HANSER

Simon Widjaja

Rich Internet Applications mit Adobe Flex 3

ISBN-10: 3-446-41366-9
ISBN-13: 978-3-446-41366-5

Leseprobe

12

12 Umgang mit lokalen Daten

12.1 Einführung

Das Flex-Framework stellt Ihnen verschiedene Werkzeuge zur Verfügung, mit denen Benutzereingaben überprüft, die Ausgabe formatiert und Daten strukturiert und typsicher gehalten sowie zwischen verschiedenen Objekten bzw. Schichten ausgetauscht werden können. Dieses Kapitel widmet sich ausschließlich der Haltung, Darstellung und Verarbeitung lokaler Daten innerhalb der Anwendung. Wie Sie externe Daten (z.B. WebServices oder RemoteObjects) in die lokale Anwendung einbeziehen, erfahren Sie in Kapitel 13: „Externe Datenquellen“.

Zunächst möchte ich Ihnen erst einmal ein Grundverständnis für den Datenfluss einer typischen Flex-Anwendung vermitteln. Bevor wir uns den einzelnen Themenblöcken im Detail zuwenden, folgen eine grobe Übersicht der einzelnen Vorgänge sowie ein Datenflussdiagramm.

12.1.1 Daten aktuell halten: Datenbindung (Data Binding)

Das sogenannte *Data Binding* ist ein sehr zentraler Bestandteil im Umgang mit Daten innerhalb der Flex-Anwendung. Kurz gesagt, beschreibt Data Binding die Möglichkeit, auf einfache Weise Daten in Form von Objekteigenschaften auf ein anderes Objekt zu übertragen. Verändert sich also die Eigenschaft des einen Objekts, so wird die des zweiten Objekts informiert und dementsprechend angepasst. Da so eine Bindung zwischen den beiden Eigenschaften entsteht, spricht man hier von der Datenbindung (Data Binding). Diese Bindungen werden während der Entwicklung festgelegt, sodass sie zur Laufzeit automatisch angewandt werden können.

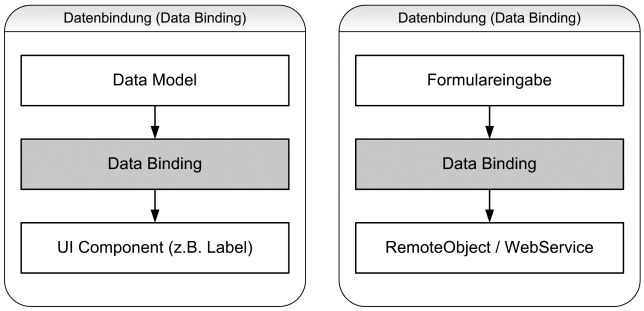

Abbildung 12.1 Beispiele für den Einsatz von Data Binding

Folgende konkrete Anwendungsbeispiele sind denkbar:

- Die Eigenschaft `value` eines HSliders soll automatisch den Wert der Eigenschaft `playheadTime` eines Video-Displays (Position des Abspielkopfes) übernehmen.
- Mehrere Eigenschaften sollen automatisch an ein und dieselbe Eigenschaft gebunden werden.
- Die Felder eines Formulars sollen automatisch die Werte eines Data Models übernehmen und somit vorausgefüllt werden.
- Benutzereingaben sollen automatisch an ein Data Model übergeben werden, um so an einen Webservice gesendet werden zu können.

Das nachstehende Beispiel zeigt eine der möglichen Arten, ein solches Data Binding zu realisieren. Das Label übernimmt automatisch die `text`-Eigenschaft des Eingabefeldes, und zwar immer dann, wenn sich diese verändert. Hierbei kommen die geschweiften Klammern ({}) zum Einsatz, welche die Inline-Schreibweise für Data Binding darstellen.

Listing 12.1 Ein einfaches Data Binding

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">

  <!-- Eingabe -->
  <mx:TextInput id="myName" />

  <!-- Gebundene Ausgabe -->
  <mx:Label text="{myName.text}" />

</mx:Application>
```

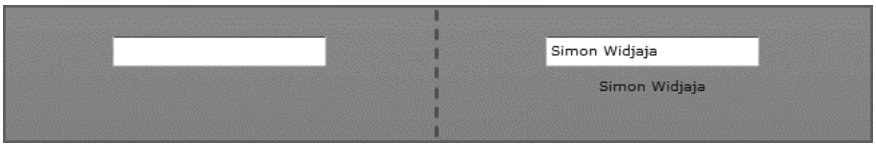

Abbildung 12.2 Data Binding: ohne und mit Eingabe

Was beim Data Binding genau hinter den Kulissen passiert und welche Möglichkeiten Ihnen zur Verfügung stehen, ein Data Binding zu realisieren, erfahren Sie in Abschnitt 12.2.: „Data Binding“.

12.1.2 Daten richtig strukturieren und bereitstellen: Data Model

Da Flex-Anwendungen nicht selten sehr datenlastig sind und die Komplexität der Daten schon den einen oder anderen Webentwickler an die Grenzen seines Können gebracht hat, spielen die Strukturierung und Modellierung von Daten eine sehr wichtige Rolle und sollten von Anfang an die notwendige Aufmerksamkeit erhalten. Werden bereits hier grundlegende Fehler begangen, hat dies meist mit fortschreitendem Entwicklungsstand fatale Folgen.

Das Flex-Framework bietet generell verschiedene Möglichkeiten, Daten strukturiert zu verwalten. Für Fortgeschrittene bzw. bei komplexen Aufgabenstellungen möchte ich bereits an dieser Stelle das Flex-Framework *Cairngorm* erwähnen. Dieses Framework für Mikroarchitektur ist eine Sammlung verschiedener, etablierter Design-Pattern, die u.a. sehr sinnvolle Mechanismen und Klassen bereitstellen, um eine gute Datenhaltung selbst bei umfangreichen und/oder modularen Projekten zu ermöglichen. Mehr zum Cairngorm Framework erfahren Sie in Kapitel 15.3: „Cairngorm“..

Das Data Model in Flex versetzt Sie in die Lage, Daten verschiedenster Art und Herkunft in Form von clientseitigen Objekten zwischenzuspeichern. Hierzu existieren mehrere ActionScript-Klassen, die jedoch nicht nur in ActionScript, sondern auch in MXML definiert werden können. Eine MXML-Variante ist `<mx:Model/>`. So stellt folgender Tag ein beispielhaftes Data Model für ein Benutzerkonto dar:

```
<mx:Model id="myBenutzerKonto">
  <benutzerKonto>
    <name/>
    <benutzerName/>
    <email/>
  </benutzerKonto>
</mx:Model>
```

Dieses Model kann also die zentrale Datenhaltung für Ihre Anwendung bilden.

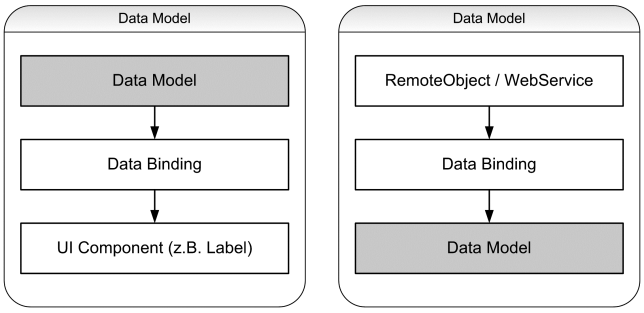

Abbildung 12.3 Beispiele für den Einsatz eines Datenmodells

In Kombination mit dem zuvor kennengelernten Data Binding können Sie nun bereits ein Formular und das dazugehörige Data Model erstellen und diese miteinander verknüpfen:

Listing 12.2 Data Model mit Data Binding

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">

  <!-- Data Model -->
  <mx:Model id="myBenutzerKonto">
    <benutzerKonto>
      <name>Simon Widjaja</name>
      <benutzerName>simonwidjaja</benutzerName>
      <email>mail@simonwidjaja.com</email>
    </benutzerKonto>
  </mx:Model>

  <!-- Formular -->
  <mx:Form>
    <mx:FormItem label="Name">
      <mx:TextInput text="{myBenutzerKonto.name}"/>
    </mx:FormItem>
    <mx:FormItem label="Benutzername">
      <mx:TextInput text="{myBenutzerKonto.benutzerName}"/>
    </mx:FormItem>
    <mx:FormItem label="EMail">
      <mx:TextInput text="{myBenutzerKonto.email}"/>
    </mx:FormItem>
  </mx:Form>

</mx:Application>
```

Abbildung 12.4 Einfaches Formular mit Data Model und Data Binding

Neben dem `<mx:Model/>`-Tag gibt es noch weitere Möglichkeiten, um ein Data Model abzubilden. Welche dies sind und welche Vor- und Nachteile die jeweiligen Varianten haben, erfahren Sie in Abschnitt 12.3: „Data Model“.

12.1.3 Daten aufbereiten und formatieren: Data Formatter

Für die Formatierung von Daten stellt Flex die *Data Formatter Components* bereit. Mit diesen können Sie Daten aufbereiten, bevor sie im User-Interface dargestellt werden. Häufige Anwendungsbeispiele sind:

- Darstellung eines Datums oder Timestamps
- Ausgabe einer Telefonnummer

- Darstellung von Dezimalzahlen mit Tausender-Separator (Punkt oder Komma)
- Anzeigen länderspezifischer Angaben (Währungen etc.)

Eine *Data Formatter Component* ist letztlich ein Objekt, das einen Wert im Rohzustand übergeben bekommt und einen formatierten Wert zurückgibt. Flex bietet neben den mitgelieferten Formatter-Komponenten, wie z.B. NumberFormatter, DateFormatter oder PhoneFormatter, auch die Möglichkeit, eigene Formatter-Klassen zu implementieren. Häufig kommen Formatter im Zusammenspiel mit dem zuvor vorgestellten Data Model und Data Binding zum Einsatz. Auf diese Weise können Werte in ihrem Rohzustand im Data Model abgelegt und dennoch sinnvoll visualisiert werden.

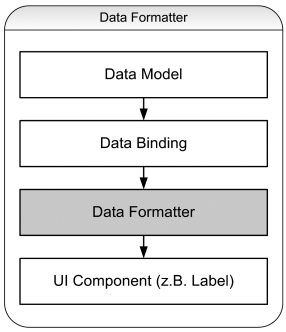

Abbildung 12.5 Beispiel für den Einsatz eines Data Formatters

Das folgende Beispiel zeigt, wie Sie ein formatiertes Datum in einem Label ausgeben:

```
<!-- Formatierung -->
<mx:DateFormatter id="myDateFormatter" formatString="DD.MM.YYYY"/>

<!-- Ausgabe des formatierten Datums -->
<mx:Label text="{myDateFormatter.format( new Date() )}"/>
```

Dieses Beispiel gibt statt des unformatierten `Fri Nov 30 02:23:29 GMT+0100 2007` die Zeichenkette `30.11.2007` aus.

Welche Formatter Flex genau bereitstellt und wie Sie Ihre eigene Formatter-Klasse schreiben, erfahren Sie in Abschnitt 12.4: „Data Formatting“.

12.1.4 Daten auf Gültigkeit überprüfen: Data Validation

Im Umgang mit Daten muss fast immer sichergestellt sein, dass einzelne Werte gewissen Vorgaben bzw. Regeln entsprechen. Besonders bei Benutzereingaben ist dies eigentlich stets notwendig. Natürlich kann die Prüfung der eingegebenen Daten auch ausschließlich serverseitig stattfinden. In den meisten Fällen ist es jedoch sinnvoll, die Eingaben direkt auf der Clientseite zu überprüfen. Dies schmälert zum einen den entstehenden Traffic und reduziert die Serverlast, und zum anderen kann der Benutzer auf diese Weise unmittelbar über seine fehlerhaften Eingaben informiert werden.

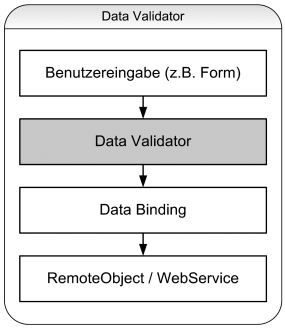

Abbildung 12.6 Beispiel für den Einsatz eines Data Validators

Typische Fälle, bei denen eine Validierung der Daten notwendig ist:

- E-Mail-Adressen
- Datumsangaben (z.B. Geburtsdatum)
- Alter (z.B. nur gültige Werte von 18–99)
- Telefonnummern
- Postleitzahlen
- Adressen samt Hausnummern
- Kreditkartennummern

Für solche Standardfälle gibt es im Flex-Framework fertige Klassen, die leicht einzusetzen sind. Eine sehr flexible Lösung stellt der `RegExpValidator` dar, mit dem selbst definierte reguläre Ausdrücke zur Validierung verwendet werden können. Das nachstehende Beispiel zeigt ein Data Model, eine Eingabemöglichkeit für eine E-Mail sowie die dazugehörige Validierung.

Listing 12.3 Beispiel einer Data Validator-Klasse

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  layout="vertical">

  <!-- Data Model -->
  <mx:Model id="myModel">
    <root>
      <email>{myInput.text}</email>
    </root>
  </mx:Model>

  <!-- Validierung -->
  <mx:EmailValidator source="{myModel}" property="email"
    listener="{myInput}" trigger="{myInput}"/>

  <!-- Ausgabe -->
  <mx:Label text="Ihre EMail:"/>
  <mx:TextInput id="myInput"/>
  <mx:Button label="Prüfen!"/>

</mx:Application>
```

Und so sieht das Ergebnis der Prüfung bei Nichterfüllen aus:

Abbildung 12.7 Data Validator in Aktion

Eine Übersicht über die bereitgestellten Validator-Klassen sowie eine Anleitung zum Schreiben eigener Validator-Klassen finden Sie in Abschnitt 12.5: „Data Validation“.

12.1.5 Datenfluss

Nachdem Sie nun ein Grundverständnis für die Datenhaltung und -manipulation in Flex haben, soll das folgende Diagramm anhand eines kleinen Beispielszenarios das Zusammenspiel der einzelnen Phasen und somit den gesamten Datenfluss veranschaulichen.

Nehmen wir einmal an, es soll eine kleine Anwendung erstellt werden, die zwei Zustände hat: Im ersten Zustand soll der Benutzer über ein Texteingabefeld seine E-Mail eingeben. Im zweiten Zustand soll das Datum ausgegeben werden, an dem der jeweilige Benutzer sich das letzte Mal eingeloggt hat. Die Ausgabe kann beispielsweise in einem einfachen Label stattfinden. Die Auswertung, also das Auslesen des Datums zur jeweiligen E-Mail-Adresse, soll über einen Serveraufruf realisiert werden. Der Datenfluss könnte wie folgt aussehen:

- Die Benutzereingabe wird via Datenbindung an ein Datenmodell gebunden.
- Anschließend wird die Eingabe über einen Validator geprüft. In diesem Fall würde der E-Mail-Validator zum Einsatz kommen, um sicherzustellen, dass es sich bei der Eingabe um eine gültige E-Mail-Adresse handelt.
- Sofern der Validator grünes Licht gibt, werden die Daten wieder per Datenbindung beispielsweise in ein RemoteObject geschrieben, das schließlich als Request zum Server gesendet wird. Dies kann als HTTP-, SOAP- oder AMF-Request geschehen.
- Der Server startet seine Abfrageroutine und sendet die Antwort (Response) im selben Format zurück.
- Über einen Event-Handler des RemoteObjects wird die Antwort entgegengenommen, von wo aus sie über eine Datenbindung an das Datenmodell übergeben wird, um dort zwischengespeichert zu werden.
- Über eine erneute Datenbindung werden die Antwortdaten mithilfe eines Formatters aufbereitet (in unserem Fall kommt ein DateFormatter zum Einsatz), um letztendlich dem Benutzer präsentiert zu werden.

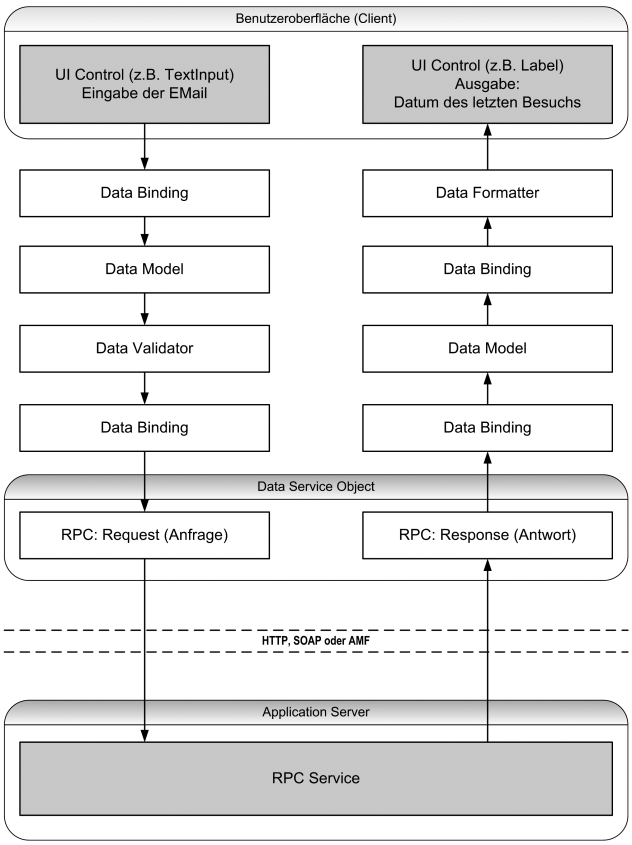

Abbildung 12.8 Typischer Datenfluss in einer Flex-Anwendung (Client/Server)

12.2 Data Binding

Mit Data Binding können Sie Objekteigenschaften verschiedener Objekte aneinander binden. Auf diese Weise kann der Datenfluss innerhalb der Anwendung stark vereinfacht werden, da Sie bei einer Änderung nicht selbst jedes Mal alle weiteren, von diesen Änderungen betroffenen Objekte informieren und diese ggf. aktualisieren müssen.

Für eine solche Datenbindung wird stets neben einer Ausgangseigenschaft und einer Zieleigenschaft zusätzlich noch ein auslösendes Event (Trigger) benötigt. Der Trigger beinhaltet die Information, zu welchem Zeitpunkt der Datenabgleich zwischen Ausgangseigenschaft und Zieleigenschaft stattfinden soll. Im Normalfall ist ein Trigger bereits implementiert und so eingestellt, dass dieser Datenabgleich sofort ausgeführt wird, sobald sich die Ausgangseigenschaft verändert. Selbstverständlich können Sie auch eigene Trigger definieren. Doch dazu später mehr.