues Framework handelt, werde ich anhand der Tippmaske zeigen, wie effizient man damit arbeiten kann (Abb. 2).

Die Maske stellt drei unterschiedliche Bereiche dar. Im oberen Bereich werden benutzerbezogene Informationen wie

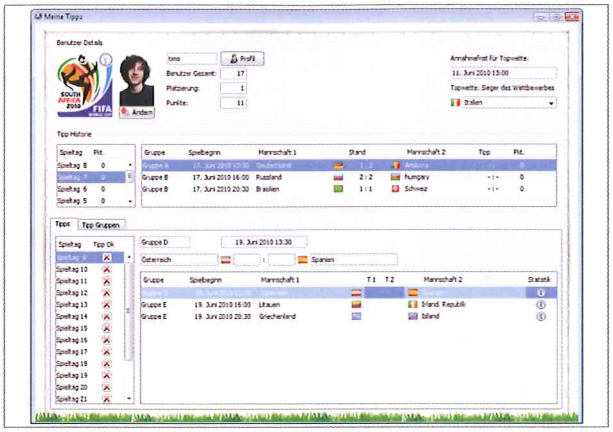

Abb. 2: Tippmaske

Foto und persönlicher Turniersieger dargestellt. Im mittleren Bereich findet man, abhängig vom gewählten Spieltag, alle Tipps und Ergebnisse für Spiele, die bereits gespielt wurden. Im unteren Bereich können die ausstehenden Spiele getippt werden. Um dem Benutzer die Änderung des Fotos zu ermöglichen, wird einerseits ein Button benötigt, der beim Klick die Dateiauswahl öffnet, und andererseits ein Objekt, in dem wir das Foto ablegen können:

```
//Serververbindung
rdbUserDefaults = new RemoteDataBook();
rdbUserDefaults.setDataSource(dataSource);
rdbUserDefaults.setName("userDefaults");
rdbUserDefaults.open();

//Buttondefinition
UIButton butImageUpload = new UIButton("Change");
butImageUpload.setImage(UIImage.getImage(UIImage.IMPORT_SMALL));
butImageUpload.eventAction().addListener(this, "doUpload");
```

Das *RemoteDataBook* kümmert sich um die Übertragung der Daten zwischen Client und Server. Durch den Namen findet sich am Server das passende Businessobjekt wieder, das die notwendigen Benutzerdaten bereitstellt. An der Definition des Buttons ist

Abb. 3: Android-Anwendung

zu erkennen, dass ein spezielles Action Handling verwendet wird. Es wird einfach definiert, welche Methode beim Klick aufgerufen werden soll (natürlich werden auch klassische Action Handler unterstützt). Nun fehlt noch die Anzeige der Dateiauswahl:

```
//Dateiauswahl
public void doUpload() throws Throwable
{
 //Nach erfolgter Auswahl wird "storeFile" aufgerufen
 getApplication().getLauncher().getFileHandle
            (this, "storeFile", "User Image");
}
```

sowie die Speicherung des Fotos:

```
public void storeFile(IFileHandle pFileHandle) throws Throwable
{
 //Speichert das Bild in der Businesslogik
 rdbUserDefaults.setValue
            ("IMAGE", FileUtil.getContent(pFileHandle.getInputStream()));
 rdbUserDefaults.saveSelectedRow();
}
```

Für den Beweis der Einfachheit des Frameworks definieren wir die Abhängigkeit zwischen Spieltag und Spiel aus dem unteren Bereich der Maske. Dafür benötigen wir je ein Objekt für die Daten der Spieltage und der einzelnen Spiele.

```
//Spieltage
RemoteDataBook rdbMatchdays = new RemoteDataBook();
rdbMatchdays.setDataSource(dataSource);
rdbMatchdays.setName("matchdays");
rdbMatchdays.setReadOnly(true);
rdbMatchdays.open();

//Alle ausstehenden Spiele
RemoteDataBook rdbBets = new RemoteDataBook();
rdbBets.setDataSource(dataSource);
rdbBets.setName("bets");
rdbBets.setMasterReference
 (new ReferenceDefinition
  (new String[] {"MADA_ID"}, rdbMatchdays, new String[] {"ID"}));
rdbBets.open();
```

Wir erstellen ein *RemoteDataBook* für die Spieltage und ein weiteres für die ausstehenden Spiele. Um zu definieren, dass bei jedem Spieltagwechsel auch die dazugehörigen Spiele angezeigt werden, stellen wir eine Beziehung zwischen den beiden DataBooks her. Diese Beziehung ist die logische UI-Sichtweise und kann, muss aber nicht, mit der Beziehung im Datenmodell übereinstimmen. Um letztendlich die Daten auch anzuzeigen, sind folgende Zeilen notwendig:

```
UITable tabMatchdays = new UITable();
tabMatchdays.setDataBook(rdbMatchdays);

UITable tabBets = new UITable();
tabBets.setDataBook(rdbBets);
```

Die Tabellen müssten nun noch zum Layout hinzugefügt werden, doch darauf verzichten wir an dieser Stelle.

Lessons Learned

Nun ist es an der Zeit zu erklären, warum wir uns mit weniger hätten zufriedengeben sollen. Alle Teamitglieder haben ausreichend Erfahrungen im Umgang mit Datenbanken, doch leider nicht mit Derby und vor allem nicht mit den Tuningoptionen der Datenbank. Wir kennen das Framework in- und auswendig und wussten ungefähr, wie lange wir für die Umsetzung aller Anforderungen brauchen würden. Der Build-Prozess war auch nicht mehr Aufwand als geplant. Aber die Datenbank machte uns etwas Probleme, weil wir zuerst mit einer Oracle XE entwickelten und danach auf die Derby-Datenbank wechseln wollten.

Warum Oracle? Um zu demonstrieren, dass die JVx-Datenbank unabhängig ist, dass die Anwendung auch unter Volllast mit unterschiedlichsten Turnieren noch effizient läuft und um eine Livevorschau des aktuellen Punktestands auf einfachste Art und Weise zu ermöglichen.

Wir kamen zu dem Ergebnis, dass man Views mit Derby nicht ausreizen sollte und die Sortierung in Views einfach (noch) nicht unterstützt wird. Dass Oracle bei manchen Operationen einfach um zwei bis drei Sekunden schneller arbeitet als Derby, ließen wir dann aber auch ungeklärt. Aber letztendlich ging doch alles ganz gut über die Bühne und wir haben die Anwendung mit einer Fülle an Funktionen ausgestattet – und das in respektabler Zeit. Der Aufwand, den wir investierten, lag bei 18 Personentagen in einem Zeitraum von 21 Tagen mit drei Personen, und wir produzierten:

- 4,6k Zeilen Code (alle Schichten eingerechnet, ohne Ausnahme)
- 2,4k Kommentarzeilen
- Eine vollständig integrierte Onlinehilfe
- Dokumentation des Datenmodells (ohne konnten wir einfach nicht abgeben)

Zum Zeitpunkt der Challenge war die Anbindung an mobile Clients noch nicht umgesetzt war, die Android-Anwendung wurde vollständig mit JVx und den JVx Android AddOns umgesetzt. Das Team „Packung!“ plant bereits den Einsatz des Tippspiels zur WM in Südafrika und freut sich auf viele Mitstreiter. ■

René Jahn ist Mitbegründer der SIB Visions GmbH aus Österreich und Head of Research & Development. Er ist außerdem Softwareentwickler und zuständig für das Qualitätsmanagement.