10 Web Dynpro

Die Web-Dynpro-Technologie ist ein Bausatz für sämtliche Aspekte der Frontend-Entwicklung. Mit diesem Bausatz können Sie Oberflächen bis auf die Ebene von einzelnen Elementen wie Buttons oder Eingabefeldern definieren. Sie können die Elemente zu Masken zusammenfassen und die Navigationspfade zwischen den Masken festlegen. Sie können Oberflächenereignisse an Navigationsaktionen koppeln. Und schließlich können Sie die dahinter liegenden Daten strukturieren.

Der Backend-Zugriff ist nicht integraler Bestandteil von Web Dynpro. Dafür ist unter anderem das Java Dictionary zuständig. Dennoch sind die Frontend- und Backend-Technologien über das Netweaver Developer Studio nahtlos aneinander gekoppelt.

Wenn Sie ein Frontend als Web Dynpro modellieren, müssen Sie sich nicht mit den unangenehmen Details der Client-seitigen Zieltechnologie herumschlagen. Auch wenn ein Web Dynpro typischerweise auf dem Client-Rechner in einem Browser abläuft, müssen Sie keine Zeile JavaScript selbst schreiben. Als Folge können Sie aber die Wahl der Zieltechnologie auch nicht beeinflussen. Sie wird von der Entwicklungsumgebung diktiert. Außerdem haben Sie, ähnlich wie bei den Dynpro in ABAP, keinen Einfluss darauf, welcher Teil des Frontends auf dem Client und welcher auf dem Server ausgeführt wird. Das reduziert die Komplexität Ihrer Entwicklung deutlich und delegiert die technischen Details an eine ausgereifte Bibliothek.

Ihre Geschäftslogik müssen Sie aber nach wie vor selbst in Java codieren. Es ist also zum jetzigen Zeitpunkt nicht möglich, die graphisch entworfene Programmstruktur auf Knopfdruck nach ABAP oder C# zu portieren.

Um die Ausführungen überschaubar zu halten, wird in diesem und in den beiden folgenden Kapiteln für das Deployment nicht das JDI-konforme Verfahren beschrieben. Daher kommen beispielsweise Eclipse-Projekte anstelle von Development Components zum Einsatz. Der gesamte Bedienvorgang reduziert sich somit auf seine wesentlichen Bestandteile.

10.1 MVC und Web Dynpro

Es gibt kaum ein GUI-Framework, das sich nicht auf das MVC-Modell von Smalltalk beruft. Die Dreiteilung von Oberflächen in Model, View und Controller ist genauso verbreitet wie die Deutung der Begriffe variiert.

In Smalltalk steht das Model für den statischen Ablageort von Daten, also beispielsweise eine Terminliste. Der View stellt die Daten für den Benutzer dar, etwa in Form eines Kalenders mit bunt markierten Terminen. Der Controller wiederum führt Änderungen an den Daten im Model durch, in unserem Fall also Terminänderungen.

Die Hersteller von GUI-Frameworks bedienen sich gern dieses etablierten Vokabulars. Dadurch suggerieren sie dem Anwender von vornherein eine Vertrautheit mit dem Produkt. Allzu oft enden die Gemeinsamkeiten zum MVC-Konzept aber bei den Bezeichnungen, während die dahinter stehenden Sachverhalte ganz andere sind. Daher ist in einem für Sie neuen Kontext gerade bei den Begriffen Model, View und Controller große Sorgfalt geboten.

Der Zusammenhang zwischen den ursprünglichen MVC-Elementen und den zentralen Bausteinen in der Web-Dynpro-Architektur ist in der folgenden Tabelle aufgeführt.

Tabelle 10.1. Gegenüberstellung von Web-Dynpro-Begriff und Bedeutung aus MVC-Sicht

Web-Dynpro-Begriff	Bedeutung
View	Zusammenhängender Ausschnitt einer Maske, durchaus mit dem MVC-View vergleichbar
Context	Ablageort für Daten, existiert in einer View-spezifischen und einer globalen Ausprägung. Die globale Ausprägung ist mit dem MVC-Model vergleichbar. Man kann sich den Context aber auch wie einen Session Context eines Servlets vorstellen.
Controller	Verwaltet Events, existiert in einer View-spezifischen und einer globalen Ausprägung. Zusammengenommen sind die Controller mit dem MVC-Controller vergleichbar.

Das Zusammenspiel der Web-Dynpro-Bausteine aus Tabelle 10.1 ist trotz der verwirrenden Begrifflichkeit schnell beschrieben und durchaus einleuchtend.

Ein Web-Dynpro-View ist ein kleiner, logisch und gestalterisch zusammenhängender Anteil einer Seite. Der zuvor erwähnte Kalender-View trifft den Sachverhalt ganz gut. Auch eine feste Gruppe von Eingabefeldern für eine Adresse könnte als View realisiert werden. Jeder View verfügt über

einen View Context, der dessen Daten enthält. Außerdem werden Änderungen am View Context über den zugehörigen View Controller durchgeführt.

Mehrere Views inclusive ihrer Controller und Contexte werden zu einer Web Dynpro Component zusammengefasst. Die Web Dynpro Component ist keine Development Component, wie sie im vorigen Kapitel über den Entwicklungsprozess beschrieben wurde, sondern eine feinere und GUI-spezifische Gruppierung. Eine Web Dynpro Component enthält genau einen Component Controller. Dieser hat die Aufgabe, unterschiedliche Views bei Bedarf sichtbar oder unsichtbar zu machen und Events an sie weiterzuleiten. Stellvertretend für den View werden die Events von deren View Controller entgegengenommen. Außerdem enthält die Component einen Component Context, der die Nutzdaten der Component verwaltet. Die View-Kontexte sind immer nur Ausschnitte des Component Contexts.

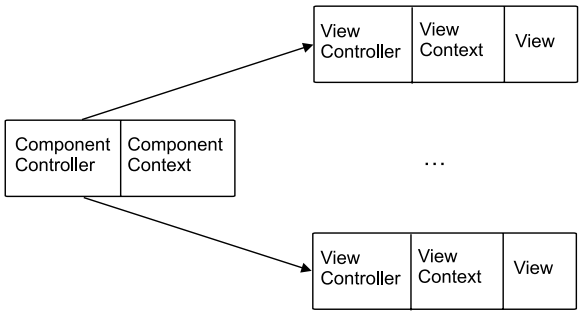

Abb. 10.1. Grobe Skizze des Zusammenspiels von Controller, Context und View

Bevor wir das Zusammenspiel von View, Context und Controller anhand der Entwicklungsumgebung genauer erörtern, wollen wir Sie noch auf die folgenden Ausdrücke aufmerksam machen. Sie ähneln den MVC-Ausdrücken, haben aber teils eine ganz andere, teils eine geringfügig andere Bedeutung. Wenn man darauf gefasst ist, kann man jedoch aus dem Zusammenhang erschließen, wie sie zu verstehen sind.

Tabelle 10.2. Begriffe, die auf irreführende Weise MVC-Termini ähneln

Begriff	Bedeutung
View	Auch die Entwicklungsumgebung Eclipse nutzt den Ausdruck View. Ein Eclipse-View ist ein spezialisierter Editor oder Wizard, der ein Panel der Eclipse- bzw. NWDS-Oberfläche einnimmt. Wir bemühen uns, diese Art von View nur in der Übersetzung Sicht oder in Kombination mit einem anderen Ausdruck wie Layout-

	View zu verwenden.
iView	Ein ganzes Web Dynpro, das portalfähig ist, wird als iView bezeichnet.
Model	Wenn im Zusammenhang mit Web Dynpro von Model die Rede ist, dann bezieht sich dies oft auf eine graphische Strukturbeschreibung eines Programms im Sinne der Model Driven Architecture.
Model Node	Unterstruktur im Context, die zur Anbindung eines Backend-Systems dient. Entspricht dem MVC-Model unter der Einschränkung, dass dieses über ein Backend befüllt wird.

10.2 Web Dynpro in der Entwicklungsumgebung

Wie sehen Views nun wirklich aus? Wie kann man sich anschaulich vorstellen, welche Aufgabe der Controller hat? Wenn man sich diese Objekte in der Entwicklungsumgebung anschaut, wird ihre Bedeutung klarer. Wir wollen dies anhand einer Beispielapplikation zum Verwalten von Terminen tun. Damit Sie sich vorstellen können, wie die Applikation funktioniert, zeigen wir vorweg das Endergebnis. Anschließend erläutern wir die Entwicklung der Applikation von Anfang an.

Auf der Einstiegsseite (s. Abb. 10.2) geben Sie an, für welche Kalenderwoche Sie sich Termine anzeigen lassen wollen. Über den Weiter-Button gelangen Sie auf die zweite Seite.

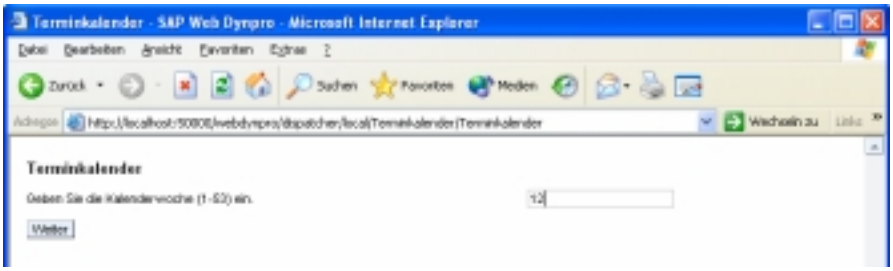

Abb. 10.2. Erste Seite der Beispielapplikation, Wochenauswahl

Die in Abb. 10.3 dargestellte zweite Seite enthält eine Übersicht über die vorhandenen Termine in der ausgewählten Woche. Über „Woche wechseln“ gelangen Sie zurück zur Einstiegsseite und können dort eine andere Kalenderwoche eingeben. Über „Ändern“ und „Einfügen“ gelangen Sie auf eine Seite, in der Sie einen einzelnen Termineintrag bearbeiten können. Der Löschen-Button entfernt lediglich den gegenwärtig selektierten Termin, verzweigt aber auf keine andere Seite.

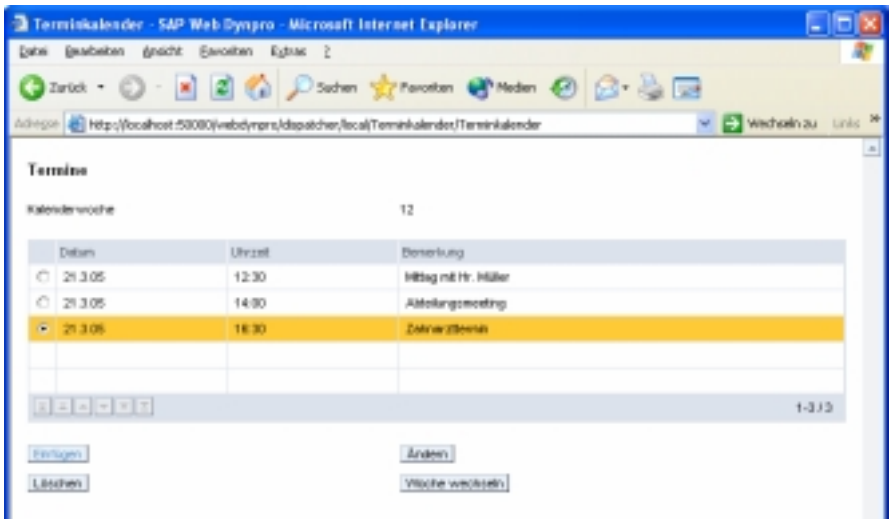

Abb. 10.3. Zweite Seite der Beispielapplikation, Terminübersicht

Die letzte Seite (s. Abb. 10.4) erlaubt es, die drei Anteile eines Termins − Datum, Uhrzeit und Bemerkung − zu ändern. Über „Zurück“ gelangen Sie auf Seite zwei.

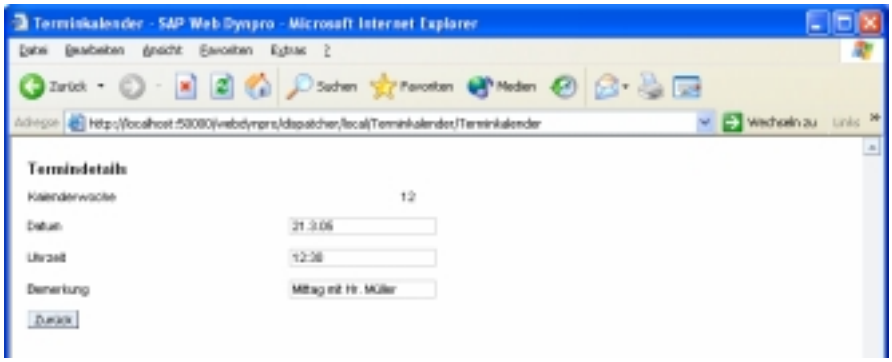

Abb. 10.4. Dritte Seite der Beispielapplikation, Termindetails

Views, Navigationspfade und Oberflächenereignisse

Sie beginnen ein neues Projekt mit dem Anlegen einer Component und deren Views. Das geschieht zunächst aus übergeordneter Sicht, so dass jeder View nur durch seinen Namen gekennzeichnet wird. Beim Anlegen von Component und Views wird automatisch der Code für deren Controller und Contexte generiert.

Einrichten von Navigationspfaden

Als nächstes können Sie Navigationsbeziehungen zwischen den Views aufbauen. Sie bilden die Voraussetzung für die Navigation von View zu View; auf welchem dieser Wege tatsächlich navigiert wird, ist dadurch allerdings noch nicht festgelegt.

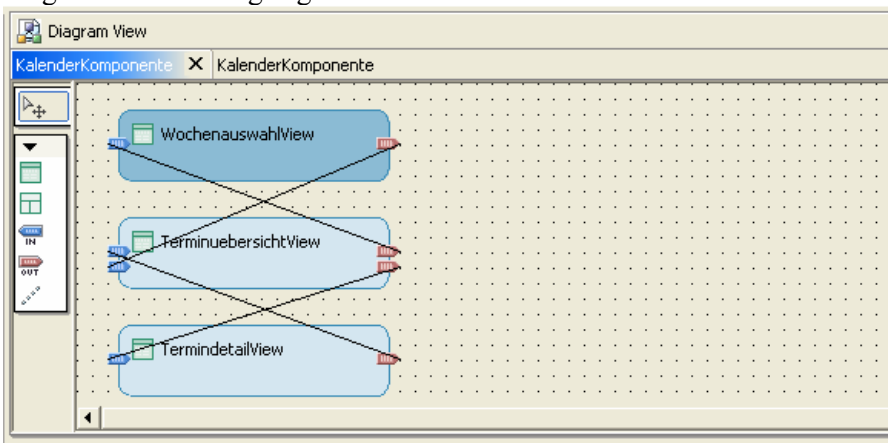

Abb. 10.5. Navigationsschema zwischen unterschiedlichen Views

In Abb. 10.5 sehen Sie die Views und Navigationspfade der Terminkalender-Applikation. Sie besteht aus dem `WochenauswahlView`, dem `TerminuebersichtView` und dem `TermindetailView`, die den weiter oben abgebildeten Bildschirmseiten 1-3 entsprechen. Die Navigation zwischen den Views soll in der Reihenfolge der Erwähnung, aber auch in der entgegengesetzten Richtung möglich sein. Die Richtung einer Navigationslinie ist an ihren roten und blauen Endpunkten erkennbar. Bei genauem Hinsehen sieht man, dass sie pfeilförmig sind.

Definieren von Oberfläche und Oberflächenereignissen

Um die Navigation zwischen unterschiedlichen Views tatsächlich nutzbar zu machen, muss man zweierlei definieren: eine benannte Aktion, die die Navigation durchführt und ein Oberflächenelement, das wiederum die Aktion anstößt. Beginnen wir mit dem Oberflächenelement.

Die Oberfläche eines Views lässt sich mit einem GUI-Editor zusammenklicken. Wer Erfahrung mit Swing hat, wird sich schnell damit zurechtfinden, denn das Vorgehen ist ähnlich. Sämtliche Oberflächenelemente sind in einer Hierarchie angeordnet. Sie werden also dem gesamten View zunächst beispielsweise ein `GridLayout` zuordnen. Dann fügen Sie die einzelnen Oberflächenelemente als Kinder dem View hinzu, wobei

Sie sich der gitterartigen Struktur bedienen, die das `GridLayout` bietet. In Abb. 10.6 ist die Hierarchie der Oberflächenelemente für den `WochenauswahlView` zu sehen. Beim Vergleich mit Abb. 10.2 können Sie die einzelnen sichtbaren Elemente wiederfinden.

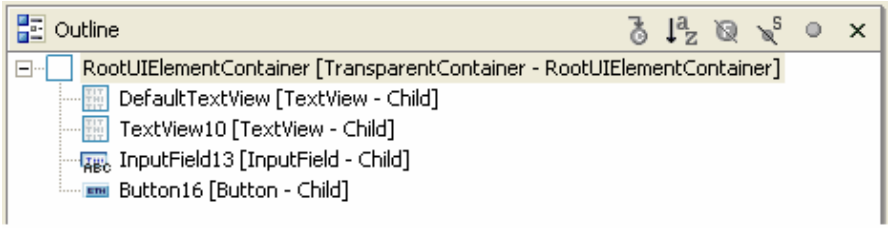

Abb. 10.6. Layout-Sicht für den `WochenauswahlView`

Bedenken Sie, dass Sie eine allgemeingültige Beschreibung der Oberfläche festlegen, die nicht auf ein bestimmtes Endgerät zugeschnitten ist. So wie der Inhalt eines Webbrowsers sich der Größe des Browserfensters anpasst, so flexibel verhält sich auch Ihre Applikation. Entsprechend dynamisch sind die Strukturen, durch die Sie Ihre Oberfläche definieren.

In Abb. 10.7 sehen Sie denselben View wie zuvor im View-Editor. Die Ansicht passt sich bei jeder Änderung in der Elementhierarchie automatisch an. Am unteren Rand des View-Editors befindet sich eine Reihe von Reitern. Gegenwärtig ist der Layout-Reiter ausgewählt.

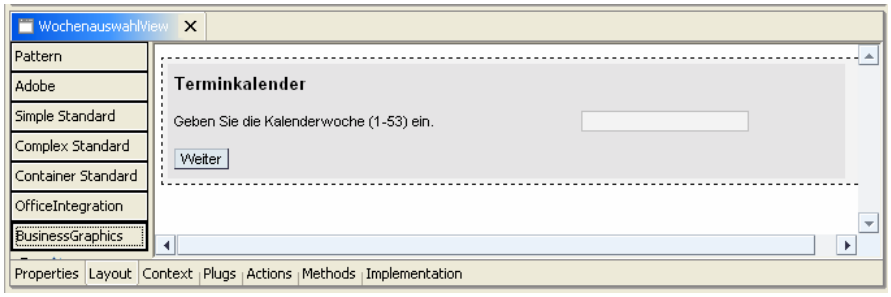

Abb. 10.7. Der `WochenauswahlView` im View-Editor

Über den Actions-Reiter können Sie die weiter oben erwähnten Aktionen zum Navigieren definieren. Für die Beispielaktion legen Sie an dieser Stelle eine Aktion namens `Weiter` an und koppeln sie an den Navigationspfad zum `TerminuebersichtView`. Anschließend ordnen Sie im Properties-Reiter der onAction-Eigenschaft des Buttons die soeben definierte Aktion zu, wie in Abb. 10.8 zu sehen. Nun ist der Navigationspfad

von dem `WochenauswahlView` zum `TerminuebersichtView` tatsächlich nutzbar.

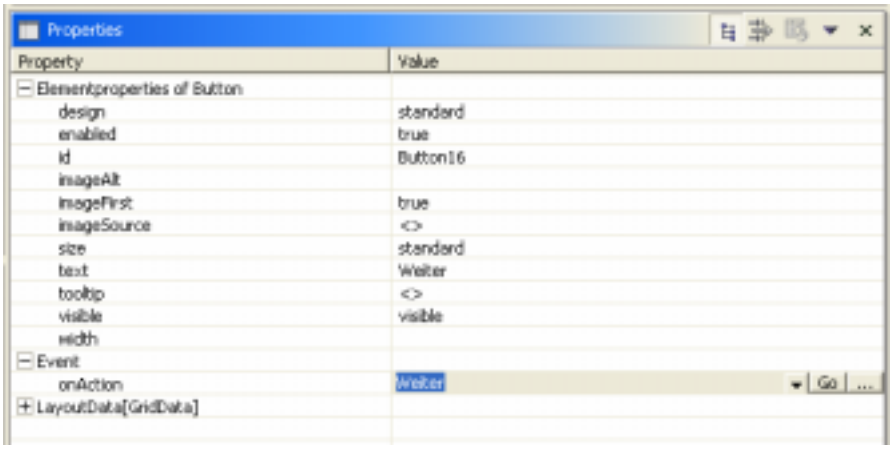

Abb. 10.8. Properties-Reiter des Weiter-Buttons

Wie hier beim Button-Element ist im Editor für jedes Element, das ein Ereignis auslösen kann, bereits eine Ereignismethode bereitgestellt. Beispielsweise können Buttons oder Eingabefelder die Ereignisse `onAction` bzw. `onEnter` auslösen. Einer Überschrift dagegen ist kein Ereignis zugeordnet.

Insgesamt fällt auf, dass mögliche Ereignisse äußerst sparsam angeboten werden. So gibt es für ein Eingabefeld wirklich nur dieses eine Ereignis. Aus anderen Umgebungen sind Sie vielleicht gewohnt, dass ein Eingabefeld auch dann ein Ereignis auslöst, wenn es den Fokus erhält oder in das Feld hineingeklickt wurde. Web Dynpro beschränkt sich hier nur auf das Allernötigste und stellt dadurch sicher, dass die Komplexität Ihrer Applikation beherrschbar bleibt und dass sie sich leicht auf eine andere Zieltechnologie übertragen ließe.

Implementierung der Ereignisbehandlung

Im Implementation-Reiter des View-Editors können Sie sich den generierten Code ansehen. Er entspricht dem View Controller, den wir zu Beginn dieses Kapitels skizziert haben. Lassen Sie sich nicht davon irritieren, dass die Controller-Klasse einen Namen trägt, der mit `View` endet. Jeder View-Controller wird durch eine einzelne Klasse implementiert, die den Namen des zugehörigen Views trägt. Den Code für den eigentlichen View bekommen Sie nicht zu Gesicht.

Für jedes Ereignis, das wir im Actions-Reiter definiert haben, enthält der View Controller eine Methode. Hier sehen Sie die Methode, die innerhalb des View Controllers des `TerminuebersichtView` zum `TermindetailView` verzweigt, nachdem der Einfügen-Button gedrückt wurde:

```
public class TerminuebersichtView {

// viel Code ausgelassen
...

  public void onActionEinfuegen(
    IWDCustomEvent wdEvent ) {
    //@@begin onActionEinfuegen(ServerEvent)
    wdThis.wdFirePlugToTermindetailView();
    //@@end
  }

// viel Code ausgelassen
...
}
```

Die Kommentarzeilen mit dem doppelten Klammeraffen sind Markierungen der Entwicklungsumgebung. Sie kennzeichnen den Codebereich, den man manuell abändern darf. An diese Stellen werden wir später den Code einfügen, der die Terminliste modifiziert. Damit wird der Controller seiner Aufgabe gerecht, Datenänderungen vorzunehmen. Gleichzeitig wird klar, dass der Zustandsautomat für die Navigation von einem View zum nächsten über sämtliche beteiligten View Controller verteilt ist. Dieser Zusammenhang sollte für Ihre Anschauung des View Controllers hilfreich sein.

Datenelemente

Wenden wir uns nun den Kontexten zu. Sie fungieren ja als Ablageort für die Daten, die an der Oberfläche angezeigt werden sollen. Das Web-Dynpro-Konzept trennt sorgfältig die Daten, die für die unterschiedlichen Views benötigt werden und die Daten, die komponentenweit genutzt werden. Erstere werden in den View-Kontexten abgelegt, letztere im Component Context. Zur Veranschaulichung sei an dieser Stelle noch einmal auf Abb. 10.1 verwiesen.

Bei der Entscheidung dafür, wo man ein Datenfeld ansiedelt, spielen zwei Gesichtspunkte eine Rolle. Einerseits soll ein View Context nicht mehr Daten enthalten, als für die Darstellung des Views notwendig sind. Andererseits müssen manche Felder zentral für die ganze Komponente ab-

gelegt werden, da sie für unterschiedliche Views benötigt werden und dort jeweils denselben Wert haben sollen. Um doppelte Datenhaltung zu vermeiden, werden in diesem Fall die Felder der View-Kontexte durch einen Verweis auf das jeweils entsprechende Feld im Component Context realisiert. Dieses Vorgehen nennt man Context Mapping.

Datenelemente des Terminkalenders

Am Beispiel des Terminkalenders lassen sich die Eigenheiten des Context-Konzepts gut untersuchen. Die Applikation benötigt zwei unterschiedliche Informationen: die Nummer der Kalenderwoche und eine Liste von Terminen für diese Woche. Außerdem muss ein Eintrag in der Terminliste als der gegenwärtig ausgewählte erkennbar sein, damit der `TermindetailView` darauf operieren kann. In Abb. 10.9 ist die Aufteilung dieser Daten auf die verfügbaren Kontexte dargestellt. Der Component Context enthält ein Feld namens `Woche` und eine Liste namens `Termine`. Auch der `WochenauswahlView` enthält ein Feld namens `Woche`, das als Verweis auf das gleichnamige Feld im Component Context realisiert ist.

Die eingegebene Kalenderwoche soll später dazu dienen, alle für diesen Zeitraum vorhandenen Termine aus einer Datenbank zu holen und in die Terminliste zu füllen. In Kapitel 12 über Persistenz wird beschrieben, wie sich Datenbankzugriffe implementieren lassen. Für den Moment soll es genügen, dass die eingegebenen Termine in der Terminliste im Component Context abgelegt werden. Dies ändert nichts an der Beziehung der Context-Attribute untereinander.

Die Liste der Termine wird dem View Context des `TerminuebersichtView` wiederum durch einen Verweis auf den Component Context verfügbar gemacht. Damit der `TerminuebersichtView` die Kalenderwoche als Überschrift anzeigen kann, erhält er außerdem Zugriff auf die `Woche` im Component Context.

Die Terminliste wird in einem weiteren View wiederverwendet, im `TermindetailView`. Der greift allerdings nicht auf die gesamte Liste zu, sondern nur auf den gegenwärtig ausgewählten Eintrag – die sogenannte Lead Section. Daher enthält der View Context des `TermindetailView` keine ganze Liste, sondern nur einen Verweis auf die Lead Section der Terminliste im Component Context. Der Übersichtlichkeit halber wurde dieser im Diagramm als `Termin` im Singular bezeichnet.

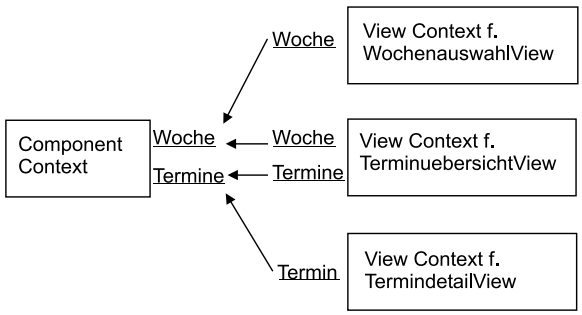

Abb. 10.9. Beziehung zwischen den Kontexten der Beispielapplikation

Es gibt mehrere Gründe dafür, die Applikationsdaten wie beschrieben zu strukturieren. Zum einen soll später der Backend-Zugriff, der die Datenfelder befüllt, an zentraler Stelle angesiedelt sein. Das kann gut im Component Context geschehen. Des Weiteren möchte man doppelte Datenhaltung vermeiden. Daher referenzieren die View-Kontexte die Daten des Component Context, statt sie zu duplizieren. Und schließlich sollen die View-Kontexte nicht mehr Datenfelder sehen, als sie wirklich benötigen. Dadurch lassen sich unerwartete Seiteneffekte vermeiden.

Definieren von Datenelementen im Context

Wir wollen nun die Kontexthierarchie aus Abb. 10.9 in der Netweaver-Entwicklungsumgebung definieren. Man erreicht dort den Component Context über den Context-Reiter des Component Controllers. Er wird als silbrige Perle dargestellt, wie in Abb. 10.10 zu sehen.

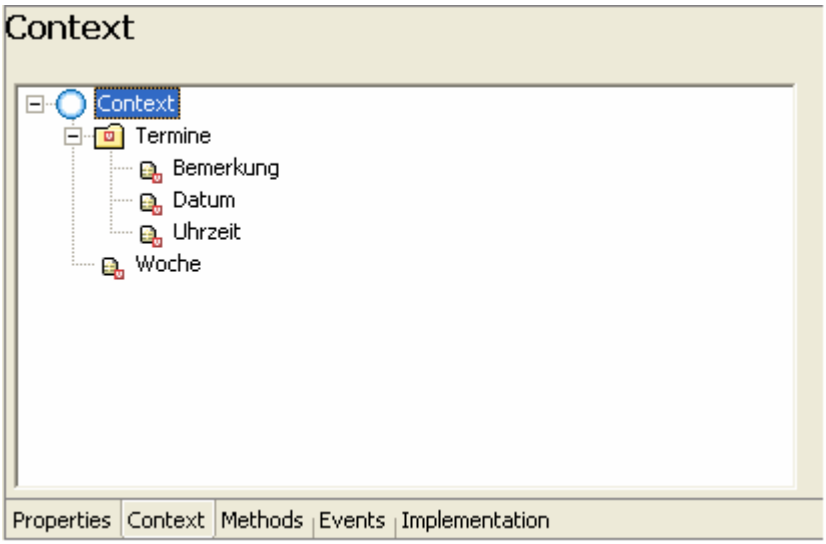

Abb. 10.10. Der Component Context in der Entwicklungsumgebung

Für unser Beispiel enthält der Component Context die Elemente `Termine` und `Woche`. Da die Untergliederung eines Context baumartig strukturiert ist, bezeichnet man die Knoten des Baums wie `Termine` als Nodes. Die Blätter des Baums wie `Woche` werden Attribute genannt.

Die Attribute, die unterhalb von `Termine` liegen, also `Bemerkung`, `Datum` und `Uhrzeit` bedürfen einer Erklärung. Jeder Node können Sie eine Kardinalität zuweisen. Die Kardinalität legt fest, wie oft die unterhalb der Node angeordneten Elemente vorkommen dürfen. Mögliche Kardinalitäten sind „0..1“, „1..1“, „0..n“ und „1..n“. Für das Beispiel weisen wir der Node `Termine` die Kardinalität „0..n“ zu. Die Attribute `Bemerkung`, `Datum` und `Uhrzeit` dürfen also mehrfach vorkommen, dürfen aber auch ganz fehlen. Die Besonderheit dieser Notation ist die, dass die drei Attribute nur zusammen vorkommen dürfen. Sie stellen also eigentlich Felder eines Datensatzes dar, und genau das benötigen wir ja, um eine Terminliste zu realisieren.

Die Definition des View Context und die Abbildung auf den Component Context sehen Sie in Abb. 10.11 und Abb. 10.12 für den `WochenauswahlView` und den `TermindetailView`. Ersterer enthält nur ein Attribut namens `Woche`, das auf das gleichnamige Attribut im Component Context abgebildet wird. Auf diese Weise gelangt die Kalenderwoche, die Sie auf der ersten Seite eingeben, in den globalen Context.

Abb. 10.11. View Context des `WochenauswahlView`

Der `TerminuebersichtView` enthält dieselbe Unterstruktur wie der Component Context, also `Termine` und `Woche`. Beide werden auch direkt auf ihre Entsprechung im Component Context abgebildet.

Interssanter ist der `TermindetailView`, dessen Context Mapping in Abb. 10.12 gezeigt wird. Termindetails werden ja nur für einen einzelnen Termindatensatz angezeigt. Deswegen besitzt der Context des `TermindetailView` neben dem Attribut `Woche` lediglich die weiteren Attribute `Bemerkung`, `Datum` und `Uhrzeit`, nicht aber einen strukturierten Knoten. Das Mapping bildete die drei Termineigenschaften auf die entsprechenden Attribute im Component Context ab, die unterhalb der `Termine`-Node liegen.

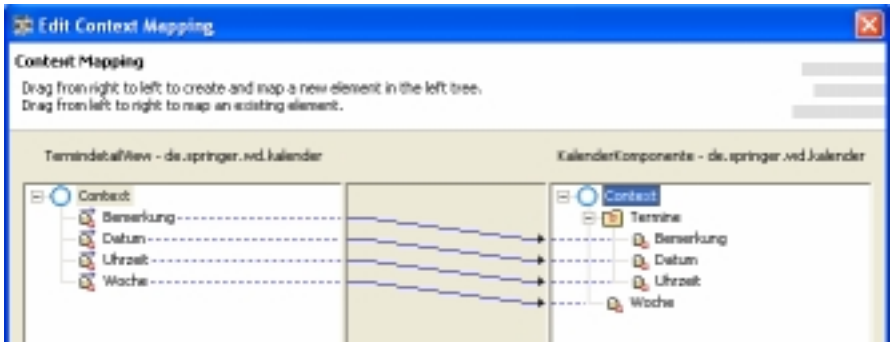

Abb. 10.12. View Context des `TermindetailView`

Solch eine Zuordnung wirkt auf den ersten Blick unsinnig, denn `Termine` ist ja eine Liste, enthält also viele Exemplare der besagten Attribute. Auf welches davon sollte sich die Zuordnung beziehen? In der Tat bezieht sich dieses Mapping nur auf ein einziges Dreitupel von Attributen, nämlich die weiter oben erwähnte Lead Section. Die Lead Section ist diejenige Zeile einer Liste, die sich gegenwärtig in Bearbeitung befindet. Diese Bedeu-

tung wird unter anderem dann mit Leben gefüllt, wenn die Liste an ein entsprechendes Oberflächenelement zur Listendarstellung gekoppelt wird. Dann entspricht die Lead Section der jeweils ausgewählten Listenzeile.

Die Lead Section einer Listen-Node in Web Dynpro entspricht der Work Area einer internen Tabelle in ABAP.

Es bleibt anzumerken, dass nicht alle Attribute oder Nodes eines View Context auf den Component Context abgebildet werden müssen. Es kann durchaus Daten geben, die nur innerhalb eines Views benötigt werden und nicht mit anderen Views oder dem Backend-System ausgetauscht werden. Denken Sie an eine Rechnung aus mehreren Einzelpositionen, von der nur der Endbetrag in der Datenbank gespeichert wird. Die Einzelpositionen sind als Attribute des View Context ohne Abbildung zum Component Context angemessen modelliert.

Einbinden der Datenelemente in die Oberfläche

Doch wofür treibt man so großen Aufwand mit den Datenelementen im Context? Doch nur, um sie schließlich an der Oberfläche anzuzeigen! Sie können im Properties-Reiter eines Views an den meisten Stellen, an denen Sie freie Texte eingeben, alternativ ein Context-Element angeben. Dafür ist am rechten Ende des Feldes ein Auswahl-Button vorgesehen. Auf diese Weise gelangt die Kalenderwoche in die ersten Zeilen des `TerminuebersichtView` und des `TermindetailView`, siehe Abb. 10.3 und 10.4.

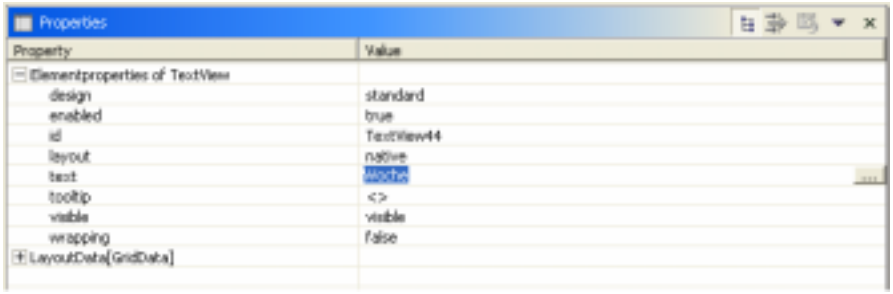

Abb. 10.13. Einbinden des `Woche`-Attributs über den Properties-Reiter

Ein wenig komplizierter ist es, die Terminliste aus dem View Context des `TerminuebersichtView` an das Oberflächenelement für die entsprechende Tabelle zu koppeln. Dazu gehen Sie in die Layout-Sicht dieses

Views und rufen das Kontextmenü der Tabelle auf. Über „Create Binding“ können Sie nun den Tabellenspalten Context-Attribute zuordnen.

Java-Code der Präsentationslogik

Bei den Untersuchungen für dieses Buch hat sich die deklarative Beschreibungsweise von Web Dynpro als durchaus tragfähig erweisen. Mit einigen Klicks und Parametereinstellungen gelingt es, das Grundgerüst einer navigierbaren Benutzeroberfläche zusammenzustellen. Das Ergebnis wirkt sauber strukturiert, so dass es zu mehr als nur zum schnellen Prototyping geeignet ist.

Früher oder später gelangt man aber an den Punkt, an dem man Code schreiben muss, um die gewünschte Logik festzuhalten. Dafür sieht das Web-Dynpro-Konzept eine klare Aufgabenteilung vor. Die Bedienlogik ist im Frontend anzusiedeln, also im Web Dynpro. Die Geschäftslogik der Applikation findet ihren Platz im Backend, also typischerweise in Session Beans, wie in Kapitel 12 beschrieben.

Innerhalb des Frontends ist wiederum eine Aufgabenteilung zwischen Component Controller und den View Controllern vorgesehen. Der Component Controller fungiert als Bindeglied zum Backend. Er beschafft also die zur Anzeige benötigten Daten und schreibt sie bei Bedarf wieder zurück. Die View Controller dagegen bereiten den Informationsstrom von und zum Benutzer auf. Sie führen also Format- und Plausibilitätsprüfungen an den Benutzereingaben durch und stellen andererseits vorhandene Daten so dar, dass der Benutzer damit etwas anfangen kann. Im weiteren Sinne umfasst der zuletzt genannte Punkt auch die Navigation zwischen unterschiedlichen Views.

Der Java-Code, der die Bedienlogik innerhalb der View Controller abwickelt, lässt sich hier sehr gut anhand des Terminkalenders erläutern. Der Code des Component Controllers wird ebenfalls im Persistenz-Kapitel 12 beschrieben, da dazu noch ein paar Grundlagen fehlen, unter anderem die bereits angekündigte Model Node.

Jeder Knoten und jedes Attribut innerhalb eines View Context ist über eine Java-Schnittstelle erreichbar. Das ist notwendig, um innerhalb des View Controllers die gewünschten Aktionen vorzunehmen. Betrachten wir dazu den `TerminuebersichtView`. Wenn man auf den Einfügen-Button drückt, wird die Aktion `Einfuegen` angestoßen. Sie hat nicht nur den Zweck, zum `TermindetailView` zu wechseln, in dem man die Werte für den neuen Termin eingeben kann. Vielmehr muss sie zunächst sicherstellen, dass ein neuer leerer Eintrag in die Terminliste eingefügt wird. Außerdem muss sie die Lead Section auf den neuen Termin bewe-

gen. Hier ist die entsprechende Methode der Klasse `TerminuebersichtView` zu sehen:

```
public void onActionEinfuegen(IWDCustomEvent wdEvent ) {
  //@@begin onActionEinfuegen(ServerEvent)
  ITermineElement t =
    wdContext.nodeTermine().createTermineElement();
  wdContext.nodeTermine().addElement(t);
  wdContext.nodeTermine().setTreeSelection(t);
  wdThis.wdFirePlugToTermindetailView();
  //@@end
}
```

Sämtliche Zugriffe auf den View Context nimmt man über das vorgegebene Java-Attribut `wdContext` vor. Es enthält für jeden Unterknoten eine eigene Zugriffsmethode, hier `nodeTermine`. Die Lead Section verschiebt man über `setTreeSelection`. Analog dazu geschieht das Löschen eines Termineintrags aus der Liste. Die gegenwärtig ausgewählte Zeile ermittelt man über `getCurrentElement`. Da nach der Löschaktion weiterhin die Terminliste angezeigt werden soll, entfällt in der Methode `onActionLoeschen` der Aufruf von `wdFirePlugTo...`, der in `onActionEinfuegen` noch notwendig war.

```
public void onActionLoeschen(IWDCustomEvent wdEvent ) {
  //@@begin onActionLoeschen(ServerEvent)
  wdContext.nodeTermine().removeElement(
    wdContext.nodeTermine().getCurrentElement());
  //@@end
}
```

10.3 Weitere Web-Dynpro-Mechanismen

Das kurze Beispiel eines Web Dynpro zur Terminverwaltung kann keinen vollständigen Überblick über diese Technologie geben. Wenn Sie Web Dynpro in Ihrem Projekt einsetzen wollen, sollten Sie noch eine Reihe weiterer Mechanismen kennen.

Windows, View Sets, etc.

Unsere Beispielapplikation wird immer in einem einzigen Browserfenster angezeigt. Sie können alternativ auch Applikationen entwickeln, die sich über mehrere Fenster erstrecken. Die Abstraktion eines Fensters im Netweaver Developer Studio nennt sich Window. Eine Applikation kann also

mehrere Windows enthalten, von denen aber immer eines als Hauptfenster markiert werden muss.

Innerhalb eines Fensters befindet sich im Normalfall ein View. Sie können aber auch sogenannte View Sets definieren. Das sind Gruppen von Views, von denen jeder einzelne View in einer View Area Platz findet.

Im Normalfall verfügt Ihre Applikation über einen Component Controller und mehrere View Controller. Gelegentlich ist es aber auch sinnvoll, zusätzliche Controller zu definieren, die nicht an einen speziellen View gekoppelt sind. Diese Controller können einem bestimmten Zweck dienen wie etwa der Backend-Anbindung.

Web Dynpro im Enterprise Portal

Ihren vollen Nutzen entfalten Web Dynpro in Kombination mit dem Enterprise Portal. Das Portal bietet wie in Übersichtskapitel 8 beschrieben eine vereinheitlichte Ausführungsumgebung für unterschiedliche Applikationen.

Web Dynpro, die im Enterprise Portal laufen, werden als „iView" bezeichnet. Es sind allerdings keine Änderungen am Web Dynpro nötig, um es zu einem iView zu machen. Voraussetzung ist, dass Ihr Web Dynpro bereits auf einem Web AS deployt ist und dass Sie über ein laufendes Enterprise Portal verfügen. Über die Verwaltungskonsole des Portal-Servers machen Sie dem Portal ihr Web Dynpro bekannt. Der Bezug wird über die URL des Web AS hergestellt, auf dem es deployt wurde, und über den Namen der Development Component, die Ihr Web Dynpro enthält.

Ist Ihr Web Dynpro einmal in das Portal integriert, kann es an dem Client Side Eventing teilnehmen, mit dem unterschiedliche Applikationen innerhalb des Browsers miteinander kommunizieren. Durch einen Aufruf von `WDPortalEventing.fire` in Ihrem Web-Dynpro-Code lösen Sie ein solches Event aus. Als Parameter geben Sie die Portal-URL, den Event-Namen und einen Parameter mit. Bereits zum Übersetzungszeitpunkt wird daraus ein Aufruf im Client Side Framework generiert. Er bewirkt, dass in einer anderen Applikation, die sich für besagtes Event angemeldet hat, eine Web-Dynpro-Aktion ausgeführt wird. Das Anmelden für ein Event erfolgt über `WDPortalEventing.subscribe`, wobei Sie wiederum die Portal-URL und den Event-Namen und als drittes die anzustoßende Aktion übergeben.

An dieser Stelle kommt die Mächtigkeit des Web-Dynpro-Konzepts besonders gut zur Geltung. Man überlege sich, welche Umsetzung notwendig ist, um durch den Server-seitig hinterlegten Code einen Kommunikationsmechanismus innerhalb des Clients zu steuern. Sie können dem Netweaver

dabei ein wenig in die Karten schauen. Dies ist ein Auszug aus dem Quelltext der Webseite, die von dem ersten View der Terminkalender-Applikation generiert wird:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
  Transitional//EN">
<html>
...
<body ...>
...
<script language="JavaScript1.2">
if (!window.csf) {
  csf = parent.csf;
  csf.initialize(window, csf);
}
var appWindow = window;
new CSF_CSFManager(appWindow,...);
</script>
...
```

So bekommt das Client Side Framework als JavaScript-Objekt `csf` ein Gesicht, auch wenn sich der von der Entwicklungsumgebung generierte Code prinzipiell von einer Version zur anderen vollständig ändern könnte.

Das Web-Forum zum Buch, von dem auch dieses Dokument stammt, findet sich unter
www.nw-stammtisch.de