Easy Rider

Client-/Server-Applikationen mit dem Aviantes-Framework

Frank Merfort

Die Erstellung von professionellen Client-/Server-Applikationen ist in der Regel ein sehr mühseliger und zeitaufwendiger Prozess, vor allem dann, wenn so eine Applikation zum ersten Mal von einem Entwicklerteam implementiert werden soll. Zwar gibt es sehr viele Java-Frameworks, die bei der Implementierung helfen, doch decken diese oft nur einen kleinen Teilbereich einer kompletten Applikation ab. Das Aviantes-Framework verfolgt hier einen anderen Ansatz und stellt Entwicklern eine grundlegende und voll funktionsfähige Basisapplikation zur Verfügung. Ziel ist es, die Entwicklung zu vereinfachen und zu beschleunigen, ohne die Entwickler zu sehr einzuschränken. Dazu werden Module für die fachspezifische Logik sowohl für die Server- als auch die Client-Seite erstellt und in das Framework integriert. Die oft benötigten Standard-CRUD-Module werden dabei durch XML-Dateien konfiguriert und sind somit sehr schnell verfügbar.

Einführung

Das Java-Framework Aviantes verfolgt drei wesentliche Ziele:

- die einfache, schnelle und damit auch kostengünstige Erstellung von Client-/Server-Applikationen,
- Berücksichtigung der Bedürfnisse und Wünsche der Entwickler bzw. Kunden durch flexible Anpassungs- und Erweiterungsmöglichkeiten,
- die effiziente Nutzung der vorhandenen Ressourcen und damit Sicherstellung einer zügig ablaufenden Applikation.

Viele Konfigurationsmöglichkeiten (hauptsächlich durch im Klassenpfad liegende XML-Dateien) sorgen für eine rasche Erstellung einer Basisapplikation. Weitere Anpassungen und Erweiterungen werden darauf aufbauend in Java implementiert. Auch Kernbestandteile des Frameworks können ausgetauscht bzw. erweitert werden. Beispielsweise könnte das Logging von einem anderen Framework wie z. B. log4j übernommen werden [LOG4J].

Architekturüberblick

Das Framework setzt mindestens Java 1.5 SE voraus und stellt alle grundlegenden Bereiche einer Client-/Server-Applikation zur Verfügung. Im Einzelnen sind das:

- Anbindung und Einrichtung der Datenbank(en),
- Server-Modul-Verwaltung (Geschäftslogik),
- Client-Autorisierung und Session-Verwaltung,
- Konfiguration von Server und Clients,
- Logging,
- Sperrmechanismen,
- Client-/Server-Kommunikation,
- Rich Client auf der Basis von Swing,
- Benutzer-, Rollen-, Rechte-, Menü- und Mandantenverwaltung,
- Client-Modul-Verwaltung,
- XML-Modul-Framework.

Als Datenbanken können zurzeit Oracle, Microsoft SQL-Server, MySQL und PostgreSQL verwendet werden. Bei Bedarf ist eine Erweiterung für andere Datenbanken durch die Erstellung entsprechender Adapter-Klassen möglich.

Server

Die Server-Komponente besitzt keine Oberfläche. Wrapper-Klassen sorgen für eine Einbettung in verschiedene Laufzeitumgebungen. Eine Implementierung für einen Servlet-Container wie z. B. dem Tomcat [TOMCAT] ist dabei im Framework enthalten.

Der Server übernimmt die Anbindung und Einrichtung der Datenbank(en) für die Applikation. Es gibt dabei eine Systemdatenbank und, wenn gewünscht, mehrere zusätzliche Mandantendatenbanken. XML-Dateien beschreiben den Aufbau des Datenbankschemas.

Listing 1 zeigt einen Ausschnitt einer solchen Datei mit der Definition einer Tabelle zur Speicherung von Artikeln. Die Angaben dürften weitgehend selbsterklärend sein. Es wird eine Tabelle mit mehreren Spalten und einem Primärschlüssel definiert. Weiter können in so einer XML-Datei auch Indizes, Fremdschlüssel, Views und auszuführende SQL-Anweisungen definiert werden. Eigene Erweiterungsklassen können in den Ablauf des Datenbankaufbaus eingreifen.

```
<table name="wwArtikel" description="Artikel">
  <column name="nNr" type="int" nullable="false"/>
  <column name="sArtikelnummer"
          type="varchar" length="16" nullable="true"/>
  <column name="sBezeichnung"
          type="varchar" length="32" nullable="false"/>
  <column name="nArtikelgruppeNr" type="int" nullable="false"/>
  <column name="fPreis" type="float" nullable="true"/>
  <column name="fIstmenge" type="double" nullable="true"/>
  <column name="dtAufnahme" type="datetime" nullable="true"/>
  ...
  <primaryKey name="pk_wwArtikel">
```

```
      <column name="nNr"/>
    </primaryKey>
  </table>
```

Listing 1: Beschreibung einer Datenbanktabelle

Der Server überprüft bei jedem Start, ob sich die Datenbankstruktur verändert hat. Während der Datenbankeinrichtung können Datensätze aus Textdateien in die Tabellen importiert werden. Ein Administrator ist in der Lage, über ein Bearbeitungsmodul weitere Mandanten während der Laufzeit hinzuzufügen. Diese sind somit ohne Unterbrechung des Betriebs sofort für Anwender verfügbar.

Der Server verwaltet die von Entwicklern zu erstellenden Server-Module. Diese implementieren die Geschäftslogik der Applikation und dienen im Wesentlichen als Schnittstelle zwischen den Clients und der Datenbank. Die Clients haben über das Netzwerk Zugriff auf diese Module und rufen Methoden dieser Klassen auf.

Client

Der Client basiert auf Swing und lässt sich z. B. über Java-Webstart starten. Jeder Anwender muss sich zunächst mit Benutzername und Kennwort anmelden. Die Benutzerverwaltung ist rollenbasiert, d. h. jedem Benutzer werden Rollen zugewiesen, die wiederum bestimmte Rechte besitzen. Anhand dieser Rechte erfolgt eine Festlegung darüber, welche Module der angemeldete Anwender öffnen darf und welche Aktionen sich in diesen Modulen ausführen lassen (z. B. Daten einsehen ja, aber verändern nicht).

Ebenso wie der Server ist auch der Client modular aufgebaut. Ein Menü enthält alle Module, die der Anwender öffnen darf. Über schon im Framework existierende Administrationsmodule erfolgt eine Verwaltung der Benutzer, Rollen, Rechte, Menüstruktur und Mandanten.

Es gibt verschiedene Darstellungsvarianten für geöffnete Module:

- als Karteireiter innerhalb eines Fensters,
- als SDI-Anwendung mit einzelnen unabhängigen Fenstern für jedes Modul,
- als MDI-Anwendung mit einzelnen Unterfenstern innerhalb eines übergeordneten Client-Fensters (s. Abb. 1).

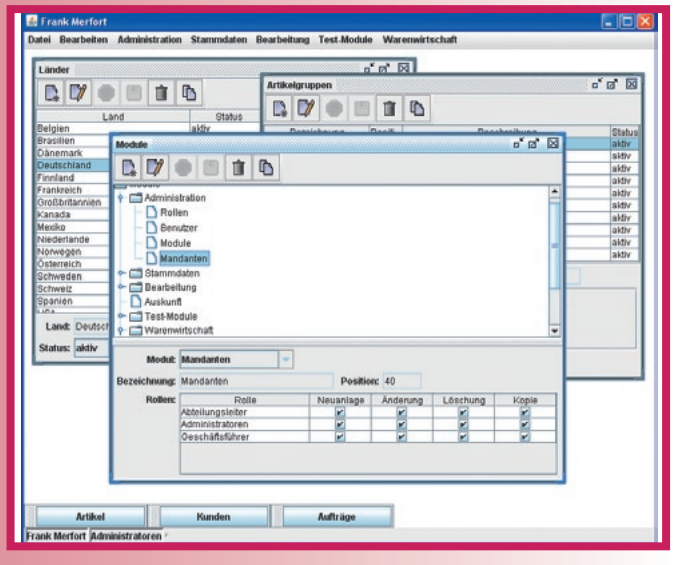

Abb. 1: Client als MDI-Anwendung

Jedes Modul besteht aus einer oder mehreren Bearbeitungsmasken, die in Form von Karteireitern angeordnet sind. Oft bestehen dabei Abhängigkeiten zwischen den Masken eines Moduls.

Entwickler haben vielfältige Möglichkeiten, das Aussehen und die Funktionalität des Clients anzupassen. So ist es z. B. möglich,

- eigene Icons für die Schaltflächen festzulegen,
- eine Lokalisierung in beliebige Sprachen vorzunehmen,
- die Tastaturkürzel für Aktionen vorzugeben,
- die Farben und Schriften anzupassen,
- weitere Menüpunkte aufzunehmen,
- die Darstellungsweise der Module anders zu gestalten
- usw.

Kommunikation

Die Abwicklung der Kommunikation zwischen den Clients und dem Server erfolgt ähnlich wie bei RMI (Remote Method Invocation). Allerdings verwendet das Framework nicht die RMI-Klassen von Java, sondern implementiert die Kommunikation komplett selbst. Dadurch werden einige Nachteile von RMI vermieden. Das Framework benötigt keine RMI-Registry und auch keine speziellen Port-Freigaben, wie sie für RMI notwendig sind. Die Erfahrung hat gezeigt, dass hier bei Kunden, die später so eine Anwendung verwenden möchten, oft sehr restriktive Firewall-Einstellungen vorhanden sind. Die Kommunikation des Frameworks benötigt deswegen nur einen einzigen auf dem Server freigegebenen Port (z. B. Port 80).

Für einen Entwickler gestaltet sich die Anwendung aber vollkommen analog zu RMI. Es beginnt mit der Definition eines Interfaces mit den von den Clients benötigten Methoden.

Die Implementierung dieses Interfaces ergibt das Server-Modul. Jedes beliebige Client-Modul hat die Möglichkeit, eine Instanz dieses Interfaces über eine öffentliche Client-Methode anzufordern und anschließend deren Methoden aufzurufen. Alles Weitere erledigt das Framework.

Wie in vielen anderen Bereichen des Frameworks auch kann ein Entwickler in den Kommunikationsvorgang eingreifen und so spezielle Anpassungen und Erweiterungen vornehmen, beispielsweise um den Datenverkehr zwischen Client und Server komplett zu verschlüsseln.

XML-Module und -Panels

Sehr oft werden in einer Applikation Bearbeitungsmasken (Panel) benötigt, die auf der CRUD-Funktionalität basieren (neue Datensätze erstellen, bestehende suchen, verändern und löschen, also CRUD=Create, *R*ead, *U*pdate, *D*elete). Das Framework bietet hierfür die Möglichkeit, diese Masken sehr einfach alleine durch die Erstellung von XML-Dateien zu erzeugen. Diese XML-Dateien legen zum einen fest, welche Widgets (Komponenten) wo auf der Maske platziert werden sollen. Zum anderen erfolgt auch eine direkte Verknüpfung der Widgets mit Feldern in Tabellen der Datenbank (Data-Binding).

Listing 2 zeigt dies an einem Ausschnitt einer XML-Datei für ein Panel. Dieser Ausschnitt definiert ein Label und ein einzeiliges Texteingabefeld, die sich nebeneinander auf dem Panel befinden sollen. Das constraints-Attribut legt die Platzierung der Widgets in dem übergeordneten Container (JPanel) fest. Hier im Beispiel geben die Daten die Parameter für ein `Grid-`

```
<label
  text="Name:"
  constraints="3 0 1 1 0.0 0.0 EAST NONE 5 5 0 0 0 0"
/>
<textField
  id="idName"
  name="Name"
  toolTipText="Name"
  constraints="4 +0 2 1 0.0 0.0 WEST NONE 5 5 0 0 0 0"
  obligatory="true"
  size="227 21"
  field="sName"
  type="STRING"
/>
```

Listing 2: Definition von Widgets

BagConstraints-Objekt an, liefern also Zeile, Spalte, Ausdehnung, Ausrichtung usw. für einen **GridBagLayout**-Manager.

Das Texteingabefeld erhält eine ID, über die später ein Entwickler im Quellcode auf dieses Widget und damit auch auf das dazugehörige **JTextField**-Objekt zugreifen kann, falls dies erforderlich ist. Der Name ist wichtig für eventuelle Fehlermeldungen an den Anwender, z. B. wenn wie hier in dem Textfeld eine Eingabe verpflichtend ist (Attribut **obligatory**), der Anwender das Feld aber leer gelassen hat. Die Attribute **field** und **type** bestimmen das Datenbankfeld in der Tabelle und den dazugehörigen Datentyp (wichtig für Überprüfungen der Eingabe).

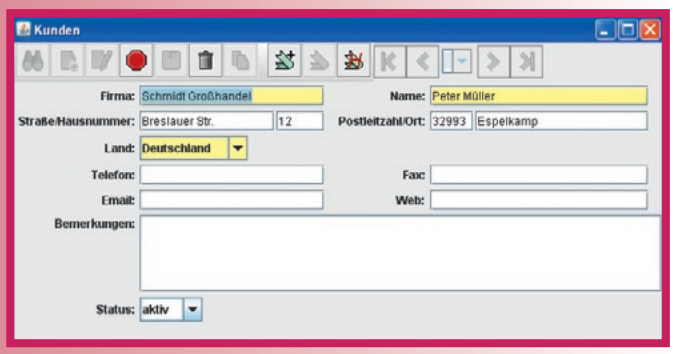

Abb. 3: Such-Dialog

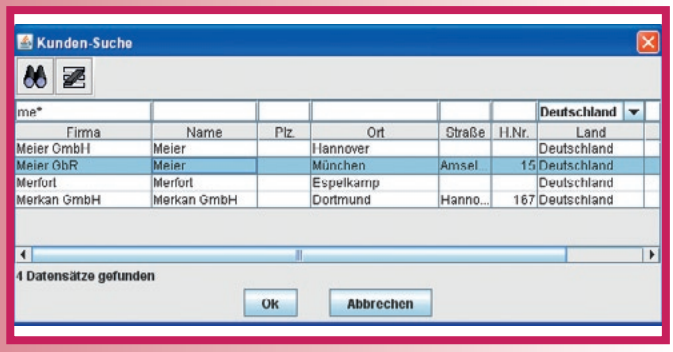

Abb. 3: Such-Dialog

Alle Texte in diesen XML-Dateien, die ein Anwender zu sehen bekommt, sind über verknüpfte Properties-Dateien lokalisierbar.

Im günstigsten Fall ist durch die Anlage einer solchen XML-Datei die Erstellung einer voll funktionsfähigen Bearbeitungsmaske abgeschlossen. Sind Besonderheiten erforderlich, so erfolgt darauf aufbauend eine Erweiterung durch Java-Code in einer von der Framework-Klasse abgeleiteten eigenen Klasse. Hier stehen dem Entwickler nun alle Möglichkeiten offen, da ein Zugriff auf sämtliche Komponenten und Abläufe des Panels besteht.

Zurzeit gibt es zwei verschiedene Varianten von Bearbeitungsmasken. Die erste Panel-Variante (s. Abb. 1) zeigt im oberen Teil des Panels eine Übersicht über alle vorhandenen Datensätze (z. B. in Form einer Tabelle oder als Baum). Klickt der Anwender auf einen Datensatz, so zeigt das Panel die dazugehörigen Daten im unteren Bereich an und ermöglicht so die Bearbeitung des ausgewählten Datensatzes.

Die zweite Panel-Variante (s. Abb. 2) zeigt dagegen immer nur einen Datensatz an. Über einen Such-Dialog (s. Abb. 3) sucht der Anwender nach Datensätzen mit bestimmten Kriterien und wählt aus den Ergebnissen einen zur Bearbeitung aus. Es bietet auch die Möglichkeit einer Historie und Lesezeichenverwaltung, wenn der Entwickler dies so in der XML-Datei konfiguriert hat. Das funktioniert analog zu einem Web-Browser. In der Historie bekommt der Anwender eine Übersicht über die zuletzt aufgerufenen Datensätze zu sehen (jeweils sowohl pro Modul als auch global über alle Module). In der Lesezeichenverwaltung lassen sich weitere Daten wie Bemerkungen, Stichwörter oder auch ein Wiedervorlagetermin einem Datensatz zuordnen. In der lokalen bzw. globalen Lesezeichenübersicht hat der Anwender nun die Gelegenheit, nach eben diesen Angaben zu suchen und somit die so markierten Datensätze erneut zu öffnen.

Zwischen den Masken in einem Modul können Abhängigkeiten hergestellt werden (Master-Detail-Beziehungen). In dem ersten Panel wählt ein Anwender z. B. den Hauptdatensatz aus. Weitere Panels zeigen davon abhängige Datensätze an und gestatten deren Bearbeitung.

Auch innerhalb eines Panels ist eine Einrichtung von Beziehungen zwischen einzelnen Widgets realisierbar (Master-Slave-Widgets). Beispielsweise wählt der Anwender in einer Länder-Combobox ein Land aus, woraufhin in einer Städte-Combobox die Städte zu diesem Land erscheinen.

Sollten die im Framework vorhandenen Widgets nicht ausreichen, so lassen sich eigene Widgets in das Framework integrieren.

Möglicherweise werden in einer Applikation Masken benötigt, die einer ganz anderen Bearbeitungsphilosophie folgen müssen. In diesem Fall erstellt ein Entwickler eine neue Panel-Klasse, die diesen Vorgaben gerecht wird, aber dennoch die Fähigkeiten der Widgets und damit der einfachen Konfiguration nutzt.

Fazit

In diesem Artikel wurde das Aviantes-Java-Framework zur Entwicklung von Client-/Server-Applikationen vorgestellt. Es zeichnet sich dadurch aus, dass alle grundlegenden Bereiche einer solchen Applikation vom Framework zur Verfügung gestellt werden. Außerdem bietet es die Möglichkeit, Standard-Bearbeitungsmasken sehr schnell und damit auch kostengünstig zu erstellen.

Ganz gegen den aktuellen Trend wird als Client nicht ein Web-Browser verwendet, sondern ein Rich Client. Das mag für manche ein großer Nachteil des Frameworks sein. Allerdings ist es nach wie vor trotz vieler Web-Frameworks sehr mühselig und zeitintensiv, solch eine komplexe Anwendung für den Browser zu entwickeln. Schon alleine die unterschiedliche Darstellung in den verschiedenen Browsern kann einem hier das Leben schwer machen. Dabei stellt sich die Frage, ob es nicht sinnvoller ist, eine Applikation, die z. B. von Sachbear-

beitern tagtäglich genutzt wird, als klassischen Desktop-Client zur Verfügung zu stellen und nicht im Web-Browser. Sicher gibt es viele Anwendungen (z. B. Shops), die im Browser sehr viel besser aufgehoben sind, aber gilt das auch für alle Anwendungen?

Links

[AVIANTES] Informationen zum Aviantes-Java-Framework, `http://www.aviantes.de`
[LOG4J] Apache Logging Framework, `http://logging.apache.org/log4j/`
[TOMCAT] Apache Tomcat, `http://tomcat.apache.org`

Dipl.-Ing. Frank Merfort ist seit 15 Jahren im Bereich Softwareentwicklung tätig, davon in den letzten neun Jahren hauptsächlich im Java-Umfeld. Hierbei hat er sich vor allem mit den Themengebieten Client-/Server-Applikationen, Frameworks und Webapplikationen beschäftigt. Seit einem Jahr ist er selbstständig und entwickelt das Aviantes-Java-Framework.
E-Mail: frank.merfort@aviantes.de.