

ABAP für Fortgeschrittene

Teil 4: Web Dynpro

Copyright

- Das vorliegende Skriptum baut zu großen Teilen auf den Publikationen zum TAW11- und TAW12-Kurs – das Copyright dieser Teile liegt bei der SAP AG.
- Die in diesem Kurs verwendeten Abbildungen wurden – falls nicht anders gekennzeichnet – in Anlehnung zum TAW11- und TAW12-Kurs erstellt. Das Copyright dieser Teile liegt bei der SAP AG.
- Für alle Screenshots im Skriptum, auch wenn diese nur verkürzt oder auszugsweise gezeigt werden, gilt der Hinweis: Copyright SAP AG
- Die Weitergabe und Vervielfältigung dieser Publikation oder von Teilen daraus sind, zu welchem Zweck und in welcher Form auch immer, ohne die schriftliche Genehmigung von Prof. Dr. Heimo H. Adelsberger, Dipl.-Wirt.-Inf. Pouyan Khatami und Dipl.-Wirt.-Inf. Taymaz Khatami nicht gestattet..

Inhaltsverzeichnis

COPYRIGHT	2
INHALTSVERZEICHNIS	3
ABBILDUNGSVERZEICHNIS	5
7 WEB DYNPRO.....	8
7.1 EINFÜHRUNG	8
7.2 VORTEILE DER WEB-DYNPRO-ARCHITEKTUR	10
7.3 ELEMENTE DER WEB-DYNPRO-ARCHITEKTUR	11
7.4 PRAXIS: EIN HELLO-WORLD PROGRAMM MIT WEB DYNPRO	21
7.5 WEB DYNPRO-CONTROLLERS	25
7.5.1 Allgemeine Elemente von Controllers	26
7.5.2 Elemente von Component- und Custom-Controllers	28
7.5.3 Elemente von View-Controllers.....	29
7.5.4 Elemente von Window-Controllers.....	30
7.5.5 Praxis: Übung zur Navigation zwischen Views	30
7.6 HANDHABUNG VON DATEN MIT CONTEXTS.....	34
7.6.1 Lead Selection	37
7.6.2 Singleton-Knoten	38
7.6.3 Supply-Funktionen	39
7.6.4 Context-Mapping	41
7.6.5 Praxis: Übung zum Datenaustausch über den Context	42
7.7 WEB-DYNPRO-OBERFLÄCHENGESTALTUNG.....	48
7.7.1 Praxis: Testen von UI-Elementen	49
7.7.2 Hierarchien von UI-Elementen	49
7.7.3 Einfügen von UI-Elementen.....	53
7.7.4 Datenbindung	55
7.7.5 Praxis: Übung zu Datenbindung und UI-Element-Eigenschaften.....	58
7.7.6 Texte aus dem Dictionary	59
7.7.7 Zusammengesetzte UI-Elemente.....	60
7.7.8 Das Table-UI-Element	61
7.7.9 Praxis: Übung zum Tray-UI-Element	64
7.7.10 Praxis: Übung zur Oberflächengestaltung.....	66
7.8 PROGRAMMIERUNG IN WEB DYNPRO-COMPONENTS	71
7.8.1 Methoden von Controllern	71
7.8.2 Attribute von Controllern	73
7.8.3 Kontextzugriff zur Laufzeit.....	74
7.8.3.1 Zugriff auf Knoten und Elemente.....	74
7.8.3.2 Zugriff auf Attribute	76
7.8.4 Einfügen und Entfernen von Kontextelementen	81
7.8.5 Praxis: Übung zum Contextzugriff über eine Hook-Methode	86
7.8.6 Praxis: Übung zu Bindung von Internen Tabellen, Lead-Selection und Supply-Funktionen.....	87

7.9	INTERNATIONALISIERUNG UND NACHRICHTEN IN WEB DYNPRO-ANWENDUNGEN.....	92
7.9.1	Möglichkeiten zu Internationalisierung von Texten	92
7.9.2	Auslösen von Nachrichten in Web Dynpro.....	97
7.9.3	Praxis: Übung zur Internationalisierung	101
7.9.4	Praxis: Übung zu Nachrichten.....	102
7.10	KONTROLLFRAGEN	105
7.11	ANTWORTEN.....	106
7.12	KAPITELABSCHLUSS	107

Abbildungsverzeichnis

Abbildung 1: Erstellen von Web Dynpro-Anwendungen	9
Abbildung 2: Anwendungsszenarien mit Web Dynpro.....	9
Abbildung 3: Bestandteile von Web Dynpro-Components	11
Abbildung 4: Datenbindung und Context-Mapping.....	13
Abbildung 5: Deklaration des Context-Mappings.....	14
Abbildung 6: Datenbindung und Context-Mapping.....	14
Abbildung 7: Navigation zwischen Views	15
Abbildung 8: Windows und Viewkombination durch Container-Views	16
Abbildung 9: Aufbau des klassischen MVC-Modells.....	17
Abbildung 10: Sichtbarkeit von Windows und Views	18
Abbildung 11: Component-Controller und Sichtbarkeit	19
Abbildung 12: Ergänzen von Custom-Controllers	19
Abbildung 13: Extern sichtbare Elemente von Components.....	20
Abbildung 14: Anlegen der Component: SAP-System-Screenshot	21
Abbildung 15: Anlegen der Component: SAP-System-Screenshot	22
Abbildung 16: Automatisch angelegte Elemente: SAP-System-Screenshot.....	22
Abbildung 17: Anlegen der View: SAP-System-Screenshot	22
Abbildung 18: View-Layout: SAP-System-Screenshot	23
Abbildung 19: Einfügen eines UI-Elements: SAP-System-Screenshot	23
Abbildung 20: Anzeige des TextView-Elements: SAP-System-Screenshot.....	24
Abbildung 21: Einbetten einer View: SAP-System-Screenshot.....	24
Abbildung 22: Die WD-Anwendung: SAP-System-Screenshot	25
Abbildung 23: Die Hello-World-Anwendung im Browser	25
Abbildung 24: Allgemeiner Aufbau eines Controllers.....	27
Abbildung 25: Elemente eines Component- oder Custom-Controllers	28
Abbildung 26: Elemente eines View-Controllers.....	29
Abbildung 27: Elemente von Window-Controllers.....	30
Abbildung 28: Anlegen eines Outbound-Plugs: SAP-System-Screenshot.....	31
Abbildung 29: Automatisch angelegte Methode: SAP-System-Screenshot.....	31
Abbildung 30: Ausgeklappte Window-Struktur: SAP-System-Screenshot	32
Abbildung 31: Auswahl des Navigationsziels: SAP-System-Screenshot	32
Abbildung 32: Pflege des Button-Textes: SAP-System-Screenshot	33
Abbildung 33: Erste View im Browser	33
Abbildung 34: Zweite View im Browser	34
Abbildung 35: Aufbau eines Contexts	35
Abbildung 36: Auswirkungen der Knotenkardinalität	36
Abbildung 37: Knoten mit mehreren Elementen	37
Abbildung 38: BOOKINGS als Nicht-Singleton-Knoten	38
Abbildung 39: BOOKINGS als Singleton-Knoten	39
Abbildung 40: Verwendung einer Supply-Funktion	40
Abbildung 41: Context-Mapping: SAP-System-Screenshot	41
Abbildung 42: Beispiel für Context-Mapping.....	42
Abbildung 43: Anlegen eines Kontextknotens: SAP-System-Screenshot.....	43
Abbildung 44: Eigenschaften des anzulegenden Knotens: SAP-System-Screenshot	43
Abbildung 45: Strukturkomponenten übernehmen: SAP-System-Screenshot	44
Abbildung 46: Context des Component-Controllers: SAP-System-Screenshot	44
Abbildung 47: Mapping-Pfad in den Knoteneigenschaften: SAP-System-Screenshot	44
Abbildung 48: Template Gallerie [sic!]: SAP-System-Screenshot	45
Abbildung 49: Context-Binding erzeugen: SAP-System-Screenshot	46
Abbildung 50: Eingabeformular der INPUT_VIEW: SAP-System-Screenshot	46
Abbildung 51: Eingabehilfe in Web Dynpro-Oberflächen: SAP-System-Screenshot	47

Abbildung 52: Kategorien von UI-Elementen: SAP-System-Screenshot	48
Abbildung 53: Testanwendung für UI-Elemente: SAP-System-Screenshot	49
Abbildung 54: Darstellung der Elementenhierarchie: SAP-System-Screenshot.....	50
Abbildung 55: FlowLayout	51
Abbildung 56: RowLayout.....	51
Abbildung 57: MatrixLayout	52
Abbildung 58: Die colSpan-Eigenschaft.....	52
Abbildung 59: GridLayout.....	53
Abbildung 60: Elemente einfügen: SAP-System-Screenshot	54
Abbildung 61: Elementeneigenschaften: SAP-System-Screenshot	54
Abbildung 62: UI-Elemente per Context bearbeiten.....	55
Abbildung 63: Schritte zur Anzeige von Daten: SAP-System-Screenshot (Montage)	56
Abbildung 64: Datentransport - eine wechselseitige Beziehung: SAP-System-Screenshot (Montage)	57
Abbildung 65: Typen der Web Dynpro Runtime in der Suchhilfe: SAP-System-Screenshot	58
Abbildung 66: Layout mit CheckBox: SAP-System-Screenshot	58
Abbildung 67: Eingabebereite Felder der zweiten View: SAP-System-Screenshot	59
Abbildung 68: Invertieren von Eigenschaften: SAP-System-Screenshot	59
Abbildung 69: Text aus dem Dictionary: SAP-System-Screenshot	60
Abbildung 70: Tree als Beispiel für ein zusammengesetztes UI-Element: SAP-System-Screenshot	60
Abbildung 71: TransparentContainer aus der Übung: SAP-System-Screenshot.....	61
Abbildung 72: Beispiel für eine Tabelle: SAP-System-Screenshot	61
Abbildung 73: Datenbindung von TableColumn-Elementen.....	62
Abbildung 74: Lead-Selection in der Oberfläche	63
Abbildung 75: Auswahlkardinalität eines Kontextknotens: SAP-System-Screenshot.....	64
Abbildung 76: Automatisch angelegtes untergeordnetes UI-Element : SAP-System-Screenshot	65
Abbildung 77: Zugriff auf das MIME-Repository: SAP-System-Screenshot	65
Abbildung 78: Oberfläche mit Tray und Image: SAP-System-Screenshot	66
Abbildung 79: Controllerauswahl im Wizard: SAP-System-Screenshot	67
Abbildung 80: Automatisch generierte Kontextknoten und -attribute: SAP-System-Screenshot	68
Abbildung 81: Vom Wizard generierte Felder: SAP-System-Screenshot.....	69
Abbildung 82: Durch den Wizard generierte Tabelle: SAP-System-Screenshot	69
Abbildung 83: Wizard für den Methodenauftrag: SAP-System-Screenshot	70
Abbildung 84: Generierter Code: SAP-System-Screenshot	70
Abbildung 85: Ausgabe des Programms: SAP-System-Screenshot	71
Abbildung 86: Methoden WDDOEXIT und WDDOINIT eines Controllers: SAP-System-Screenshot.....	72
Abbildung 87: Die Methoden WDDOBEFORENAVIGATION und WDDOPOSTPROCESSING: SAP-System-Screenshot	72
Abbildung 88: Schnittstelle der Methode WDDOMODIFYVIEW: SAP-System-Screenshot	73
Abbildung 89: Standardattribute WD_CONTEXT und WD_THIS: SAP-System-Screenshot	73
Abbildung 90: Das Attribut WD_COMP_CONTROLLER: SAP-System-Screenshot	74
Abbildung 91: Zugriff auf einen Kontextknoten	75
Abbildung 92: Zugriff auf ein Element eines Kontextknotens.....	76
Abbildung 93: Zugriff auf ein einzelnes Attribut.....	77
Abbildung 94: Zugriff auf alle statisch definierten Attribute eines Elements	78
Abbildung 95: Zugriff auf die statisch definierten Attribute aller Elemente eines Knotens	79
Abbildung 96: Setzen eines einzelnen Attributs	80
Abbildung 97: Setzen aller statisch definierter Attribute	80
Abbildung 98: Holen der Knoten-Referenz	82
Abbildung 99: Erzeugen eines Elements.....	82
Abbildung 100: Setzen der Attributwerte des Elements	83
Abbildung 101: Einfügen des Elements	83
Abbildung 102: Direktes Einfügen von Daten aus einer Struktur	84
Abbildung 103: Einfügen mehrerer Elemente über eine Interne Tabelle	85
Abbildung 104: Löschen eines Elements	85

Abbildung 105: Generierter Code zum Elementzugriff: SAP-System-Screenshot	87
Abbildung 106: Das definierte Eingabeformular: SAP-System-Screenshot	88
Abbildung 107: Mapping des Contexts: SAP-System-Screenshot.....	88
Abbildung 108: Holen einer Referenz des Knotens: SAP-System-Screenshot.....	89
Abbildung 109: Binden der Tabelle: SAP-System-Screenshot.....	89
Abbildung 110: Wizard-Verwendung für den Methodenaufruf: SAP-System-Screenshot.....	89
Abbildung 111: Ausgabe der Flüge: SAP-System-Screenshot	90
Abbildung 112: Vorgegebener Code nach entfernen der Kommentarsterne: SAP-System-Screenshot.....	90
Abbildung 113: Zugriff auf das Parent-Element: SAP-System-Screenshot.....	90
Abbildung 114: Oberfläche der Buchungsanwendung: SAP-System-Screenshot	91
Abbildung 115: Übersetzung einer Web Dynpro-Oberfläche	92
Abbildung 116: Geänderte Sprache beim Login: SAP-System-Screenshot	93
Abbildung 117: Durcheinander der Sprachen: SAP-System-Screenshot	93
Abbildung 118: Texte aus dem Dictionary: SAP-System-Screenshot	94
Abbildung 119: Auswahl eines OTR-Kurztextes: SAP-System-Screenshot.....	95
Abbildung 120: Auswahl einer Assistance-Klasse: SAP-System-Screenshot	95
Abbildung 121: Schnittstelle der Methode get_text: SAP-System-Screenshot.....	96
Abbildung 122: Nachrichten mit dem Wizard: SAP-System-Screenshot	97
Abbildung 123: Generierter Code: Allgemeiner Teil und Holen der Nachrichtenmanager-Referenz	97
Abbildung 124: Methoden des Interfaces IF_WD_MESSAGE_MANAGER: SAP-System-Screenshot.....	98
Abbildung 125: Melden einer Nachricht.....	99
Abbildung 126: Beispiele für Aufrufe unterschiedlicher Methoden zum Melden von Nachrichten	100
Abbildung 127: Melden einer Nachricht der Kategorie EXCEPTIONS	101
Abbildung 128: Beispiel für einen OTR-Text: SAP-System-Screenshot.....	102
Abbildung 129: Auslesen des Kontextknotens: SAP-System-Screenshot	103
Abbildung 130: Meldung erzeugen: SAP-System-Screenshot.....	103
Abbildung 131: Fertige Fehlermeldung: SAP-System-Screenshot	104

7 Web Dynpro

In diesem Kapitel lernen Sie mit Web Dynpro eine Technologie kennen, mit der Sie moderne, browserbasierte Anwendungen entwickeln können.

7.1 Einführung

Webbasierte Anwendungen spielen bereits seit langer Zeit eine wichtige Rolle für Anbieter von ERP-Systemen wie der SAP AG. Durch Anwendungen, die im Browser laufen, entfällt der Aufwand der Installation einer Client-Software wie der SAP GUI, so dass ein Zugriff vieler Nutzer ohne den sonst üblichen administrativen Aufwand möglich wird, wenn man davon ausgeht, dass auf heutigen Arbeitsplatzrechnern in der Regel ein Browser bereits vorhanden ist.

Der Trend zu webbasierten Anwendungen spiegelt sich auch in der Entwicklung der Architektur der Software von SAP wider. Bis zum Release 6.10 war noch ein spezielles Internet-Gateway wie der Internet Transaction Server erforderlich, das dann jedoch durch den Wandel der klassischen SAP Basis hin zu einem weborientierten Anwendungsserver überflüssig wurde. Durch Business Server Pages können hier spezielle webbasierte Anwendungen realisiert werden, die mit dem restlichen SAP-System interagieren.

Mit Einführung des SAP NetWeaver 2004s (NetWeaver Application Server 7.0) wurde nun von SAP der nächste Schritt hin zu noch einfacher realisierbareren Webanwendungen gegangen. Grundlage dieser Technik ist eine saubere Trennung der Aufgaben, die bei der Entwicklung einer Webanwendung anfallen, die dem Model View Controller-Modell (MVC) entsprechen. Dabei wird dem Entwickler die Möglichkeit gegeben, möglichst große Teile seiner Anwendung deklarativ umzusetzen. Damit ist gemeint, dass nicht einzelne HTML-Elemente oder JavaScript codiert und mit der Anwendungslogik vermischt werden, sondern die Oberfläche aus UI-Elementen zusammengesetzt wird und Navigationspfade zwischen den Teilen der Anwendung beschrieben werden.

So müssen etwa keine manuellen Verlinkungen durchgeführt werden, sondern das System generiert den Code der Benutzeroberfläche automatisch in einem Framework. Code des Benutzers kann dann an vordefinierten Stellen eingefügt werden. Die Entwicklung erfolgt strukturiert, einzelne Modularisierungseinheiten, die Web-Dynpro-Components können zu komplexen Anwendungen kombiniert werden. Die entsprechenden Entwicklungswerzeuge sind in den Object Navigator (Transaktion SE80) integriert. Dort wird die Anwendung in Form eines Metamodells dargestellt. Die eigentliche Anwendung wird der Web Dynpro-Architektur entsprechend generiert.

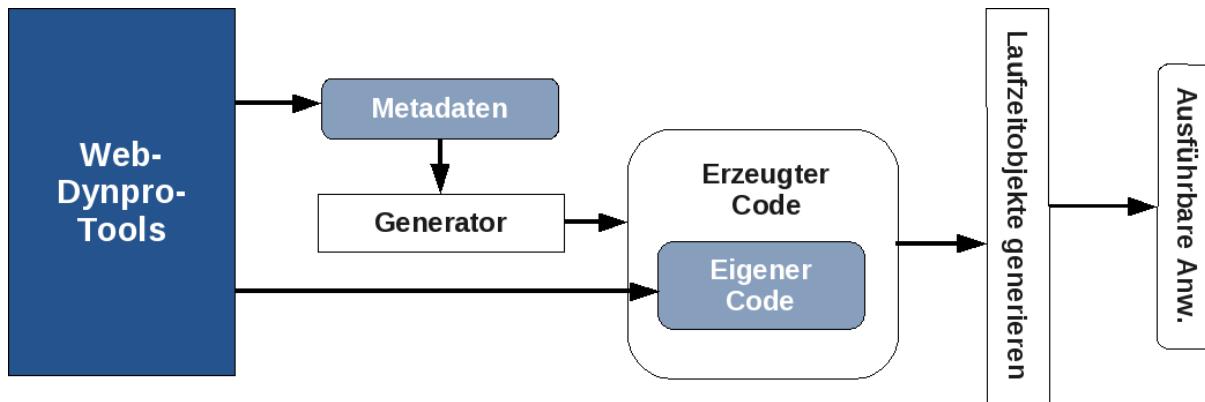


Abbildung 1: Erstellen von Web Dynpro-Anwendungen

Die obige Abbildung zeigt den Zusammenhang von deklarativen Angaben zur Anwendung und eigenem Code. Alle deklarativen Informationen gehen als Metadaten in einen Generator ein, der den dafür benötigten Code automatisch generiert. Der eigene Code des Kunden wird dann an bestimmten Stellen, im durch den Generator erzeugten Code, eingefügt. Aus dem Gesamtcode wird dann die tatsächliche Anwendung generiert, auf die die Benutzer zugreifen können.

Diese Zweiteilung der Entwicklung ermöglicht die klare Trennung von Geschäftslogik und Oberflächengestaltung. Durch die Verwendung eigenen Codes ist das Konzept universell einsetzbar, während durch die Deklaration der Oberfläche im Metamodell und Generierung des entsprechenden Codes ein einheitliches Anwendungsdesign ermöglicht wird. Hierbei bieten die entsprechenden Entwicklungstools eine umfassende Unterstützung, um die Elemente des Bildschirmlayouts zu konfigurieren, die Navigation und Fehlerbehandlung festzulegen, Datenflüsse einzurichten und die Anwendung in Komponenten aufzuteilen. Im eigenen Code können beispielsweise dynamische Bildmodifikationen vorgenommen, Geschäftsregeln implementiert oder auf Services zugegriffen werden.

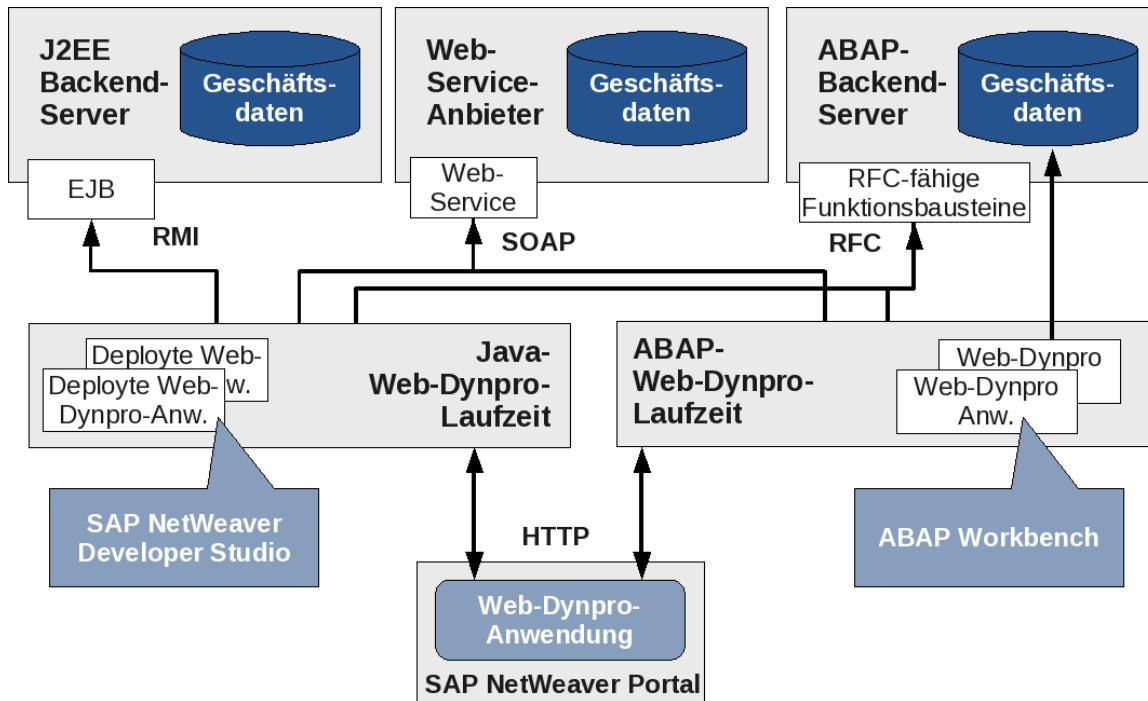


Abbildung 2: Anwendungsszenarien mit Web Dynpro

Auch mit Java sind im SAP-System webbasierte Anwendungen möglich. Diese können mit den, auf Basis von ABAP geschriebenen, Web Dynpro-Anwendungen verbunden werden.

Beide können auf Web Services über SOAP zugreifen, und mithilfe des SAP Java Connectors (JCo) können Methoden von Enterprise Java Beans (EJB) aufgerufen werden, die auf einer J2EE-Engine liegen.

Um eine ABAP-Web Dynpro-Anwendung mit Daten zu versorgen, können Methoden und Funktionsbausteine oder andere wiederverwendbare Komponenten verwendet werden. Ein direkter Datenbankzugriff mit der SELECT-Anweisung ist theoretisch auch möglich. SAP rät davon jedoch ab, da dies eine Vermischung von Ablauf- und Geschäftslogik darstellt.

Um den Code der Anwendung in wiederverwendbaren Einheiten zu kapseln, können Klassen verwendet werden, in denen die benötigte Funktionalität entwickelt wird. Es ist aber auch möglich, innerhalb des Web Dynpro Frameworks wiederverwendbare Funktionen zu hinterlegen. Hierfür werden Web Dynpro-Components angelegt, die nur diese Funktionen, aber keine eigene Oberfläche enthalten. Andere Components können auf diese Funktionen dann durch Component-Wiederverwendung zugreifen.

7.2 Vorteile der Web-Dynpro-Architektur

Durch die Einführung der Web-Dynpro-Architektur werden sowohl Vorteile für den Entwickler als auch Vorteile bei der Nutzung der Anwendung bezweckt.

Der Entwickler kann seine Anwendungen mit minimalem Aufwand erstellen, der Entwicklungsprozess läuft strukturiert ab und wird durch Werkzeuge unterstützt, die eine deskriptive Erstellung vieler Teile der Anwendung erlauben. Der Fokus liegt klar auf der Verwendung dieser Werkzeuge zur Anwendungsentwicklung. Es soll vermieden werden, eigenen Code zu schreiben, wo immer dies möglich ist. Hierfür wird im Rahmen des Metamodells die Benutzeroberfläche beschrieben, ohne dass eigener Code notwendig wäre. Erst wenn es an die Implementierung der Geschäftslogik geht, kommt eigener Code zum Einsatz. Dieser Code muss dann nur diese Aufgaben erledigen und sich nicht um die Benutzeroberfläche selbst kümmern. Es findet hier also eine saubere Trennung der Aufgaben statt. Dabei handelt es sich um eine modifizierte Implementierung des Model View Controller-Entwicklungsmodells (MVC). Auch die Internationalisierung von Anwendungen ist vorgedacht und wird vom System unterstützt.

Zusammenfassend ergeben sich folgende Vorteile aus Sicht von Entwicklern:

- Programmierung beschränkt sich auf Geschäftslogik, keine wiederholenden Aufgaben zur GUI-Programmierung
- saubere Trennung von Layout und Logik
- vielfältige Backend-Unterstützung
- Wiederverwendung von Components innerhalb des Frameworks
- Zugriff auf Web Services möglich

Der Anwender wiederum hat den Vorteil einer plattformunabhängigen Anwendung. Diese besitzt eine einheitliche Darstellung und ist ohne die Installation von SAP GUI verwendbar. Das Unternehmen kann so administrativen Aufwand sparen, wenn ohnehin ein Webbrowser auf den Arbeitsplatzrechnern der Anwender benötigt wird und entsprechend gewartet werden muss.

Zusammenfassend ergeben sich folgende Vorteile aus Sicht von Nutzern und Administratoren:

- Browserbasiertheit (keine SAP GUI erforderlich)
- komfortable Web-Anwendungen, die Standards der Benutzerfreundlichkeit (Abschnitt 508 des US-Rehabilitationsgesetzes) erfüllt
- durch Caching hohe Geschwindigkeit
- kein Neuladen der Seite zur Bildaktualisierung erforderlich
- Client-seitige Dynamik

7.3 Elemente der Web-Dynpro-Architektur

Im Folgenden werden Ihnen die Elemente vorgestellt, aus denen sich eine Web-Dynpro-Anwendung zusammensetzt. Dies sind neben den bereits oben erwähnten Components vor allem Windows, Views und Contexts.

Web Dynpro-Components sind Container für andere Elemente, die sich auf die Benutzeroberfläche oder das Programm beziehen. Sie dienen der Strukturierung komplexer Anwendungen und Entwicklung wiederverwendbarer, interagierender Einheiten. Die folgende Abbildung zeigt die Elemente, die zu einer Web Dynpro-Component gehören.

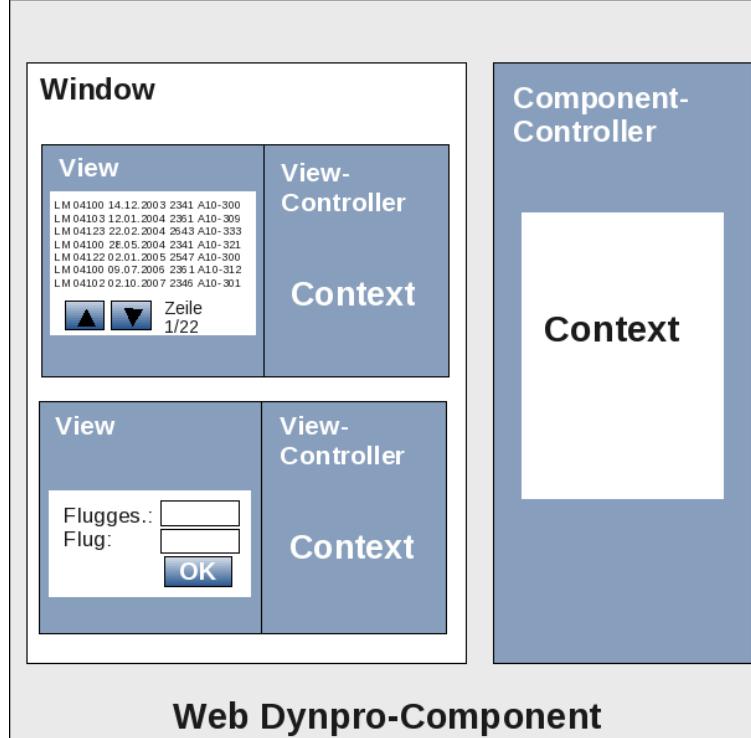


Abbildung 3: Bestandteile von Web Dynpro-Components

Die Elemente **Window** und **View** beziehen sich auf die Definition der Benutzeroberfläche, während der Quellcode in **Web Dynpro-Controllers** liegt. Der **Context** eines Controllers ist eine hierarchisch strukturierte Ablage für globale Variablen.

Eine **View** stellt einen Teil einer im Browser angezeigten Seite dar. Es wird hierbei vorausgesetzt, dass es sich um einen rechteckigen Teil handelt. Dieser enthält dann die typischen Bildschirmelemente (auch als UI-Elemente bezeichnet), etwa Tabellen, Schaltflächen oder Eingabefelder. Eine Seite, die dem Anwender dargestellt wird, kann sich aus mehreren Views zusammensetzen, es sind aber auch Seiten aus nur einer View zulässig. Die hierbei zulässigen Kombinationen und insbesondere der Ablauf zwischen den Views werden in einem **Window** definiert. Wie in der obigen Abbildung darf ein Window mehrere Views enthalten. Diese Zuordnung ist aber nicht eindeutig, das bedeutet dass ein View auch in einem oder mehreren weiteren Windows verwendet werden kann.

Die naheliegende Verwendung einer Web Dynpro-Component ist die Definition einer Web Dynpro-Anwendung, mit der die Component mit einer URL verbunden wird und vom Anwender im Browser aufgerufen werden kann. Dies ist aber nicht die einzige Zugriffsmöglichkeit. Components können auch als Unterkomponenten wiederverwendet werden. Hierbei kann die visuelle Schnittstelle der eingebundenen Component mit der der Hauptkomponente verbunden werden. Weiterhin können von der Hauptkomponente Methoden und Daten der Unterkomponente verwendet werden, die in deren Programmierschnittstelle definiert sind.

Wie oben erwähnt stellt der Context eine hierarchisch strukturierte Ablage für globale Variablen dar. In einer Anwendung werden häufig die Daten eines Controllers in einem anderen Controller benötigt. Der hierfür nötige Zugriff geschieht durch eine Referenzierung, die als **Context-Mapping** bezeichnet wird. So verwenden die Controller die Daten gemeinsam, ohne dass ein Hin- und Herkopieren erforderlich wäre. Daneben wird auch eine Verbindung von den Eingabeelementen der Benutzeroberfläche und dem entsprechenden Controller benötigt, damit die eingegebenen Daten dort zur Verfügung stehen und verarbeitet werden können. Diese Zuordnung wird als **Datenbindung** bezeichnet.

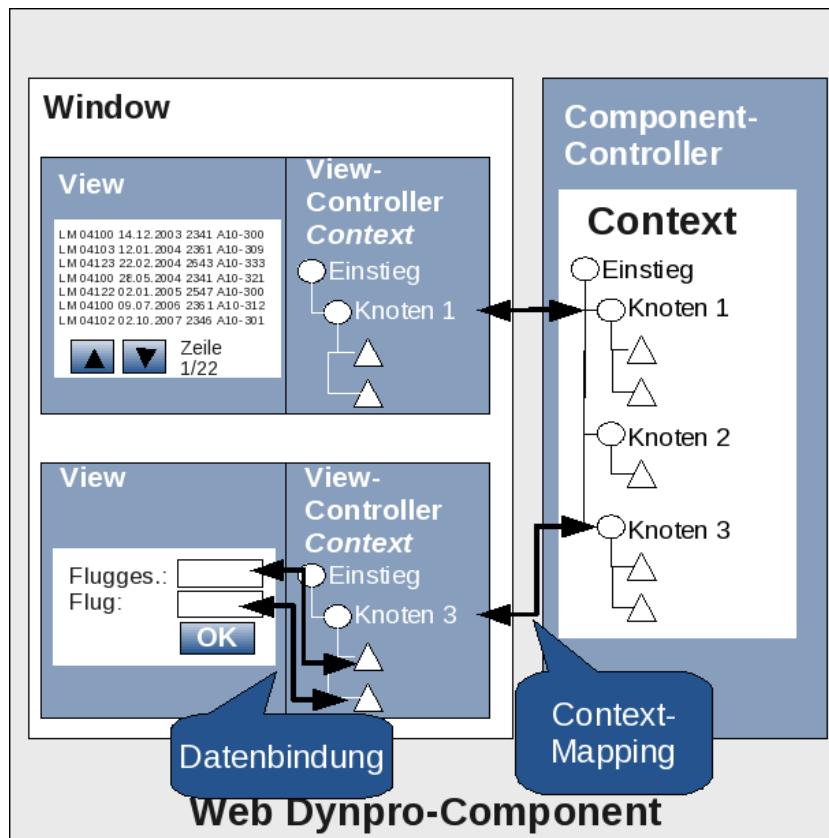


Abbildung 4: Datenbindung und Context-Mapping

Die Abbildung veranschaulicht beide Begriffe. Dargestellt ist hier auch der hierarchische Aufbau des Contexts. Die Knoten dieser Hierarchie werden miteinander verbunden. Die Definition von Datenbindung und Context-Mapping erfolgt rein deklarativ. Es ist also bei Verwendung dieses Konzepts kein Code notwendig, um einen Datenaustausch zwischen den Elementen der Anwendung zu realisieren. Durch diesen Komfort wird das Konzept vorrangig genutzt.

Es wird begrifflich zwischen dem Mapping innerhalb und zwischen Components unterschieden. Das Mapping zwischen Contexts innerhalb derselben Component wird als **internes Mapping**, das Abbilden von Contexts unterschiedlicher Components hingegen als **externes Mapping** bezeichnet. Externes Mapping wird hier nicht behandelt.

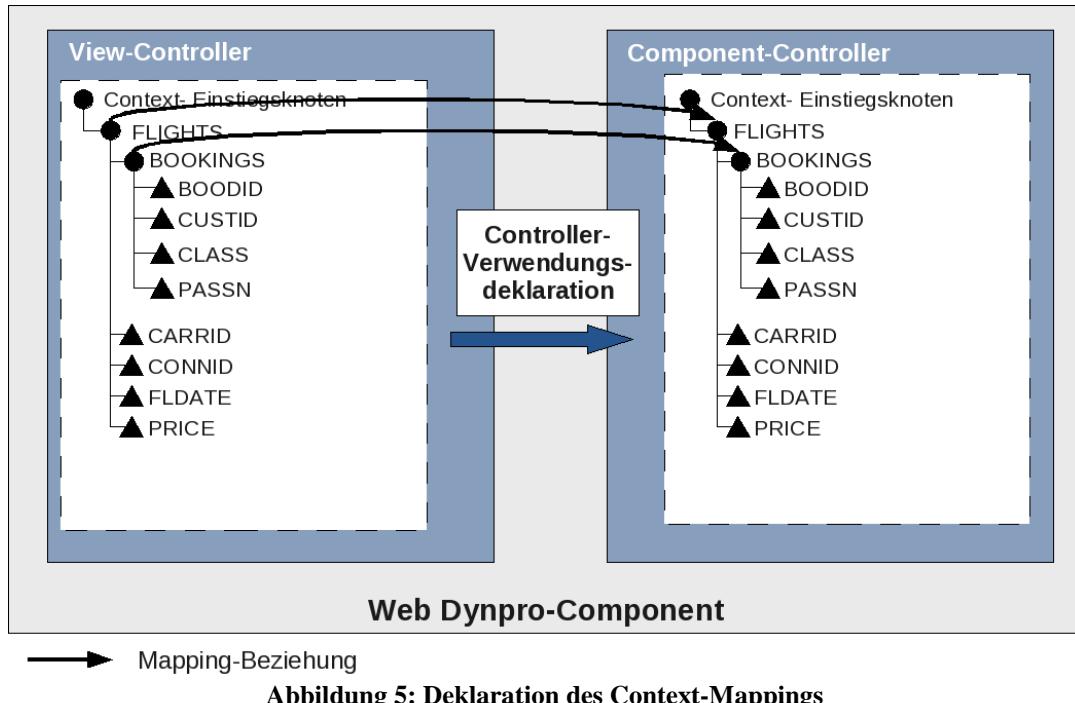


Abbildung 5: Deklaration des Context-Mappings

Die Mapping-Beziehung ist gerichtet. Der Knoten, der als Quelle der Daten dient, wird **Mapping-Ursprungsknoten**, der zugeordnete Knoten hingegen **abgebildeter Knoten** genannt. Der Mapping-Ursprung-Controller darf kein View-Controller sein.

Die Deklaration der Mapping-Beziehung wird vom Controller des abgebildeten Knoten aus vorgenommen. In diesem wird der Mapping-Ursprung-Controller als verwendeter Controller definiert (Anmerkung: Diese „umgedrehte“ Vorgehensweise bei der Deklaration erklärt auch die Richtung der Pfeildarstellung in der obigen Abbildung).

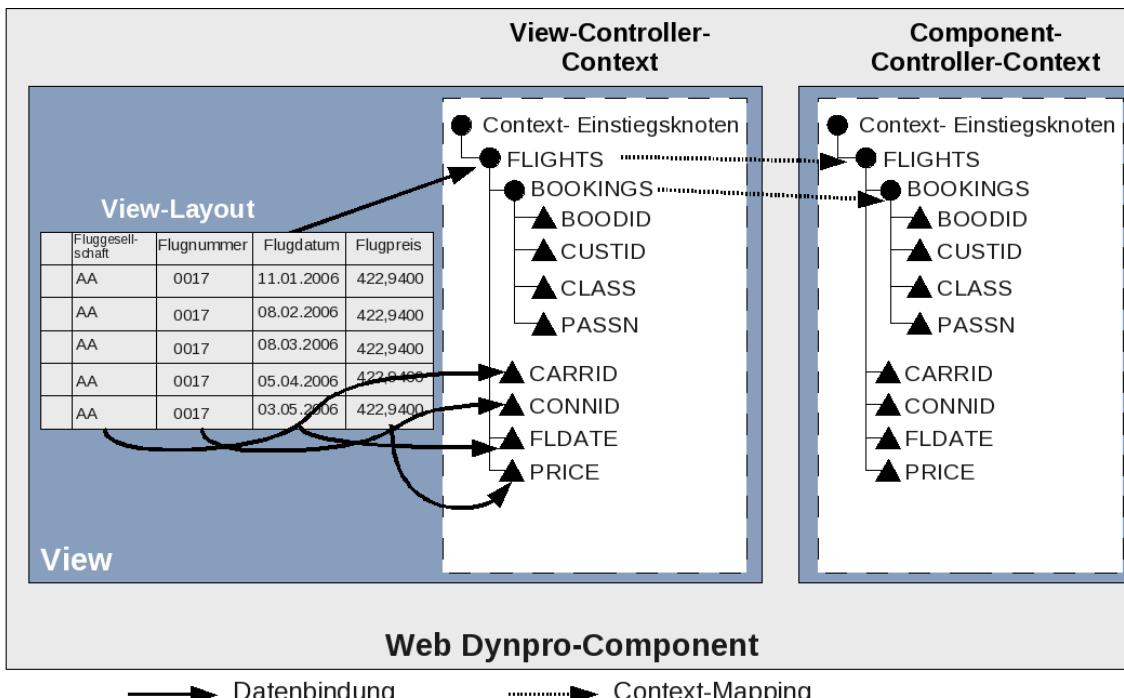


Abbildung 6: Datenbindung und Context-Mapping

Durch die Definition einer Datenbindung erfolgt automatisch ein Transport der Daten von der Oberfläche im Browser zum Context des View-Controllers und umgekehrt. Es muss sich

dabei um den View Controller derselben View handeln, die Elemente der Bildschirmoberfläche sind private Komponenten ihres View-Controllers. Dieses Konzept ermöglicht die Entkopplung von Elementen der Bildschirmoberfläche und Code im View-Controller. Soll in einer Anwendung eine Eigenschaft eines Bildschirmelements verändert werden, muss nur auf den durch die Datenbindung zugeordneten Kontextknoten und -attribute des View-Controller-Contexts zugegriffen werden. Zu den Eigenschaften, die durch die Datenbindung gesteuert werden können, zählt nicht nur der Wert. Auch Eigenschaften wie die Sichtbarkeit können hierüber gesetzt werden. Das Framework transportiert die Daten aus dem Kontext-Attribut während des Rendering-Prozesses in die Oberfläche. Nach Eingabe von Daten durch den Benutzer und Auslösen des nächsten Server-Roundtrips erfolgt der umgekehrte Transport, wobei vom Benutzer eingegebene Daten automatisch auf Ihren Typ geprüft und konvertiert werden. Fehlerhafte Eingaben führen zur Anzeige einer entsprechenden Nachricht.

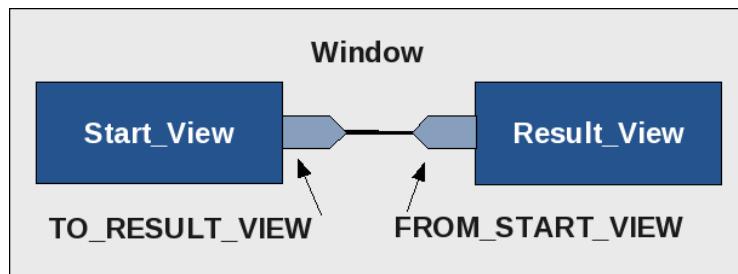


Abbildung 7: Navigation zwischen Views

Die Navigation zwischen den Views wird über Inbound- und Outbound-Plugs durchgeführt. Ein View wird verlassen, indem ein **Outbound-Plug** ausgelöst wird. Hierdurch wird wiederum ein spezielles asynchrones Ereignis ausgelöst. Diese Navigationsereignisse werden in einer Navigationswarteschlange gesammelt, die zu einem bestimmten Zeitpunkt der Web Dynpro-Verarbeitung abgearbeitet wird. Aus einer View können mehrere Outbound-Plugs ausgelöst werden. So kann die nächste Benutzeroberfläche definiert werden, die aus mehreren Views besteht.

Die durch das Auslösen (Feuern) von Outbound-Plugs erzeugten Navigationsereignisse werden von **Inbound-Plugs** behandelt. Diese sind Behandlermethoden, die auf die entsprechenden Navigationsereignisse registriert werden. Der Aufruf der Behandlermethode, also des Inbound-Plugs, erfolgt aber nicht sofort beim Auslösen des Outbound-Plugs, sondern erst nachdem die Views der aktuellen Viewgruppe ihre Outbound-Plugs beim Abarbeiten der Navigationswarteschlange ausgelöst haben. Wenn der Benutzer Daten eingegeben hat, die nicht zulässig waren und eine entsprechende Fehlermeldung erscheint, wird der Navigationsschritt abgebrochen.

Die Benennung der In- und Outbound-Plugs sollte verdeutlichen, von wo bzw. wohin navigiert wird. In Abbildung 7 sehen Sie ein einfaches Beispiel: Das Outbound-Plug, das zur Result View führt, wird **TO_RESULT_VIEW** genannt, während das Inbound-Plug des zweiten View als **FROM_START_VIEW** bezeichnet wird. So ist der Zusammenhang der Navigation besser nachvollziehbar. Die Verbindung zwischen Outbound- und Inbound-Plug wird als **Navigationslink** bezeichnet. Im Zuge dieser Navigation können Daten zwischen den Views ausgetauscht werden. Da es sich technisch um eine Ereignisbehandlung handelt, können hier die gewohnten Parameter von Ereignissen verwendet werden, die dann von der Behandlermethode entgegengenommen werden.

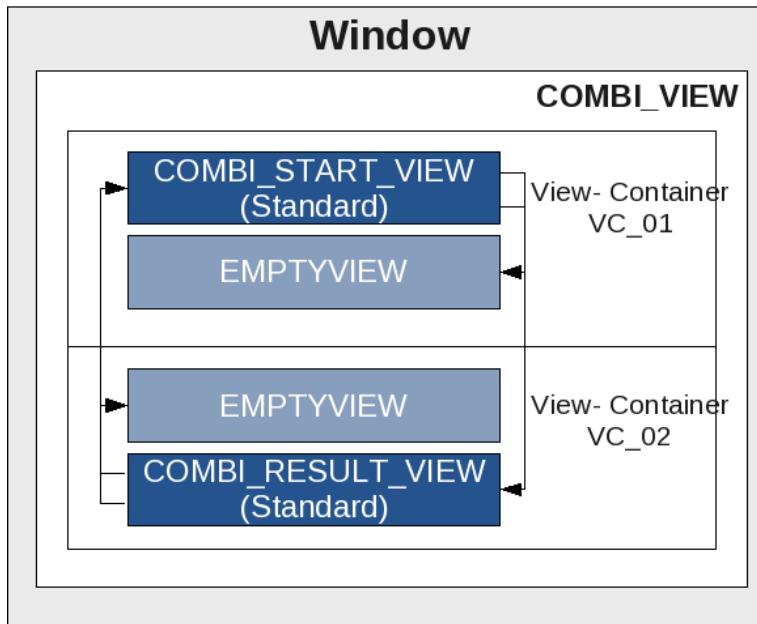


Abbildung 8: Windows und Viewkombination durch Container-Views

Alle Views einer Component müssen in einem **Window** eingebettet sein. Windows werden verwendet, um die möglichen Kombinationen von Views festzulegen und zu definieren, wie durch das Auslösen von Outbound-Plugs die View-Kombination geändert werden kann. In vielen Fällen besteht die Oberfläche aus mehreren Views. Um diese verwenden zu können, muss eine spezielle **Container-View** verwendet werden. Diese wird in das Window eingebettet und enthält Bereiche, in die wiederum die möglichen Views eingebettet werden. Diese Bereiche werden durch ViewContainerUIelements definiert und als **View-Area** bezeichnet. In jeder View-Area kann jeweils ein View angezeigt werden. Beim Start wird eine Standard-View angezeigt. Um in einer View-Area nichts anzuzeigen, kann eine leere View angelegt werden und in der View-Area angezeigt werden, indem ihr Inbound-Plug auf ein entsprechendes Navigationssereignis reagiert.

Die sichtbare View-Untermenge eines Windows zu einem bestimmten Zeitpunkt wird als **View-Komposition** bezeichnet. Durch die Navigation können entweder nur bestimmte Views in einer View-Area ersetzt werden, oder ganze View-Kombinationen des Windows. Das erscheinen einer View in einer View-Area wird durch Outbound-Plugs ausgelöst. Welche Views zu einem Zeitpunkt angezeigt werden, hängt davon ab, welchen Navigationslinks der User gefolgt ist.

Die Architektur von Web Dynpro basiert auf dem **Model View Controller**-Modell (MVC), das SAP für seine Zwecke modifiziert und erweitert hat. MVC wurde ursprünglich Ende der 1970er Jahre bei Xerox PARC vom Software-Designer Trygve Reenskaug entwickelt. In Smalltalk-80 erfolgte die erste Implementierung des Modells. Das Revolutionäre an diesem Modell war die Beschreibung der Komponenten nach funktionalen Verantwortlichkeiten, denen die Komponente genügen muss, und Nachrichtenprotokollen, auf die jede Komponente antworten sollte.

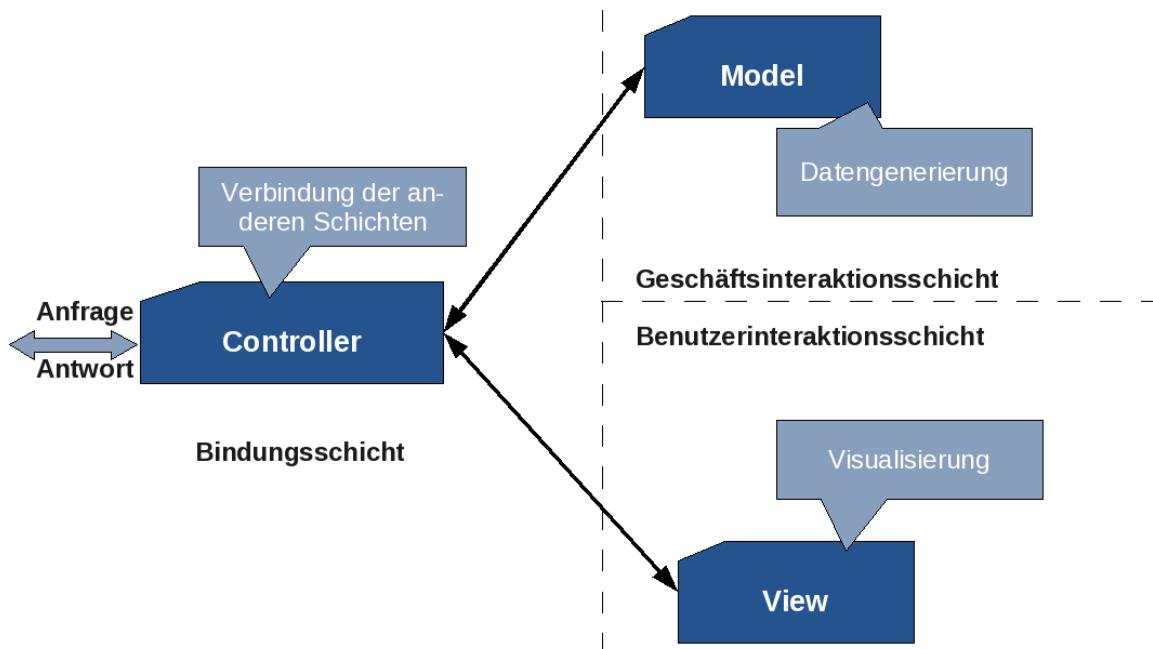


Abbildung 9: Aufbau des klassischen MVC-Modells

Die Software teilt sich dabei in drei Bereiche auf, die jeweils genau abgesteckte Aufgaben erfüllen:

- Model: Hier werden die Anwendungsdaten generiert. Dabei wird nicht betrachtet, wie diese später dargestellt werden sollen.
- View: Hier werden die Anwendungsdaten visualisiert. Diese Visualisierung ist unabhängig von der Herkunft der Daten.
- Controller: Hier werden die beiden zuvor genannten Teile miteinander verbunden. Zwischenverarbeitungsschritte werden hier ausgeführt.

Diese Aufgabentrennung findet sich auch in der Web Dynpro-Architektur wieder. Sie unterstützt die Entwicklung der Anwendung und strukturiert die Software.

Betrachtet man eine Component, so sind zwei Sichtbarkeitsbereiche zu unterscheiden. Hierbei handelt es sich um die **interne** und die **externe Sichtbarkeit**. Intern sichtbare Elemente sind nur innerhalb der Component sichtbar, während extern sichtbare Elemente auch von außen sichtbar sind.

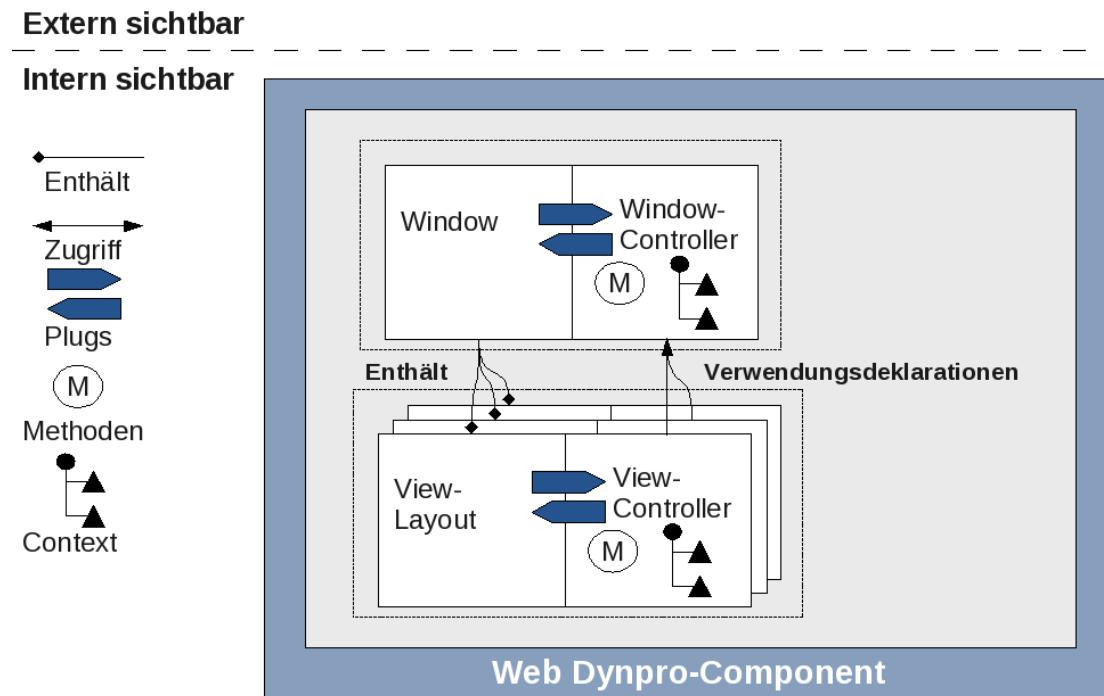


Abbildung 10: Sichtbarkeit von Windows und Views

Sie haben bereits Windows und Views kennengelernt. Diese sind intern sichtbare Komponenten von Web Dynpro-Components. Windows betten Views ein und besitzen einen Window-Controller. Die Views bestehen aus einem View-Layout und einem View-Controller. Beide Controller (View-Controller und Window-Controller) verfügen über Navigations-Plugs, Methoden und einen Context. Eine View kann in mehreren Windows verwendet werden. Für die Verbindung der Plugs gibt es Einschränkungen. Der Outbound-Plug eines Window kann mit jedem Inbound-Plug der im Window eingebetteten Views verbunden werden, und der Outbound-Plug einer View kann mit jedem Inbound-Plug des einbettenden Window verbunden werden, jedoch ist es nicht möglich, zwischen den Windows einer Component zu navigieren.

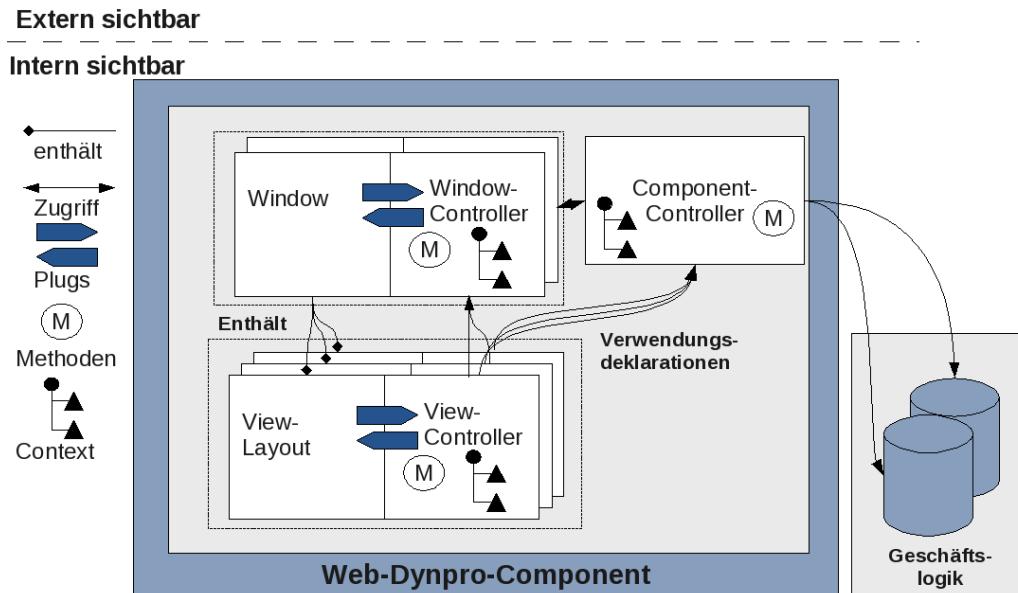


Abbildung 11: Component-Controller und Sichtbarkeit

Die Component selbst verfügt ebenfalls über einen Controller. Dieser kann komponentenweit verwendet werden und Programmlogik enthalten. Es wird nicht empfohlen, sämtliche Logik hier unterzubringen. Stattdessen sollten die Teile, die sich unmittelbar auf eine View beziehen, auch in deren Controller liegen. Darunter fallen beispielsweise Eingabeprüfungen.

Die View-Controller können die Kontextdaten und Methoden des Component-Controllers verwenden. Es kann jedoch keine Verwendungsbeziehung definiert werden, in der ein View-Controller verwendeter Controller ist, da dies dem MVC-Modell widersprechen würde.

Geschäftslogik soll sich vorzugsweise außerhalb der Component befinden (etwa durch Kapselung in einer Klasse), so dass diese bequem wiederverwendet werden kann.

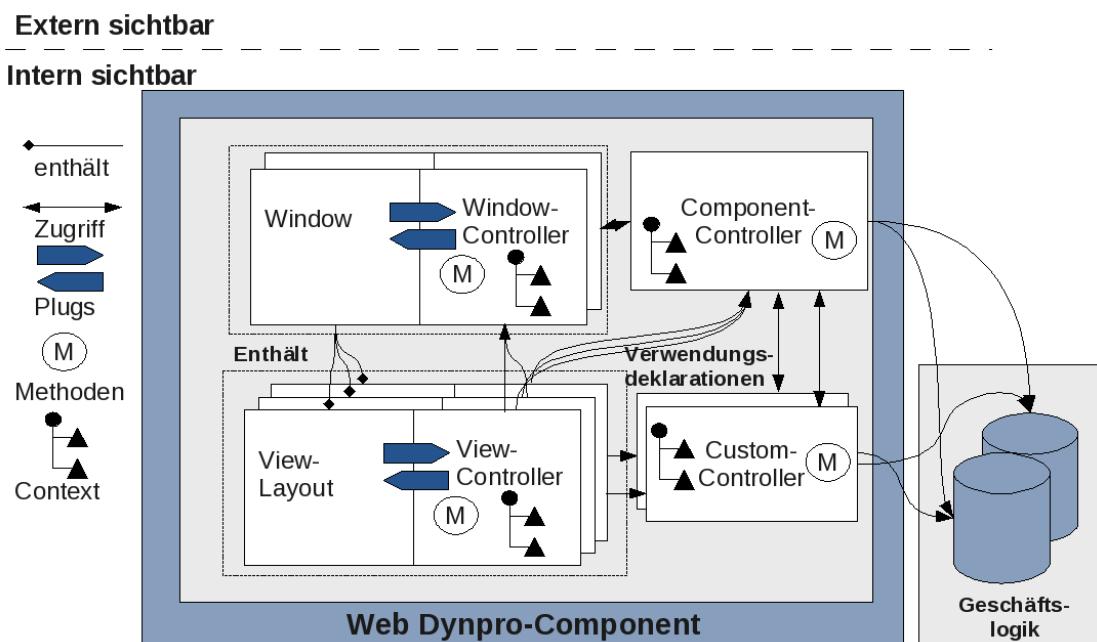


Abbildung 12: Ergänzen von Custom-Controllers

Wenn sich dennoch viel Logik im Component-Controller befindet, kommt die Verwendung von Custom Controllers infrage. Diese können zur Modularisierung verwendet werden, indem sie die Logik enthalten, die zu einem bestimmten Teil der Anwendung gehört. So kann der Funktionsumfang des Component-Controllers verringert werden.

Bislang wurden nur intern sichtbare Elemente der Component dargestellt. Für den externen Zugriff werden hingegen **Interface-Controllers** und **Interface-Views** verwendet.

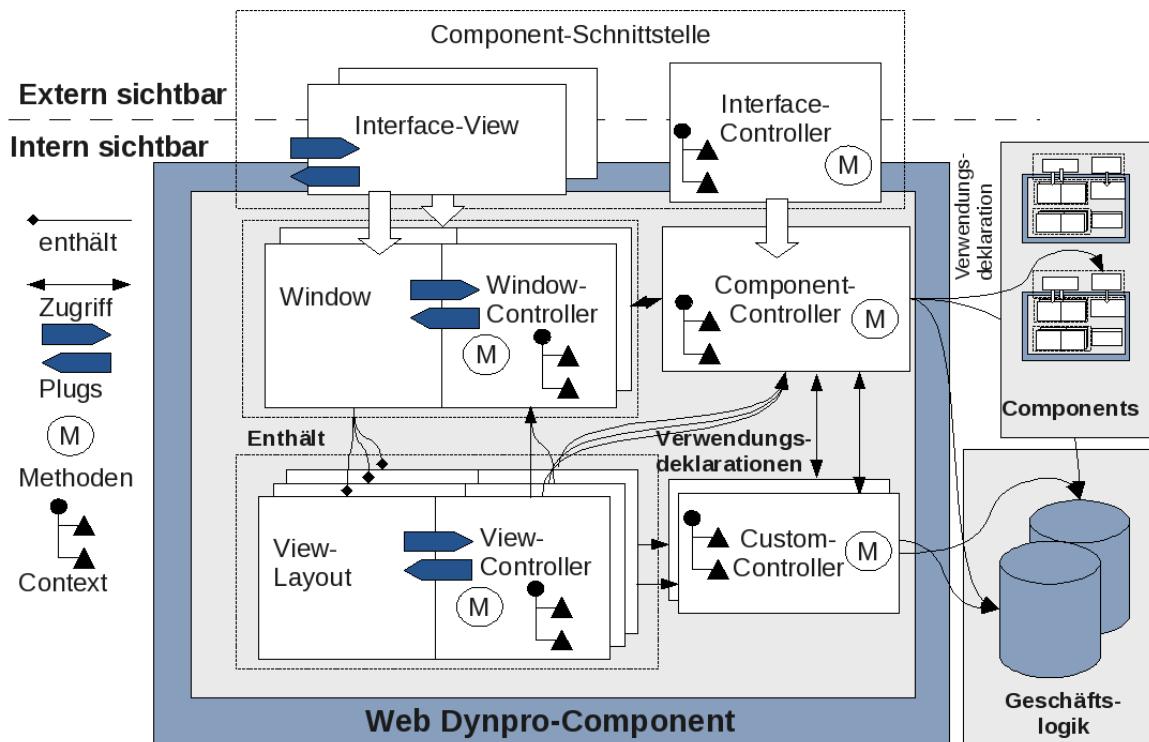


Abbildung 13: Extern sichtbare Elemente von Components

Über diese extern sichtbaren Elemente können Web Dynpro-Components aufeinander zugreifen. Hierzu wird eine Verwendungsbeziehung zwischen den Components deklariert (Anlegen einer Verwendungsinstanz). Die zugreifende Component wird dabei als **übergeordnete Component** und die Component auf die zugegriffen wird als **untergeordnete Component** bezeichnet. Die nur intern sichtbaren Component-Elemente sind für die zugreifende Component nicht sichtbar.

Der Interface-Controller dient dem Zugriff auf Daten, Methoden und Ereignisbehandler durch andere Components. Jede Component hat nur einen Interface-Controller. Die Interface-Views sind hingegen visuelle Schnittstellen zu einer Web Dynpro-Component. Für jedes Window in einer Component wird automatisch eine zugehörige Interface-View angelegt. Windows und Interface-Views stehen stets in einer 1:1-Beziehung zueinander. Durch das Interface-View kann von Außen auf das Window zugegriffen werden. Dazu stehen bestimmte In- und Outbound-Plugs des Window zur Verfügung, die die Eigenschaft **Interface** besitzen. Auf alle anderen Plugs sowie Methoden und Kontextdaten des Window besteht über die Interface-View hingegen kein Zugriff.

In der Regel hat jede Component Windows und Views, über die so zugegriffen werden kann, es gibt jedoch Ausnahmen: Components, die keine Views besitzen, benötigen auch keine Windows und implementieren dementsprechend keine Interface-View. Diese Components werden **oberflächenfreie Components** genannt. Auf diese Components kann über den Interface-Controller zugegriffen werden.

Es gibt gewisse Parallelen zwischen Web Dynpro-Anwendungen und klassischen Dynpro-Programmen (die technisch Modulpools sind). Damit Sie Ihre Dynpro-Programme starten konnten, musste eine Transaktion angelegt werden. Über diese Transaktion konnte dann auf das Dynpro-Programm zugegriffen werden, wobei in den Eigenschaften der Transaktion festgelegt wurde, welches das Einstiegsdynpro sein sollte. Bei Web Dynpro gibt es analog dazu **Web Dynpro-Anwendungen**. Eine Web Dynpro-Anwendung kann über eine URL angesprochen werden, so wie eine Transaktion über einen Transaktionscode erreichbar ist. Die Web Dynpro-Anwendung dient dann als Einstiegspunkt in eine Component, die für sich genommen nicht durch eine URL erreichbar wäre. Bei der Definition dieser Anwendung wird angegeben, welche Component aufgerufen werden soll. Diese Component wird **Einstiegs-Component** genannt. Weiterhin wird angegeben, welche Interface-View der Einstiegs-Component verwendet werden soll. Die darin angezeigte View-Komposition wird durch die Festlegung der Standard-Views bestimmt. Schließlich muss noch ein Inbound-Plug angegeben werden, der Einstiegspunkt in die Interface-View sein soll. Hierfür kommen nur Inbound-Plugs infrage, die den Typ **Startup** besitzen.

7.4 Praxis: Ein Hello-World Programm mit Web Dynpro

Im vorangegangenen Unterkapitel sind Sie mit zahlreichen neuen Begriffen konfrontiert worden, die aufgrund ihrer schieren Menge zunächst recht verwirrend erscheinen mögen. Diese Übung soll nun einen praktischen Einstieg in die Welt von Web Dynpro bringen. Zunächst werden Sie hier mit den Begriffen Component, View, Web Dynpro-Anwendung und Bildschirmelement (UI-Element) konfrontiert. Die Anwendung der weiteren Konzepte folgt später, wenn Sie mehr über deren Verwendung erfahren haben.

In dieser Übung soll eine einfache „Hello World“-Web Dynpro-Anwendung entwickelt werden. Sie werden hierfür zunächst eine Component anlegen, dann eine View mit einem entsprechenden UI-Element zur Anzeige des Textes „Hello World“ erstellen und schließlich eine Anwendung erzeugen, damit über eine URL auf die Component zugegriffen werden kann.

Die Entwicklung von Web Dynpro-Anwendungen ist in den Object Navigator integriert, so dass dieser auch den Einstieg in diese Übung darstellt. Öffnen Sie dort Ihr Paket **ZZ_####**. Klicken Sie im Navigationsbaum mit der rechten Maustaste auf das Paket im Navigationsbaum und wählen Sie den Pfad **Anlegen -> Web Dynpro -> Web-Dynpro-Component(-Interface)**.



Abbildung 14: Anlegen der Component: SAP-System-Screenshot

Nennen Sie die Component **ZZ_####_WD_HELLO**. Sie werden beim Anlegen auch nach einer Beschreibung gefragt, die Sie mit einem geeigneten Text füllen. Geben Sie außerdem als Window-Name **MAIN_WINDOW** an. So wird automatisch ein Window angelegt.

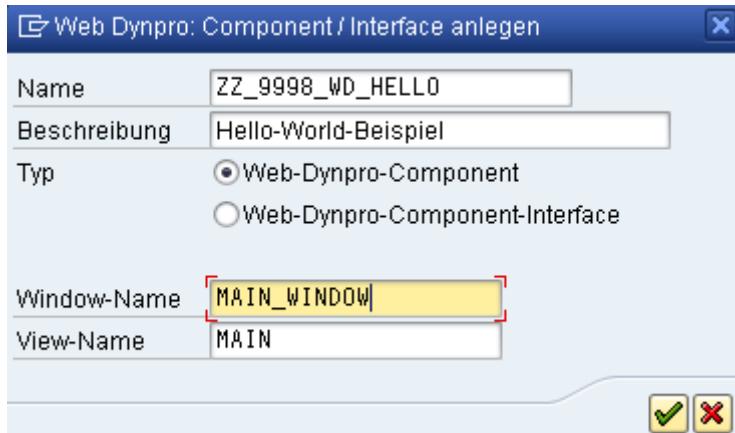


Abbildung 15: Anlegen der Component: SAP-System-Screenshot

Bestätigen Sie das Fenster und die gewohnten Nachfragen nach Paket und Transportauftrag. Klappen Sie dann den Knoten **Web Dynpro** im Navigationsbaum und alle Unterknoten auf. Sie sehen dort, dass automatisch weitere Elemente angelegt wurden:

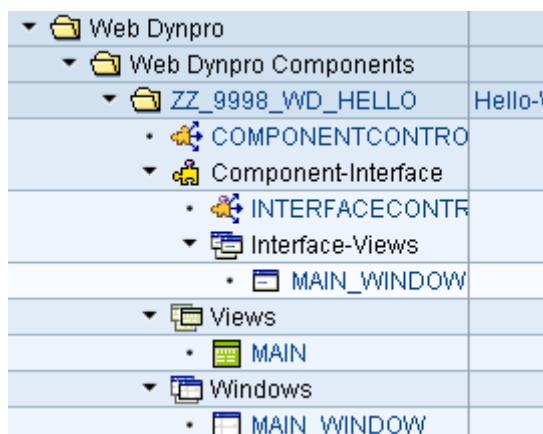


Abbildung 16: Automatisch angelegte Elemente: SAP-System-Screenshot

Hierbei handelt es sich um einen Component-Controller, einen Interface-Controller und eine Interface-View für das Window, das direkt beim Anlegen mit erzeugt wurde. Auch eine View ist bereits vorhanden, in dieser Übung soll aber ein eigenes View angelegt und in das Window eingefügt werden. Klicken Sie dafür mit der rechten Maustaste auf den Namen der Component im Navigationsbaum und wählen Sie aus dem Kontextmenü den Pfad **Anlegen -> View**. Sie werden daraufhin nach einem Namen und einer Beschreibung gefragt. Geben Sie als Namen **MAIN_VIEW** an und wählen Sie eine passende Beschreibung. Bestätigen Sie das anlegen.



Abbildung 17: Anlegen der View: SAP-System-Screenshot

Es wird automatisch die Layout-Registerkarte geöffnet, in der Sie das Bildschirmelement, das auf der View erscheinen soll, anlegen können. Sie werden hier nach Ihrem Benutzernamen

(USER#-###) und Ihrem Kennwort gefragt. Geben Sie diese Informationen ein. Sie sollten diese Ansicht sehen:

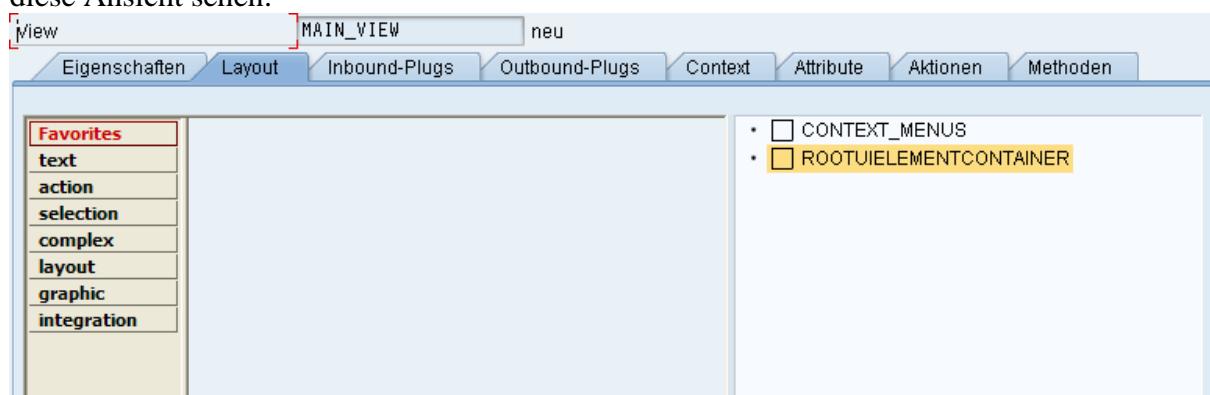


Abbildung 18: View-Layout: SAP-System-Screenshot

Sollte die Passwortabfrage nicht erscheinen bzw. der linke Bereich (Favorites, text, action etc.) und der leere Bereich in der Mitte nicht sichtbar sein, ist möglicherweise die Layoutvorschau ausgeblendet. Dies können sie mit dem Button Layout-Vorschau ein-/ausblenden korrigieren. Eine weitere Fehlerquelle könnte eine veraltete Version des Internet Explorers sein.

Klicken Sie nun mit der rechten Maustaste auf **ROOTUIELEMENTCONTAINER** im oberen rechten Bereich des Fensters (der UI-Elementhierarchie), und wählen Sie **Element einfügen** (siehe folgende Abbildung).

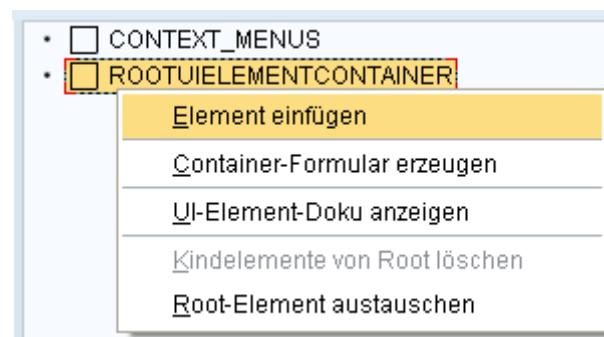


Abbildung 19: Einfügen eines UI-Elements: SAP-System-Screenshot

Im darauffolgenden Fenster werden Sie nach **ID** und **Typ** gefragt. Bei der ID handelt es sich um einen Namen für das Bildschirmelement. Als lesbarer Name empfiehlt sich **TEXT_VIEW_1**, da es sich um ein TextView-Element handeln soll und es sich um das erste solche Element auf der View handelt. Als Typ wählen Sie aus der Drop-Down-Liste **TextView** aus. Sie sehen, dass dort eine Vielzahl von UI-Elementen zur Auswahl steht. Sie werden einige weitere Elemente im Laufe dieses Kapitels noch kennenlernen.

Nachdem Sie bestätigt haben, erscheint das neue UI-Element unterhalb von **ROOTUIELEMENTCONTAINER**. Weiterhin sind dessen Eigenschaften im rechten unteren Bereich des Fensters sichtbar. Sollte dies nicht der Fall sein, genügt ein einfacher Klick auf den Namen des Elements. Suchen Sie dort die Eigenschaft **text** und geben Sie als Wert **Hello World** ein. Bestätigen Sie die Eingabe mit Enter. Sie sehen daraufhin im Hauptbereich des Fensters bereits eine Vorschau auf das Ergebnis:

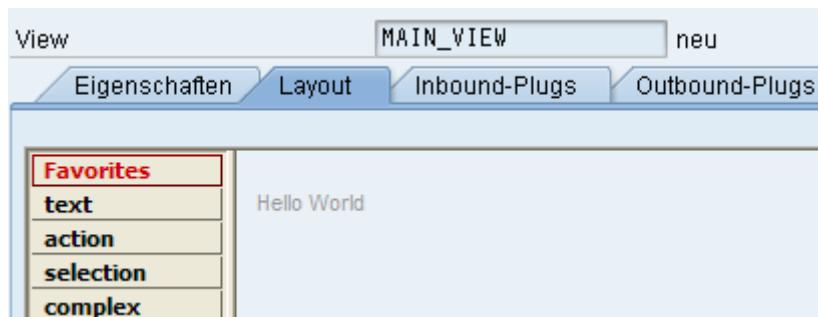


Abbildung 20: Anzeige des TextView-Elements: SAP-System-Screenshot

Die von Ihnen angelegte View muss noch zu einem Bestandteil des (automatisch angelegten) Window werden. Sichern Sie dafür die View und öffnen Sie per Doppelklick aus dem Navigationsbaum das Window **MAIN_WINDOW** (Achtung: Verwechseln Sie dieses nicht mit der gleichnamigen Interface-View). Öffnen Sie dort die Registerkarte **Window** und klappen Sie dort den Baum auf (in der Bildschirmmitte, mit **MAIN_WINDOW** beginnend). Sie sehen die View **MAIN** und ein Default-Inbound-Plug. Stellen Sie sicher, dass Sie sich im Änderungsmodus befinden und klicken Sie in der Baumstruktur auf der Registerkarte mit der rechten Maustaste auf den Window-Namen **MAIN_WINDOW**. Wählen Sie aus dem Kontextmenü **View einbetten**. Es erscheint das folgende Fenster:



Abbildung 21: Einbetten einer View: SAP-System-Screenshot

Geben Sie dort in das Feld **Einzubettende View** über die F4-Hilfe den Wert **MAIN_VIEW** ein. Bestätigen Sie das Fenster und sichern Sie.

Die vom System angelegte View ist derzeit noch als Default-View gesetzt. Klicken Sie daher im Baum der Window-Struktur mit der rechten Maustaste auf die von Ihnen angelegte View und wählen Sie **Als Default setzen**. Speichen, prüfen und aktivieren Sie anschließend die Component.

Sie haben nun die benötigte Component fertiggestellt. Es fehlt aber noch eine Anwendung, mit der eine im Browser aufrufbare URL als Einstieg verbunden ist. Klicken Sie daher im Navigationsbaum des Object Navigators mit der rechten Maustaste auf den Knoten **ZZ_###_WD_HELLO** (unter **Web-Dynpro-Components**) und wählen Sie aus dem Kontextmenü **Anlegen -> Web-Dynpro-Anwendung**. Geben Sie als Namen **ZZ_###_WD_HELLO** an und ergänzen Sie eine geeignete Beschreibung. Bestätigen Sie das Fenster. Eine Frage nach einem Administrations-Service können Sie ggf. mit „nein“ beantworten (dies gilt auch für alle Anwendungen in den weiteren Übungen).

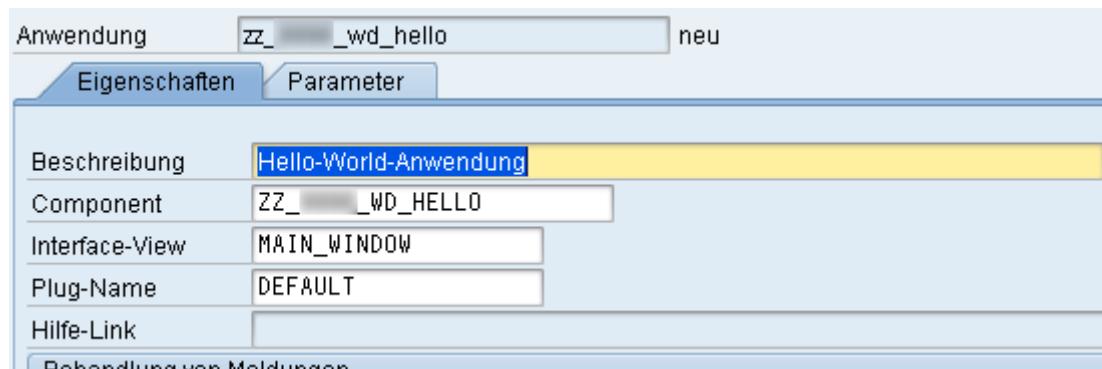


Abbildung 22: Die WD-Anwendung: SAP-System-Screenshot

Sie sehen, dass die Einstellungen für Component, Interface-View und Plug-Name vom System bestimmt wurden (siehe obige Abbildung). Sollte dies nicht der Fall sein, haben Sie möglicherweise im Navigationsbaum den rechtsklick nicht auf der Component ausgeführt. Speichern Sie die Anwendung. Bestätigen Sie wie gewohnt Paket und Transportauftrag.

Testen Sie nun die Anwendung. Es öffnet sich hierfür ein Browserfenster (evtl. müssen Sie dies in einer Sicherheitsabfrage erst erlauben). Sie müssen sich dabei möglicherweise neu anmelden, indem Sie auf **Anmelden** klicken und im Pop-Up Ihren Benutzernamen und Ihr Kennwort eingeben. Danach erscheint Ihre Anwendung mit dem von Ihnen angelegten Textelement:

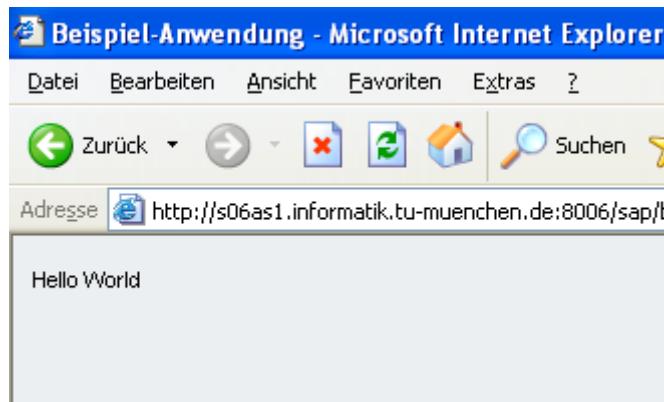


Abbildung 23: Die Hello-World-Anwendung im Browser

7.5 Web Dynpro-Controllers

Sie haben nun eine erste Anwendung geschrieben, die jedoch noch keinerlei Navigationsmöglichkeiten aufweist. In diesem Unterkapitel werden Sie näheres zu Controller erfahren, um mithilfe von Plugs und der zugehörigen Ereignisbehandlung die Ablauflogik einer Web Dynpro-Anwendung implementieren zu können.

Es werden vier grundlegende Typen von Controller in einer Component unterschieden:

- Component-Controller
- Custom-Controllers (inkl. Configuration-Controller)
- View-Controllers
- Window-Controllers

Der **Component-Controller** ist nur einmal pro Component vorhanden und global für alle anderen Controller der Component sichtbar. Im Component-Controller wird die Funktionalität abgebildet, die die gesamte Component betrifft. Er besitzt keine visuelle Schnittstelle.

Custom-Controllers können in beliebiger Anzahl zur Design-Zeit definiert werden. Sie werden verwendet, um Funktionalität aus dem Component-Controller auszugliedern. Zur Laufzeit werden die Custom-Controller vom Framework automatisch instanziert. Da bei dieser Instanziierung keine Reihenfolge vorgegeben ist, sollte ein Custom-Controller nicht von der Existenz eines anderen Custom-Controllers abhängig sein. Ein Spezialfall der Custom-Controller ist der **Configuration-Controller**. Dieser unterscheidet sich von den andern Custom-Controllern darin dass es nur einen Configuration-Controller in einer Component geben kann. Er wird verwendet, wenn die Component spezielle Konfigurations- und Personalisierungsfunktionen implementiert. Auf den Configuration-Controller kann von den anderen Controllern zugegriffen werden, er selbst hat jedoch keinen Zugriff auf andere Controller.

View-Controllers enthalten Funktionalität, die sich unmittelbar auf eine View beziehen, etwa Prüfungen der Benutzereingaben und deren Verarbeitung. Jede View besteht neben ihrem Layout aus genau einem View-Controller.

Window-Controllers stellen Methoden bereit, die von den Inbound-Plug-Methoden des Window verwendet werden können. Der Controller kann hier für die übergebenen Daten zuständig sein. Zu einem Window gehört stets genau ein Window-Controller.

7.5.1 Allgemeine Elemente von Controllers

Es gibt einige grundlegende Elemente (auch „Entitäten“), die in allen Typen von Controllern übereinstimmen. Diese sind der Context, Hook-Methoden, Instanzmethoden und Attribute. Weiterhin können alle Controller Verwendungsbeziehungen deklarieren.

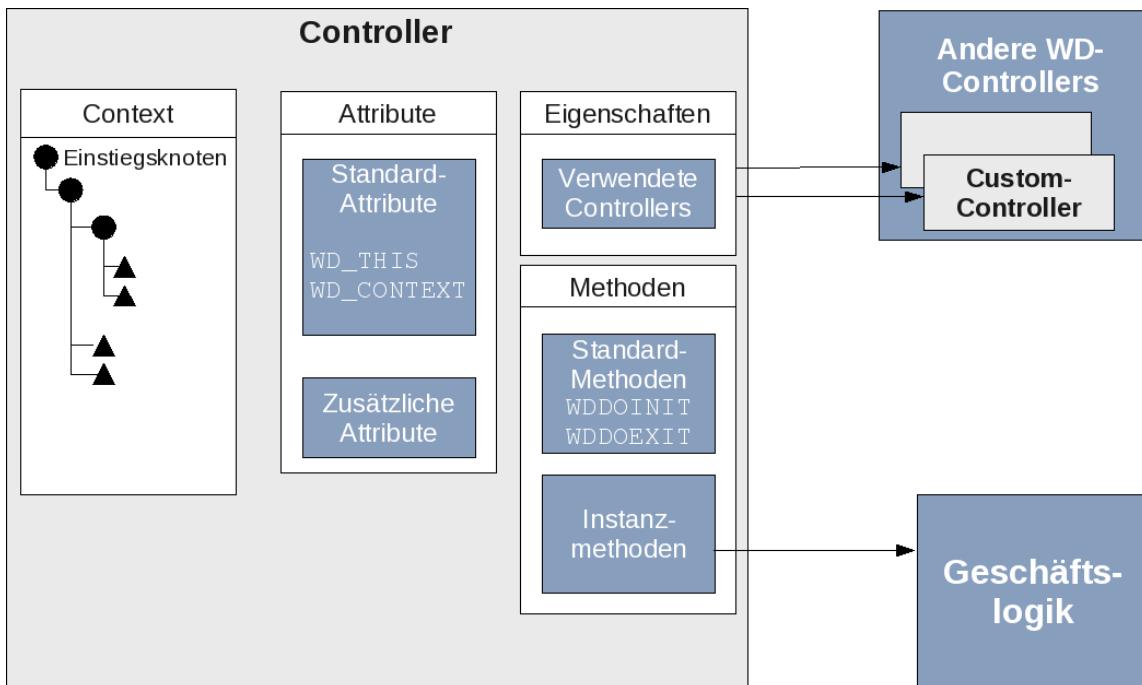


Abbildung 24: Allgemeiner Aufbau eines Controllers

Jeder Controller hat einen **Context**. In dessen hierarchischer Struktur ist zu Beginn nur der Einstiegsknoten vorhanden. Alle weiteren Knoten und Attribute müssen erst angelegt werden. Dies geschieht entweder statisch mit den Werkzeugen der ABAP Workbench, oder aber dynamisch durch entsprechende Anweisungen in Quellcode.

Attribute ermöglichen die Ablage von Daten, die sich nicht unmittelbar auf Bildschirmelemente beziehen. Sie werden auf der Registerkarte Attribute definiert und können anschließend von allen Methoden des Controllers erreicht werden. Durch das Framework werden zwei spezielle Attribute vordefiniert. Mit dem Attribut **WD_THIS** kann auf Funktionen des Controllers zugegriffen werden, mit dem Attribut **WD_CONTEXT** auf die des Contexts.

Es gibt zwei Arten von Methoden, die in Controllern vorkommen. Zum einen können beliebige herkömmliche **Instanzmethoden** definiert werden. Zum anderen gibt es vom System vordefinierte **Standard-Hook-Methoden**. Letztere werden vom System automatisch zu bestimmten Zeitpunkten im Rahmen der Verarbeitung aufgerufen. Die Hook-Methode **WDDOINIT** führt das System beim Anlegen der Controller-Instanz aus, die Methode **WDDOEXIT** hingegen beim Löschen der Instanz. Weitere Aufrufe der Methoden finden nicht statt.

Durch die Deklaration einer Verwendungsbeziehung wird der Austausch von Informationen zwischen zwei Controllers ermöglicht. Die Deklaration wird in den Eigenschaften des verwendenden Controllers vorgenommen. Als verwendeter Controller sollte nie ein View-Controller angegeben werden, da dieser gemäß MVC nur für die Darstellung von Daten (sowie die Interaktion mit dem Anwender) genutzt werden soll. Die Generierung dieser Daten obliegt dem Custom-Controller.

7.5.2 Elemente von Component- und Custom-Controllers

Die folgende Abbildung zeigt einen Component- oder Custom-Controller mit seinen speziellen Elementen.

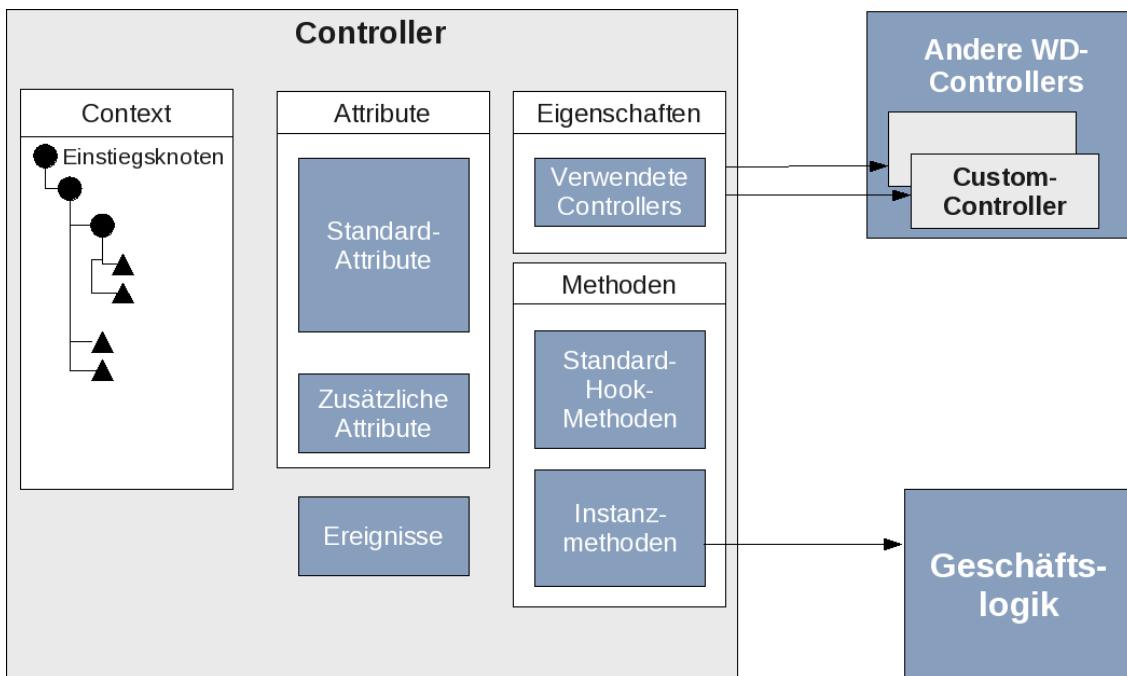


Abbildung 25: Elemente eines Component- oder Custom-Controllers

Für beide Typen von Controllern können Ereignisse definiert werden. Wie gewohnt können diese Ereignisse Parameter besitzen, um zusätzliche Informationen zu übergeben. Als Behandlermethode kommt jede als Behandlermethode deklarierte Methode anderer Controllers infrage. Dies kann insbesondere auch eine Methode eines View-Controllers sein. So kann durch das Ereignis die Bearbeitung in einem View-Controller angestoßen werden, nachdem die Bearbeitung im Component-Controller abgeschlossen ist. Das Web Dynpro-Framework bietet hier eine weitreichende Unterstützung: Die Definition, das Auslösen und das Registrieren von Behandlern erfolgen automatisch anhand der Deklarationen zur Design-Zeit. Elemente des Context, Methoden, Attribute und Ereignisse können als Schnittstellelemente gekennzeichnet werden. Andere Components können dann über den Interface-Controller auf diese Elemente zugreifen.

Der Component-Controller (nicht aber die Custom-Controllers) verfügt über zwei zusätzliche Standard-Hook-Methoden. Die Methode **WDDOBEFORENAVIGATION** wird vor Abarbeiten des Navigationsstapels, die Methode **WDDOPOSTPROCESSING** nach der Bearbeitung aller zu sendenden Views bzw. View-Kompositionen ausgeführt.

7.5.3 Elemente von View-Controllers

View-Controller werden für die grafische Interaktion mit dem Benutzer verwendet. Sie übernehmen Aufgaben zur Darstellung von Inhalten und Aufgaben im Zusammenhang mit der Benutzerinteraktion.

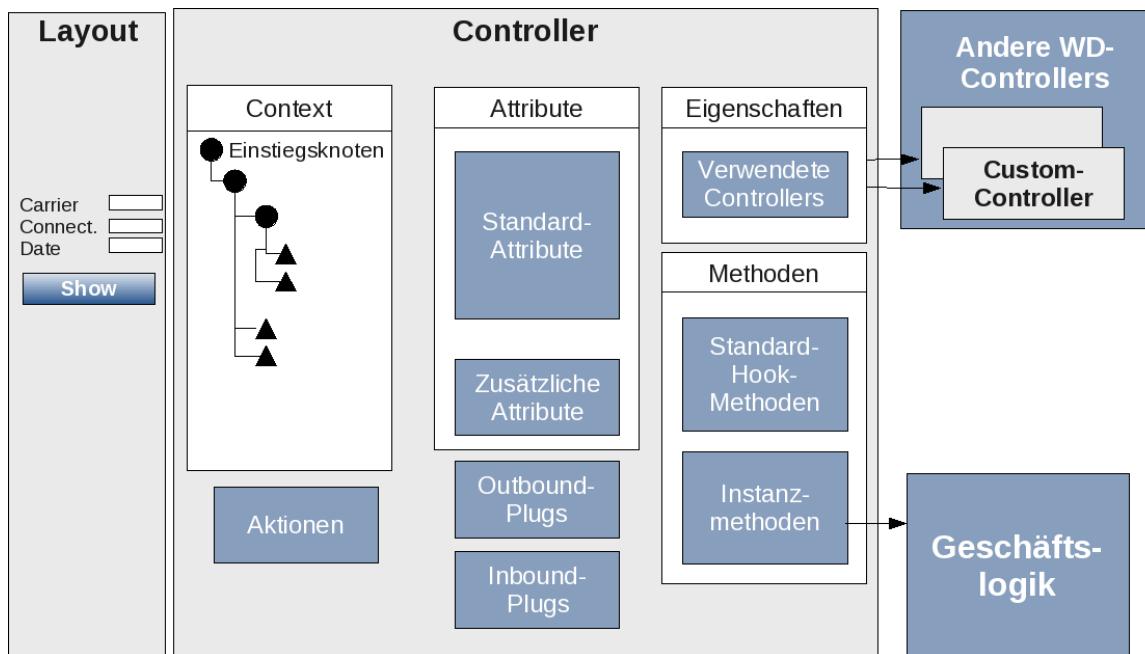


Abbildung 26: Elemente eines View-Controllers

Die Navigation zwischen den einzelnen View-Controllers wird durch **Navigations-Plugs** realisiert. Hierbei handelt es sich um spezielle Navigationsereignisse und -behandler. Unterschieden wird zwischen Outbound-Plugs und Inbound-Plugs.

Ein Navigationsereignis wird durch Feuern eines **Outbound-Plugs** ausgelöst. Ein **Inbound-Plug** ist hingegen ein Ereignisbehandler, der auf ein Navigationsereignis registriert werden kann. Das System erzeugt automatisch Methoden im Component-Controller, mit denen die Ereignissteuerung umgesetzt wird. Bei Deklaration eines Outbound-Plugs handelt es sich um die Methode **FIRE_<Outbound-Plug>_PLG**, wobei **<Outbound-Plug>** für den Namen des Outbound-Plugs steht. Diese wird nur vom Framework verwendet und ist für den Entwickler nicht sichtbar. Die Deklaration eines Inbound-Plugs führt zur Generierung einer Methode **HANDLE_<Inbound-Plug>**, wobei hier **<Inbound-Plug>** für den Namen des Inbound-Plugs steht. Die statische Registrierung eines Inbound-Plug über diese Methode auf das durch den Outbound-Plug ausgelöste Navigationsereignis wird **Navigationslink** genannt. Navigationslinks sind nicht Teil der Views, sondern Teil eines Window. Die Navigation kann sich somit bei zwei Windows unterscheiden, auch wenn diese dieselben Views einbetten.

Um tatsächlich Navigieren zu können, müssen Benutzeraktivitäten wie etwa das Klicken auf eine Schaltfläche verarbeitet werden. Hierzu dienen in der Web Dynpro-Architektur **Aktionen**. Diese verbinden ein solches Ereignis mit einer Behandlermethode. Diese Behandlermethode wird bei Definition einer Aktion automatisch generiert. Sie erhält den Namen **ONACTION<Aktion>**, wobei **<Aktion>** für den Namen der Aktion steht. Damit infolge der Aktion eine Navigation stattfindet, kann in dieser Methode der entsprechende Outbound-Plug gefeuert werden.

View-Controllers besitzen zwei spezielle Hook-Methoden:

- Wenn ein Client-Ereignis (z. B. durch Anklicken einer Schaltfläche) in einer View gefeuert wird, kommt die Methode **WDDOBEFOREACTION** zum Einsatz. Vor dem Aufruf der Ereignisbehandlermethode werden die **WDDOBEFOREACTION**-Methoden aller Views der letzten View-Komposition ausgeführt.
- Die Methode **WDDOMODIFYVIEW** wird verwendet, um aus dem Programm heraus dynamische Änderungen an der Oberfläche der View vorzunehmen bzw. Oberflächenelemente dynamisch zu definieren. Sie wird jeweils vor der Anzeige einer View aufgerufen.

Schließlich verfügen View-Controller noch über eine Referenz auf den Component-Controller. Das entsprechende Attribut hat den Namen **WD_COMP_CONTROLLER**.

7.5.4 Elemente von Window-Controllers

Auch Window-Controller besitzen spezielle Elemente. Sie sind technisch gesehen spezielle View-Controller, die über keine Oberfläche (View-Layout) verfügen. Die Views, die von einer Anwendung angezeigt werden sollen, müssen aber in einem Window-Controller eingebettet sein.

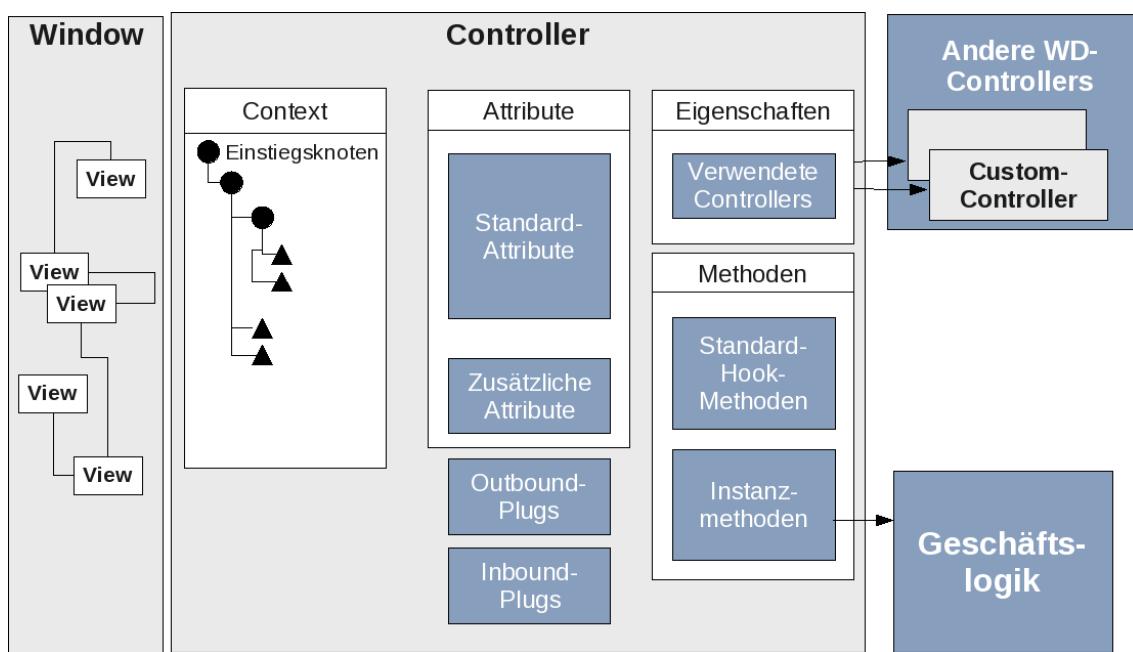


Abbildung 27: Elemente von Window-Controllers

Die Outbound- und Inbound-Plugs des Window-Controllers können verwendet werden, um Component-Übergreifend zu navigieren. Dafür müssen diese die Eigenschaft Interface zugewiesen bekommen, so dass sie Teil des Interface-Views werden.

Das Attribut **WD_COMP_CONTROLLER** ist analog zu View-Controllern auch in Window-Controllern vorhanden.

7.5.5 Praxis: Übung zur Navigation zwischen Views

In der vorangegangenen Übung haben Sie eine „Hello World“-Web Dynpro-Anwendung erstellt. Diese enthielt bereits zwei Views, jedoch wurde nur die View mit dem „Hello World“-Text angezeigt, die Sie als Default-View festgelegt haben. In dieser Übung werden

Sie nun die Navigation zwischen Views durch Verwendung von Inbound- und Outbound-Plugs praktisch anwenden.

Legen Sie für diese Übung eine neue Web Dynpro-Component mit dem Namen **ZZ_####_WD_NAVI** an. Diese soll ein Window **MAIN_WINDOW** enthalten. Dieses können Sie direkt beim Anlegen der Component anlegen, zusammen mit einer View **VIEW1**. Bestätigen Sie Paket und Transportauftrag, und legen Sie in der Component eine weitere View **VIEW2** an.

Legen Sie auf beiden Views ein **TextView**-Element an, mit dem Sie jeweils anzeigen, ob es sich um die erste oder zweite View handelt. Betten Sie die zweite View anschließend auch in das Window ein (analog zur vorangegangenen Übung) und stellen Sie sicher, dass **VIEW1** im Window als Default gesetzt ist.

In dieser Übung soll nun eine Navigation zwischen diesen beiden Views entwickelt werden. Dafür werden Schaltflächen auf den Views sowie Outbound- und Inbound-Plugs benötigt. Öffnen Sie zunächst die erste View und wechseln Sie zur Registerkarte **Outbound-Plugs**. Fügen Sie dort einen Outbound-Plug **TO_VIEW2** hinzu, indem Sie einen Eintrag in die Tabelle vornehmen und pflegen Sie dabei auch eine passende Beschreibung, wie in der folgenden Abbildung dargestellt.

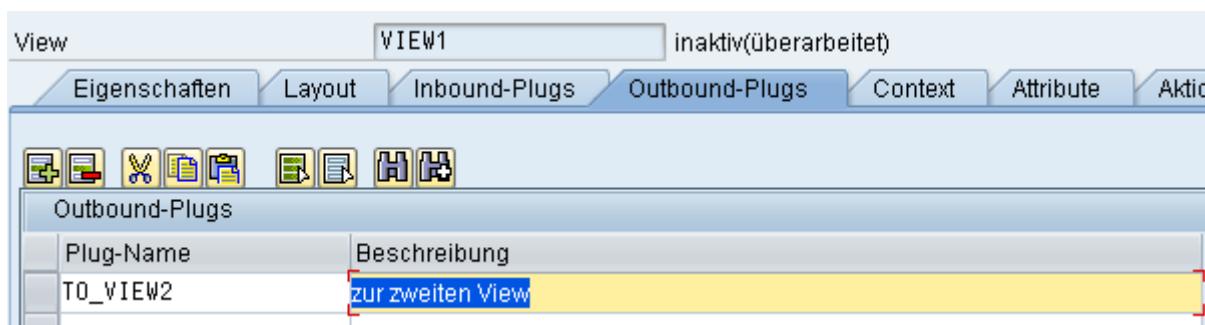


Abbildung 28: Anlegen eines Outbound-Plugs: SAP-System-Screenshot

Fügen Sie analog auf der Registerkarte **Inbound-Plugs** ein Inbound-Plug mit dem Namen **FROM_VIEW2** hinzu. Sichern Sie die View. Sie sehen auf der Registerkarte mit den Inbound-Plugs eine zusätzliche Spalte **Ereignisbehandler**. Hier wird die entsprechende Methode angezeigt, die vom System zur Behandlung angelegt wurde. Wechseln Sie zur Registerkarte **Methoden**. Sie sehen auch dort die neue Methode.

Methode	Methoden...	Beschreibung	Ereignisbehandler
HANDLEFROM_VIEW2	Ereignisbehandler	vom zweiten View	

Abbildung 29: Automatisch angelegte Methode: SAP-System-Screenshot

Sie haben nun die benötigten Plugs für eine der Views angelegt. Öffnen Sie nun die zweite View und fügen Sie dort analog Plugs hinzu. Der Outbound-Plug soll **TO_VIEW1** heißen, der Inbound-Plug **FROM_VIEW1**. Gehen Sie wie bei der ersten View vor und vergessen Sie nicht, die View zu sichern.

Es haben nun beide Views die benötigten Plugs. Es muss nun aber noch eine Verbindung in Form von Navigationslinks zwischen den Plugs hergestellt werden.

Öffnen Sie dazu das Window. Falls Sie die zweite View noch nicht eingebettet haben, können Sie neben der in der letzten Übung verwendeten Vorgehensweise auch „Drag’n‘Drop“ verwenden. Ziehen Sie die View hierfür mit der Maus aus dem Navigationsbaum des Object Navigators auf den Eintrag **MAIN_WINDOW** in der Window-Struktur. Das Systembettet die View nun ein. Setzen Sie anschließend die erste View als Default. Klappen Sie dann die baumartige Window-Struktur komplett aus, wie in der folgenden Abbildung:

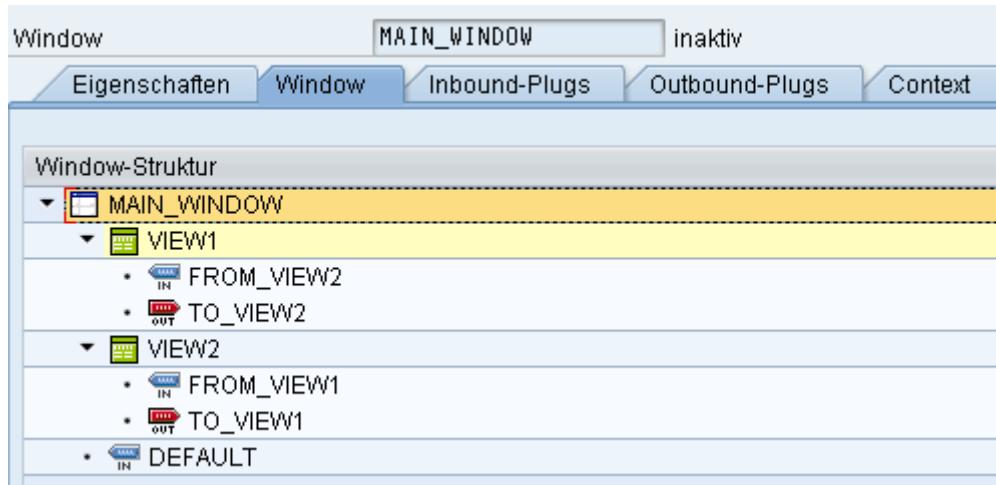


Abbildung 30: Ausgeklappte Window-Struktur: SAP-System-Screenshot

Zur Verbindungsherstellung kann nun ebenfalls „Drag’n‘Drop“ zum Einsatz kommen. Ziehen Sie dazu in der Window-Struktur **TO_VIEW2** auf **FROM_VIEW1**. Bestätigen Sie das Fenster zur Auswahl des Navigationsziels.



Abbildung 31: Auswahl des Navigationsziels: SAP-System-Screenshot

Damit später zwischen beiden Views hin- und hergeschaltet werden kann, sorgen Sie auch für einen Navigationslink vom Outbound-Plug der zweiten View zum Inbound-Plug der ersten View. Sichern Sie anschließend das Window.

Sie haben nun den Navigationslink definiert. Es fehlen jedoch noch Schaltflächen (Buttons), mit denen auf der jeweiligen View der entsprechende Outbound-Plug gefeuert werden kann. Öffnen Sie daher nun die erste View und fügen Sie zu dieser ein UI-Element des Typs **Button** hinzu. Das Vorgehen hierzu entspricht dem Vorgehen zum Einfügen der Textelemente.

Ändern Sie danach den Text des Buttons in **Zur zweiten View**. Die entsprechende Einstellung finden Sie im rechten unteren Bereich, wenn der Button ausgewählt ist.

Eigenschaft	Wert	Bind...
Eigenschaften (Button)		
Id	BUTTON1	
activateAccessKey		<input type="checkbox"/>
contextMenuBeha...	inherit	<input type="button"/>
contextMenuId		
design	standard	<input type="button"/>
enabled	<input checked="" type="checkbox"/>	<input type="button"/>
explanation		<input type="button"/>
hotkey	none	<input type="button"/>
imageFirst	<input checked="" type="checkbox"/>	<input type="button"/>
imageSource		<input type="button"/>
text	Zur zweiten View	<input type="button"/>

Abbildung 32: Pflege des Button-Textes: SAP-System-Screenshot

Damit beim Anklicken des Buttons durch den Anwender auch etwas geschieht, muss nun eine Aktion zugewiesen werden. Dies geschieht ebenfalls im Eigenschaftsbereich des Buttons. Sie finden dort eine Zeile mit der Eigenschaft **onAction**. Klicken Sie in dieser Zeile auf (in der Spalte **Binding**). Wählen Sie im daraufhin erscheinenden Fenster einen Namen für die Aktion (erlaubt sind alphanumerischen Zeichen und Unterstriche) und pflegen Sie eine passende Beschreibung. Als Outbound-Plug wählen Sie das von Ihnen erstellte Outbound-Plug der ersten View aus. Bestätigen Sie dann die Eingaben.

Wechseln Sie dann zur Registerkarte **Aktionen**. Sie sehen dort die soeben von Ihnen definierte Aktion und einen Ereignisbehandler, den Sie wiederum auf der Registerkarte **Methoden** wiederfinden.

Sichern Sie die View und fügen Sie auch auf der zweiten View einen Button ein. Der Text des Buttons auf der zweiten View soll **Zur ersten View** lauten. Erstellen Sie analog eine entsprechende Aktion und sichern Sie auch diese View. Öffnen Sie dann die Component, prüfen Sie wie gewohnt und aktivieren Sie alle noch nicht aktvierten Elemente.

Erzeugen Sie als nächstes wie in der ersten Web Dynpro-Übung eine Anwendung mit dem gleichen Namen wie dem der Component. Testen Sie die Anwendung anschließend. Nach dem Login sollten Sie zwischen Ihren beiden Views hin- und her navigieren können:



Abbildung 33: Erste View im Browser



Abbildung 34: Zweite View im Browser

Sie sind nun in der Lage, mit Plugs eine Navigation zwischen verschiedenen Views zu definieren. Im nächsten Unterkapitel werden Sie erfahren, wie Daten in den Views angezeigt werden können.

7.6 Handhabung von Daten mit Contexts

Bislang werden auf Ihren Views nur Daten angezeigt, die Sie statisch als Textelemente bei der Gestaltung der Views festgelegt haben. In einem realistischen Szenario werden aber dynamische Daten und ein Austausch dieser Daten benötigt. Zu diesem Zweck werden Contexts verwendet, die bereits bei der Vorstellung der Web Dynpro-Architektur Erwähnung fanden. Es handelt sich hierbei um eine hierarchische Datenablage. Zum Austausch der Daten müssen die Variablen im Controller-Context definiert werden. Der Controller besitzt genau einen Context, der während der Lebensdauer des Controllers existiert und dessen Inhalt nach Beenden der Instanz des Controllers wieder verloren geht. Die Contexts anderer Controller, insb. von View Controllern, können durch Context-Mapping auf diese Daten zugreifen. Beachten Sie, dass View Controller ihre Contextdaten nicht anderen Controllern für die gemeinsame Nutzung zur Verfügung stellen können.

Der Context ist aus **Knoten** und **Attributen** zusammengesetzt. Hieraus wird eine Baumstruktur aufgebaut. In diesem Baum gibt es genau einen übergeordneten Knoten, der als **Kontexteinstiegsknoten** bezeichnet wird. Dieser Knoten kann nicht manuell angelegt werden, sondern ist mit festen Eigenschaften automatisch im jeweiligen Context vorhanden und kann nicht bearbeitet oder gelöscht werden. Es können jedoch selbstverständlich untergeordnete Elemente angelegt werden.

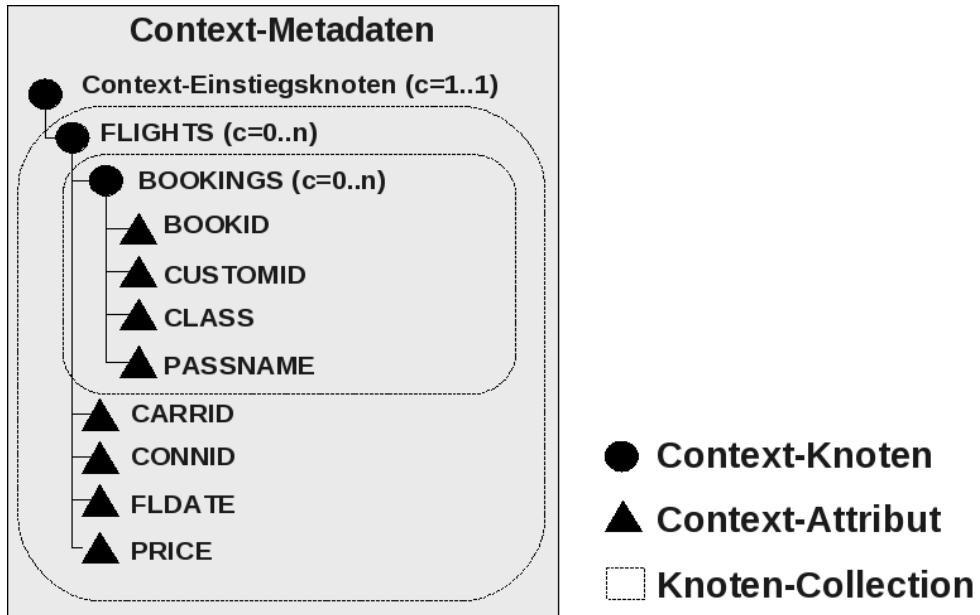


Abbildung 35: Aufbau eines Contexts

Ein **Kontextknoten** ist die Hauptabstraktionsklasse für die Datenablage. Er kann weitere Knoten oder Attribute enthalten. Er kann daher als eine Collection bezeichnet werden. Die unmittelbar untergeordneten Entitäten (Knoten, Attribute) eines Knotens werden im sogenannten **Element** zusammengefasst. Durch die Angabe einer Kardinalität wird festgelegt, wie viele Elemente eine Knoten-Collection zur Laufzeit enthalten kann. Ein **Kontextattribut** hat im Gegensatz zum Knoten keine untergeordneten Elemente. Jedes Kontextattribut muss Unterelement eines Knotens sein. Dies kann der Kontexteinstiegsknoten oder ein beliebiger anderer Knoten im Context sein. Über diese Konzepte können strukturierte Datenobjekte im Context abgelegt werden. Dafür muss ein Kontextknoten für die Collection selbst angelegt werden (z. B. FLIGHTS in der obigen Abbildung).

Beachten Sie: Strenggenommen werden hier zwei verschiedene Dinge als „Knoten“ bezeichnet: Einerseits ein zur Design-Zeit beschriebenes Element, andererseits dessen Instanz zur Laufzeit. Aus Gründen der Vereinfachung gibt es hierfür nur einen Begriff.

Die Elemente des strukturierten Datenobjekts sind dann als Unterelemente des Knotens zu definieren, wie etwa BBOKINGS oder CARRID in der obigen Abbildung. Die Kardinalität gibt dann die zulässige Anzahl der Einträge zur Laufzeit an. Diese Eigenschaft wird wie auch die gesamte Metadatenstruktur des Contexts zur Design-Zeit festgelegt. Durch die Kardinalität bzw. die Collection-Eigenschaft eines Knotens können zur Laufzeit mehrere Instanzen untergeordneter Knoten und Attribute existieren. Im obigen Beispiel könnte es zur Laufzeit etwa mehrere Flüge (FLIGHTS) geben, die wiederum jeweils mehrere Buchungen enthalten (BOOKINGS).

Die Kardinalität wird über ein Minimum und ein Maximum festgelegt. Das Minimum gibt die minimale Anzahl, das Maximum die maximale Anzahl von Elementen an, die die Collection zur Laufzeit enthalten darf. Für das Minimum kann 0 oder 1 angegeben werden, für das Maximum 1 oder n, wobei n für eine beliebige Anzahl steht. Daraus ergeben sich vier zulässige Kardinalitäten:

- 0..1: Kein oder ein Element
- 0..n: Beliebige Anzahl von Elementen (auch kein Element zulässig)
- 1..1: Genau ein Element
- 1..n: Beliebige Anzahl von Elementen, aber mindestens ein Element

Auch Knoten mit Maximal-Kardinalität 1 enthalten eine Element-Collection.

Der Kontexteinstiegsknoten besitzt die Kardinalität 1..1. Das System legt automatisch ein entsprechendes Standardelement an. Diese kann nicht gelöscht werden, da sonst die Kardinalität verletzt werden würde. Analog erhalten Sie Laufzeitfehler, wenn Sie bei anderen Knoten unzulässige Operationen durchführen: Das Standardelement eines Knotens mit Minimum-Kardinalität 1 darf nicht gelöscht werden, und einem Knoten mit Maximalkardinalität 1 darf kein zweites Element hinzugefügt werden.

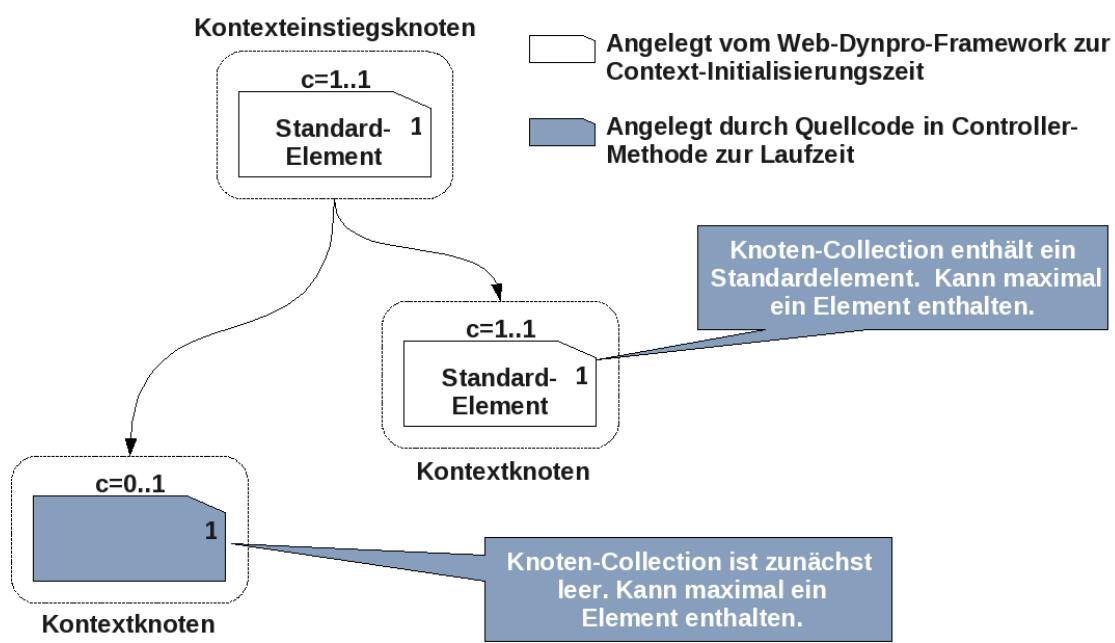


Abbildung 36: Auswirkungen der Knotenkardinalität

Jede Knoten-Collection, die dem Kontexteinstiegsknoten direkt untergeordnet ist, wird aufgrund der Kardinalität 1..1 des Kontexteinstiegsknoten automatisch angelegt. Wenn ihre Minimum-Kardinalität 0 ist, wird sie als leere Collection angelegt, wenn ihre Minimum-Kardinalität 1 ist, wird sie mit einem Standardelement angelegt. Diese Knoten werden daher als **unabhängige Knoten** bezeichnet: Sie existieren immer, im Gegensatz zu den anderen

Knoten-Collections, die nur dann existieren, wenn ihre übergeordnete Collection mindestens ein Element enthält. Letztere Knoten werden als **abhängige Knoten** bezeichnet.

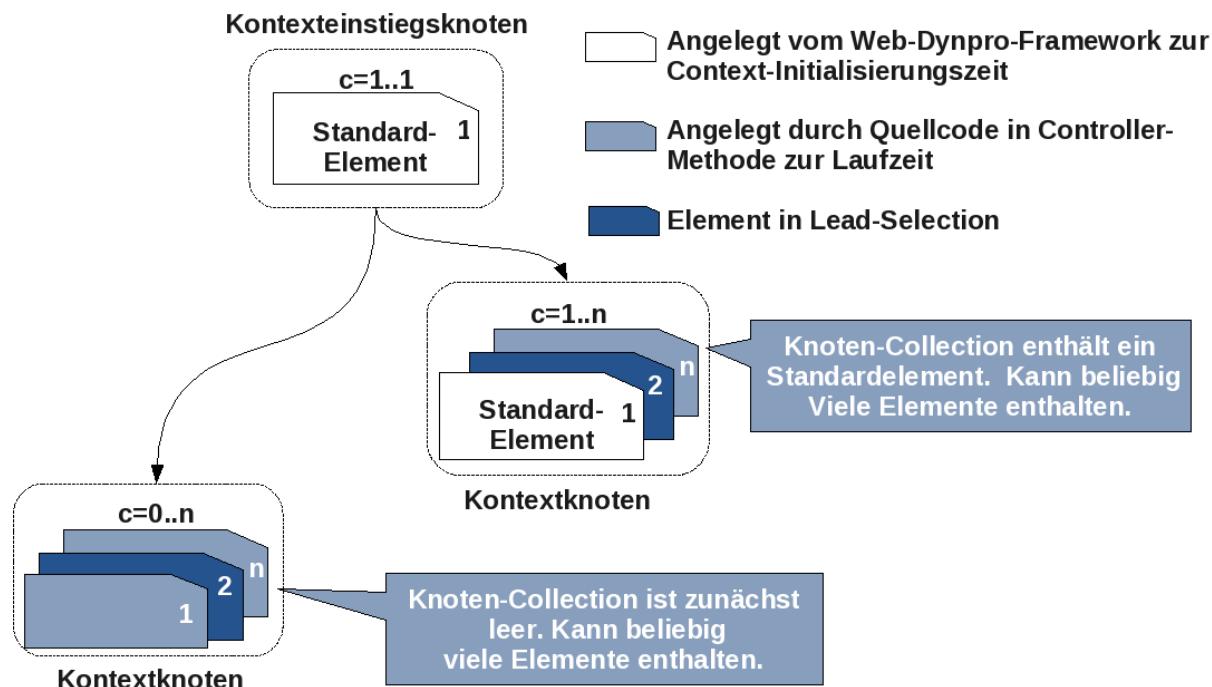


Abbildung 37: Knoten mit mehreren Elementen

Die obige Abbildung zeigt, wie sich in den Knoten zur Laufzeit mehrere Elemente befinden. Es handelt sich beim Context zur Laufzeit also nicht um eine flache, hierarchische Struktur wie zur Design-Zeit. Dieser Unterschied ist vergleichbar mit Internen Tabellen. Deren Aufbau wird über eine Struktur definiert, mit der zur Laufzeit dann durch die Einträge eine Tabelle aufgebaut wird. Im Unterschied zum Context besitzen Interne Tabellen jedoch keine Kardinalitätseigenschaft. Das bedeutet, dass eine Interne Tabelle grundsätzlich mit beliebig vielen Einträgen gefüllt werden kann, und umgekehrt kein Standardelement vorhanden sein kann, das bereits bei der Definition der Internen Tabelle vorhanden wäre. Es gibt somit sowohl Ähnlichkeiten als auch Unterschiede zwischen Internen Tabellen und Kontextknoten.

7.6.1 Lead Selection

In der obigen Abbildung sehen Sie auch Elemente, die als **Lead-Selection** hervorgehoben sind. Diese Eigenschaft bezieht sich auf den Zugriff auf die Collection eines Knotens. Auf die Elemente kann über einen Index zugegriffen werden. Dessen Zählung beginnt bei 1.

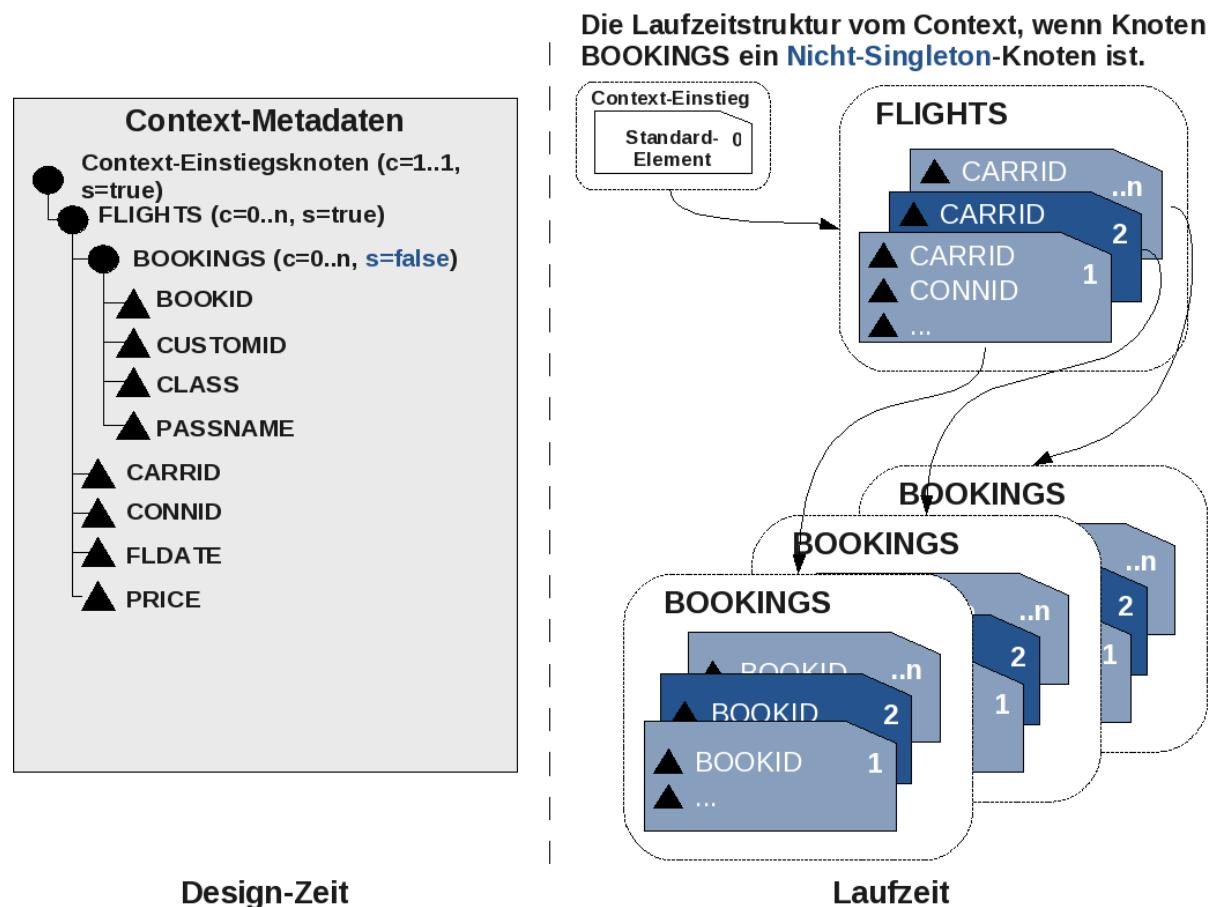
Zusätzlich kann ein Element der Collection als Lead-Selection-Element gekennzeichnet werden.

Als Lead-Selection wird entweder der Indexwert des entsprechenden Knotens gespeichert, oder die Konstante **IF_WD_CONTEXT_NODE=>NO_SELECTION**. Letzteres bedeutet, dass kein Element als Lead-Selection ausgewählt ist. Es ist möglich, das erste Element der Collection automatisch als Lead-Selection markieren zu lassen. Hierfür ist die Eigenschaft **Initialisierung Lead-Selection** zu aktivieren, die sich in den Knoteneigenschaften finden lässt. Neben der statischen Festlegung der Lead Selection kann diese auch zur Laufzeit gesetzt werden. Dies geschieht etwa durch das Markieren einer Zeile in einem Tabellen-View oder im Programmquelltext.

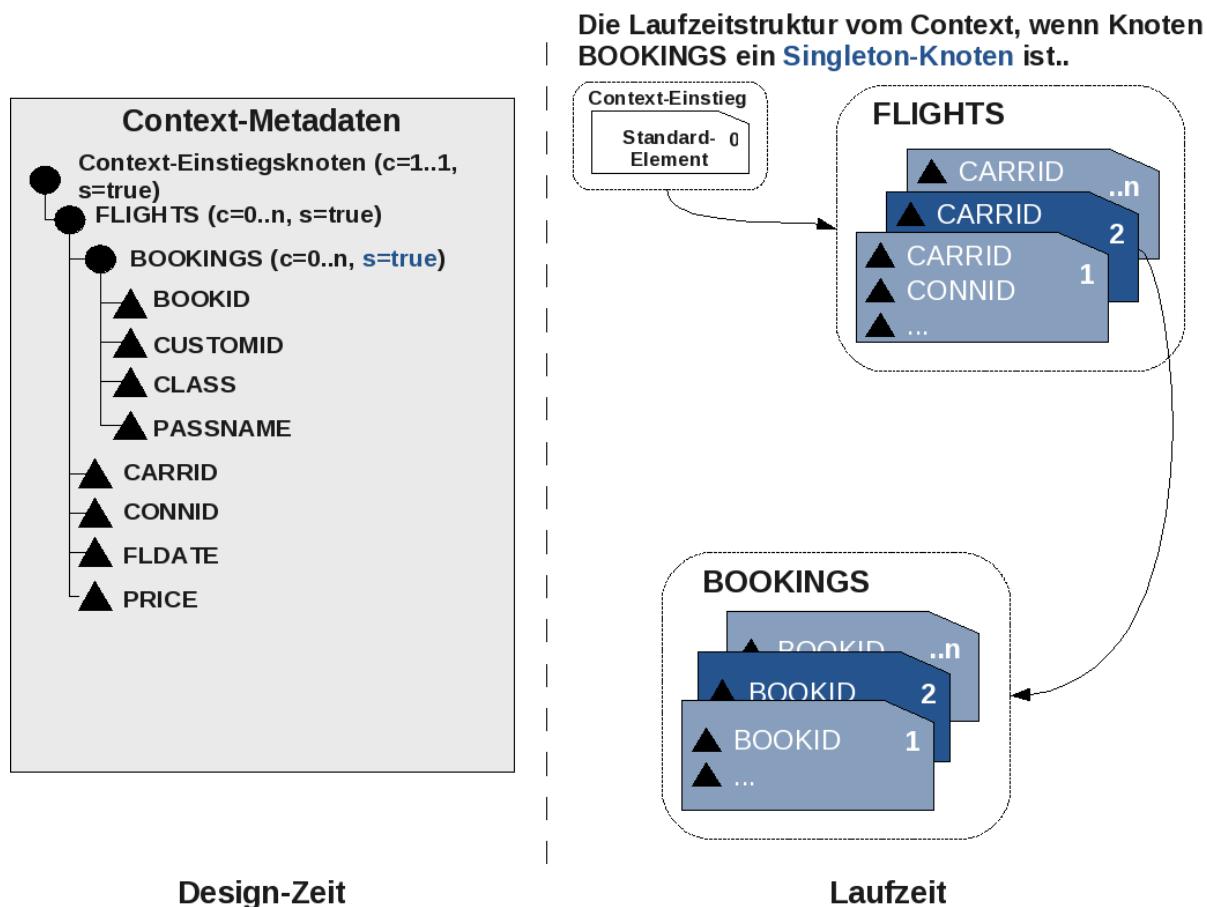
Durch setzen der Lead-Selection können UI-Elemente an Attribute des Elements gebunden werden. Weiterhin können im Code des Controllers spezielle Zugriffsmethoden benutzt werden.

7.6.2 Singleton-Knoten

Eine weitere Eigenschaft von Kontextknoten ist die **Singleton**-Eigenschaft. Sie wirkt sich auf die Beziehung zwischen einem abhängigen Knoten und dessen übergeordnetem Knoten aus.



In der obigen Abbildung ist **BOOKINGS** zur Design-Zeit als Nicht-Singleton-Knoten definiert worden. Dies hat zur Folge, dass zur Laufzeit jedem Element der Übergeordneten Knoten-Collection **FLIGHTS** eine eigene Collection **BOOKINGS** zugeordnet wird. Bei n Elementen im übergeordneten Knoten gibt es somit auch n unterschiedliche Instanzen des Nicht-Singleton-Knotens.



In der obigen Abbildung ist **BOOKINGS** zur Design-Zeit als Singleton-Knoten definiert worden. Dementsprechend gibt es zur Laufzeit nur eine einzige **BOOKINGS**-Collection, und nicht für jeden Flug eine separate Collection. Diese Einstellung ist der Standard.

7.6.3 Supply-Funktionen

Wird die Singleton-Eigenschaft eines Knotens aktiviert, wie im vorangegangenen Beispiel für den **BOOKINGS**-Knoten, ist sicherzustellen, dass sich die aktuell benötigten Daten im Knoten befindet. Befinden sich im Beispiel im Buchungsknoten die Daten zu einem Flug und wählt der Benutzer dann einen anderen Flug, muss der Buchungsknoten aktualisiert werden.

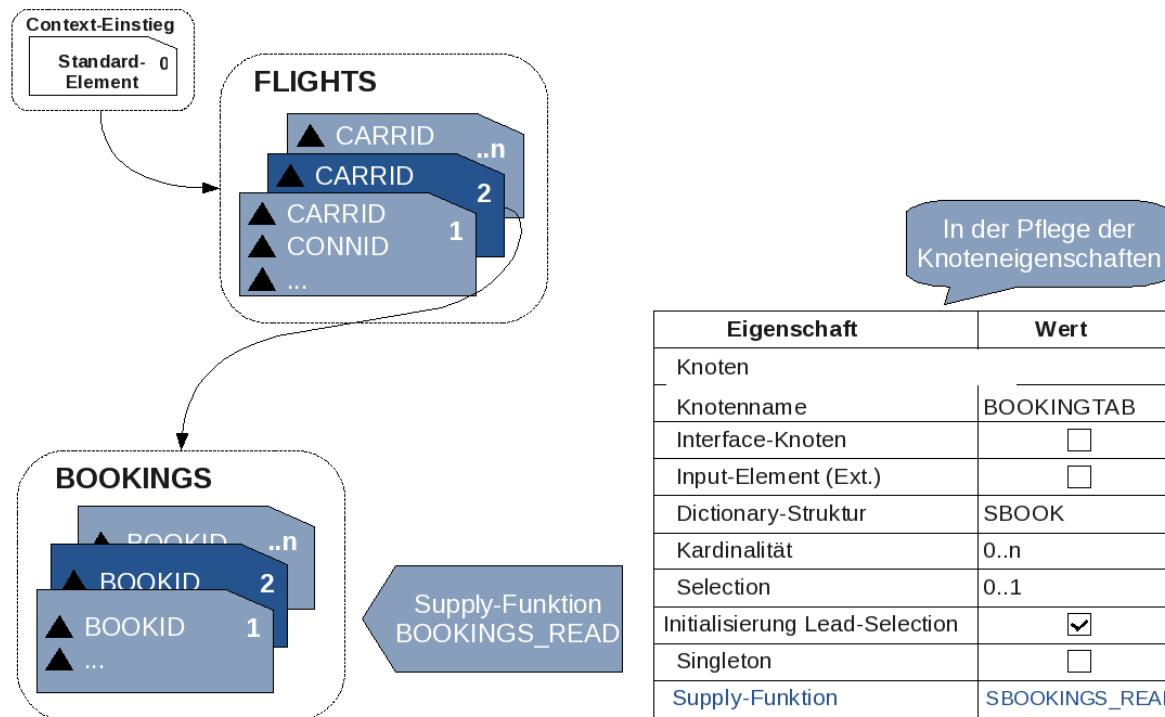


Abbildung 40: Verwendung einer Supply-Funktion

Die **Supply-Funktion** kommt zum Tragen, wenn ein Knoten neu mit Daten zu versorgen ist. Das ist auch außerhalb des Singleton-Knotens möglich: Die Funktion wird immer dann aufgerufen, wenn auf Daten eines **ungültigen Kontextknotens** zugegriffen wird. Ein Kontextknoten kann auf verschiedene Weisen ungültig werden:

- Die Knoten-Collection ist initial, es wurden also noch gar keine Daten eingefügt.
- Die Lead-Selection in der Übergeordneten Knoten-Collection wurde geändert (im Beispiel etwa durch Auswahl eines anderen Fluges). Dies ist ein besonders häufiger Anwendungsfall für Supply-Funktionen.
- Die Knoten-Collection wurde im Programmcode invalidiert.

Vereinfacht ausgedrückt ist ein Kontextknoten ungültig, wenn sich nicht die benötigten Daten darin befinden. Diese Daten können dann von der Supply-Funktion bereitgestellt werden. Bei der Supply-Funktion handelt es sich um eine automatisch angelegte Methode mit einer vorgegebenen Signatur. Diese umfasst zum einen eine Referenz auf das Lead-Selection-Element des übergeordneten Knotens, zum anderen eine Referenz auf den Knoten, der durch die Supply-Funktion mit Daten gefüllt werden soll. So können anhand der Referenz auf das Lead-Selection-Element die benötigten Daten durch die Supply-Funktion bestimmt werden, und durch die Referenz auf den zu füllenden Knoten an der benötigten Stelle hinterlegt werden.

Im Beispielszenario aus Flügen und Buchungen könnte es eine Vielzahl von Flügen geben, zu denen wiederum jeweils eine Vielzahl von Buchungen vorliegt. Eine Anwendung, die mit diesen Daten arbeitet, wird aber wahrscheinlich nicht alle Daten benötigen (ob diese Annahme zutrifft, ist natürlich für die jeweilige Anwendung zu klären). Würden beim Aufruf der Anwendung sofort alle Daten geladen, hätte dies klare Nachteile: Zum einen würde die Anwendung sehr langsam starten, da erst alle Daten gelesen werden müssten. Zum anderen würde unnötiger Datentransfer zur Datenbank verursacht und viel Speicher verbraucht. Diesen Umständen trägt die Web Dynpro-Architektur Rechnung. Anstatt wie hier angedeutet alle Daten sofort zu laden, werden Daten in Web Dynpro-Anwendungen dann geladen, wenn sie auch benötigt werden. Dieses Prinzip bezeichnet SAP als **Lazy-Data-Instanziierung**. Erst

beim ersten Zugriff auf einen untergeordneten Knoten würde dieser dann instanziert. Insbesondere in Kombination mit der Singleton-Eigenschaft kann so der Specheraufwand gering gehalten werden.

Für die einzelnen Elemente des übergeordneten Knotens werden bei der Lazy-Data-Instanziierung auch nicht sofort Collections des Abhängigen Knotens erzeugt. Erst beim Zuweisen der Lead-Collection erfolgt das Anlegen der Collection.

7.6.4 Context-Mapping

Durch Context-Mapping können die Daten eines Controllers in einem anderen Controller verfügbar gemacht werden. Der verwendende Controller ist in der Regel ein View-Controller. Im MVC-Konzept stellt der View Controller selbst seine Daten nicht für andere Controller zur Verfügung. Eine solche Mapping-Beziehung kann daher nicht definiert werden. Die anderen Controller würden dadurch vom View-Controller abhängig. Der View-Controller sollte lediglich Aufgaben der Präsentationsebene wahrnehmen, also die Daten darstellen und mit dem Benutzer interagieren, etwa durch das Validieren von Eingaben und die Ausgabe entsprechender Fehlermeldungen. Die Generierung der Daten gehört hingegen nicht in den View-Controller. Sie sollte in einem Custom-Controller untergebracht werden, der die Daten dann dem View-Controller zur Verfügung stellt.

Durch die Mappingbeziehung wird der Kontextknoten des verwendeten Controllers direkt referenziert. Es findet also kein Kopieren oder Verschieben der Daten statt. Der verwendende Controller besitzt auch keine eigene Collection für den abgebildeten Knoten. Die Mappingbeziehung setzt voraus, dass der verwendende Controller den Controller, dessen Daten er verwendet, in seinen Eigenschaften als verwendeten Controller deklariert.

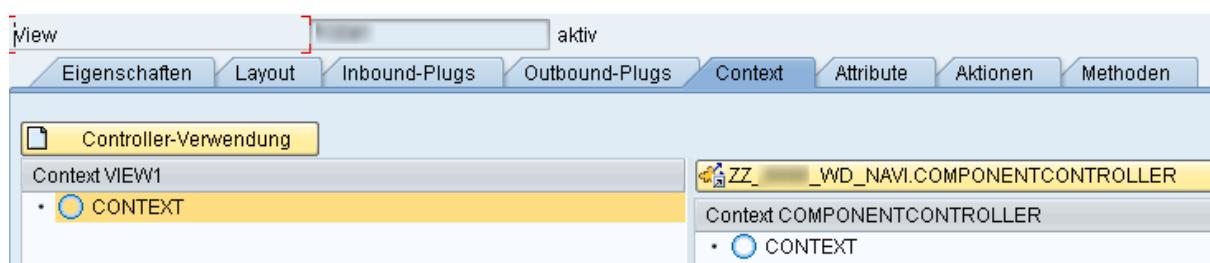


Abbildung 41: Context-Mapping: SAP-System-Screenshot

Auf der Registerkarte Context des verwendenden Controllers stehen daraufhin auf der rechten Seite die Contexts der verwendeten Controllers zur Verfügung (siehe obige Abbildung). An dieser Stelle können nun Mapping-Beziehungen definiert werden, sofern der abgebildete Knoten nicht mehr Attribute als der Mapping-Ursprung enthält. Dies geschieht durch Drag'n'Drop (Klicken und Ziehen). Es sind zwei Fälle zu unterscheiden:

- Ein unabhängiger Knoten des Quell-Contexts soll abgebildet werden, ist aber im Ziel-Context noch nicht vorhanden. In diesem Fall wird die Quellknotenstruktur auf den Einstiegsknoten des Ziel-Contexts gezogen und so abgebildet.
- Ein Knoten des Quell-Contexts soll abgebildet werden, seine Struktur ist aber im Ziel-Context bereits vorhanden. In diesem Fall ist die Richtung des Klickens und Ziehens umgekehrt: Der Zielknoten wird auf den Mapping-Ursprung im Quellknoten gezogen. Für abhängige Knoten ist dies nur möglich, wenn die zugehörigen unabhängigen Knoten bereits abgebildet sind.

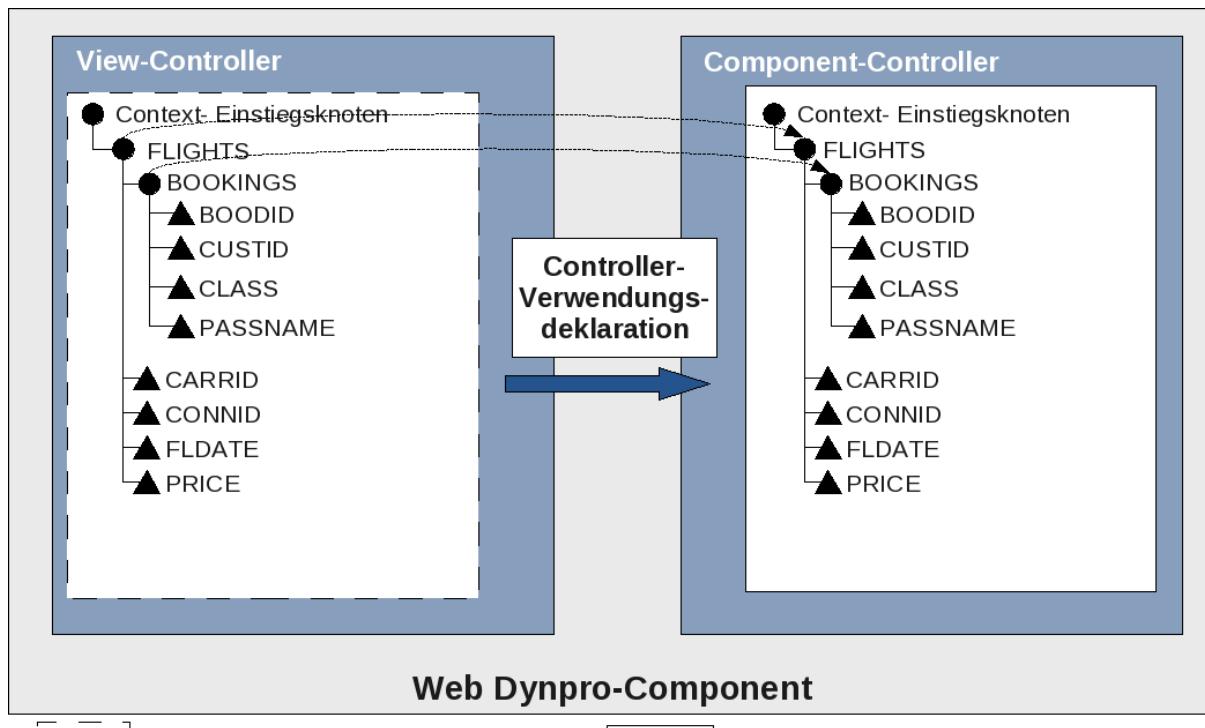


Abbildung 42: Beispiel für Context-Mapping

An dieser Stelle wird davon ausgegangen, dass sich Quell- und Zielcontext in derselben Component befinden (**internes Mapping**). Es ist jedoch auch ein Mapping über die Grenzen einer Component hinweg. Dieses **externe Mapping** wird an dieser Stelle jedoch nicht näher betrachtet.

7.6.5 Praxis: Übung zum Datenaustausch über den Context

In dieser Übung werden Sie nun den Context verwenden, um Daten zwischen Controllern auszutauschen. Legen Sie für diese Übung eine neue Web Dynpro-Component mit dem Namen **ZZ_####_WD_DATA** an. Erstellen Sie wie in der letzten Übung ein Window und zwei Views. Die erste View soll **INPUT_VIEW** heißen, die zweite **OUTPUT_VIEW**. Ziel dieser Übung ist es, Eingaben des Benutzers auf der ersten View (eine Fluggesellschaft und eine Flugnummer) auf der zweiten View auszugeben. Hierzu werden die Eingaben als Context-Daten abgelegt. Dafür muss der Component-Controller-Context verwendet werden, da ein Mapping zwischen den View-Controller-Contexts bekanntlich nicht zulässig ist. Öffnen Sie daher zunächst den Component-Controller und wechseln Sie zu dessen Context, indem Sie die entsprechende Registerkarte öffnen. Legen Sie dort einen Kontextknoten an, indem Sie mit der rechten Maustaste auf den Kontexteinstiegsknoten klicken und **Anlegen -> Knoten** wählen (siehe folgende Abbildung).

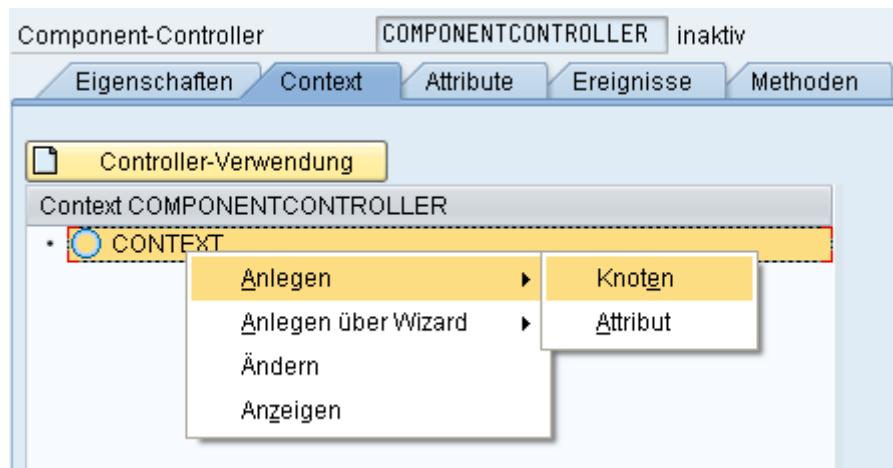


Abbildung 43: Anlegen eines Kontextknotens: SAP-System-Screenshot

Sie werden daraufhin nach den Eigenschaften des anzulegenden Knotens gefragt (siehe folgende Abbildung). Einige der Eingabemöglichkeiten sollten Sie bereits aus diesem Kapitel kennen. Geben Sie als Namen für den Knoten **FLIGHTDATA** an. Flugverbindungen werden in der Datenbank in der Tabelle **SFLIGHT** gespeichert. Auf die hierdurch gegebene Strukturdefinition kann bei der Knotendefinition zurückgegriffen werden. Geben Sie daher den Namen dieser Tabelle als **Dictionary-Struktur** an. Stellen Sie sicher, dass die Kardinalität 1..1 lautet.

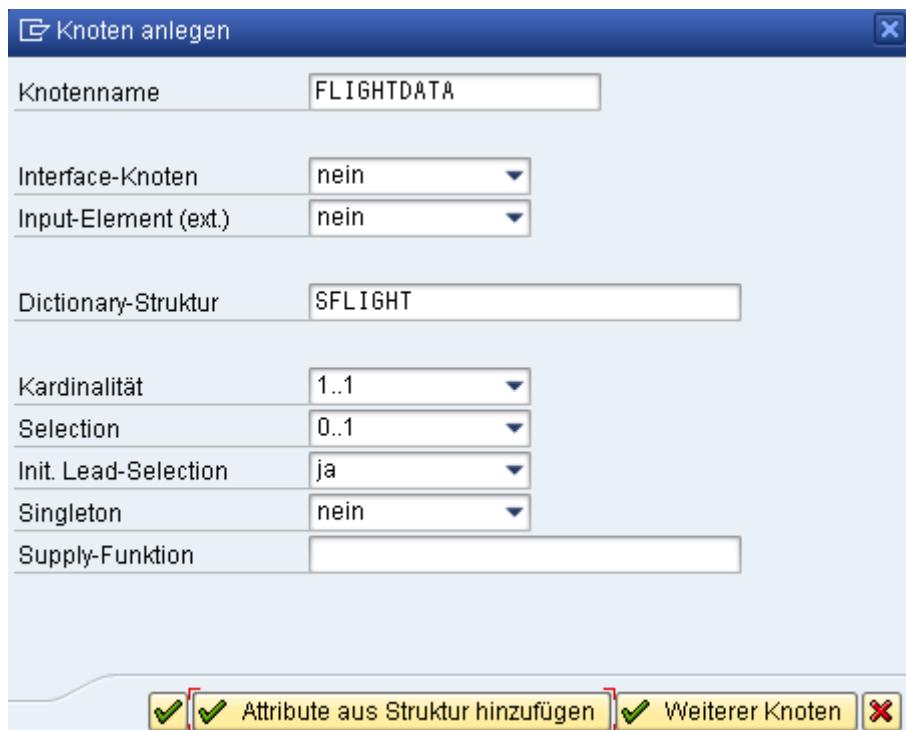


Abbildung 44: Eigenschaften des anzulegenden Knotens: SAP-System-Screenshot

Klicken Sie anschließend auf **Attribute aus Struktur hinzufügen**. Sie werden daraufhin in einem neuen Fenster gefragt, welche Felder der Struktur übernommen werden sollen. In unserem Beispiel genügen die Fluggesellschaft (CARRID) und die Flugverbindung (CONNID). Markieren Sie diese beiden Einträge, indem Sie auf die leere Schaltfläche am Beginn der jeweiligen Zeile klicken, und bestätigen Sie.

Komponenten der Struktur SFLIGHT auswählen							
Komponente	Key	R.typ	Komponententyp	Datentyp	Länge	DezSt...	
<u>MANDT</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<u>S_MANDT</u>	CLNT	3	01	
<u>CARRID</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<u>S_CARR_ID</u>	CHAR	3	01	
<u>CONNID</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<u>S_CONN_ID</u>	NUMC	4	01	
<u>FLDATE</u>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<u>S_DATE</u>	DATS	8	01	
<u>PRICE</u>	<input type="checkbox"/>	<input type="checkbox"/>	<u>S_PRICE</u>	CURR	15	21	

Abbildung 45: Strukturkomponenten übernehmen: SAP-System-Screenshot

Betrachten Sie nun den Context. Dieser besitzt nun wie erwartet eine Baumartige Struktur:

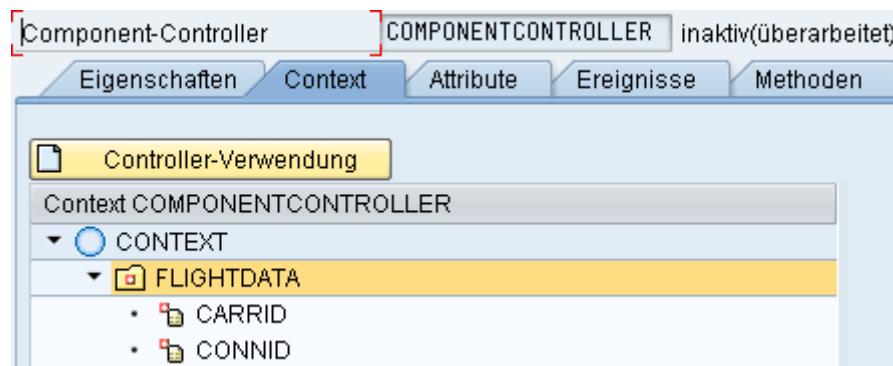


Abbildung 46: Context des Component-Controllers: SAP-System-Screenshot

Klicken Sie die Elemente dieser Baumstruktur an, um deren Eigenschaften im unteren Bereich zu sehen. Während Context (Als Kontexteingangsknoten) und FLIGHTDATA Knoten sind, handelt es sich bei CARRID und CONNID um Attribute.

Sichern Sie den Component-Controller. Legen Sie den Kontextknoten mit beiden Attributnamen anschließend auch im Contexts der View **INPUT_VIEW** an. Gehen Sie dabei analog zu den soeben beim Component-Controller gezeigten Schritten vor.

Als nächstes soll das Mapping zwischen den Contexts definiert werden. Sie finden auf der rechten Seite der Registerkarte **Context** des View-Controllers bereits den Context des Component-Controllers. Ziehen Sie mit der Maus dessen **FLIGHTDATA**-Knoten auf den des View-Controller-Contexts. Das System bestätigt Ihnen, dass die Mapping-Beziehung angelegt wurde. Sie können dies jederzeit überprüfen: In den Eigenschaften des Knotens im View-Controller-Context, die nach Anklicken im unteren Bereich des Fensters sichtbar sind, finden Sie die Eigenschaft **Mapping-Pfad**:

Eigenschaft	Wert	Att
<u>Knoten</u>		
Knotenname	FLIGHTDATA	
Dictionary-Struktur	SFLIGHT	
Kardinalität	1..1	
Selection	0..1	
Initialisierung Lead-Selection	<input checked="" type="checkbox"/>	
Singleton	<input type="checkbox"/>	
Supply-Funktion		
Mapping-Pfad	ZZ_9998_WD_DATA.COMPONENTCONTROLLER.FLIGHTDATA	

Abbildung 47: Mapping-Pfad in den Knoteneigenschaften: SAP-System-Screenshot

Für die zweite View muss nun auch ein Knoten angelegt und gemappt werden. Sichern Sie die erste View und öffnen Sie die View **OUTPUT_VIEW**. Wechseln Sie dort zur Registerkarte **Context**. Um den Schritt des Anlegens des Knotens im View-Controller-Context zu ersparen, gibt es die Möglichkeit Anlegen und Mapping-Definition in einem Schritt durchzuführen. Ziehen Sie dazu aus dem Component-Controller-Context auf der rechten Seite den Knoten **FLIGHTDATA** auf den Kontexteinstiegsknoten des View-Controller-Contexts. Vergewissern Sie sich, dass der Knoten mit beiden Attributnamen übernommen und das Mapping angelegt wurde. Sichern Sie anschließend die View.

Als nächstes werden die Eingabefelder benötigt. Öffnen Sie zunächst die View **INPUT_VIEW** und wählen Sie dort die Registerkarte Layout aus. Sie könnten hier nun die Eingabefelder wie in der letzten Übung anlegen. Es fehlen Ihnen bislang aber Kenntnisse darüber, wie die Anordnung von UI-Elementen in Web Dynpro funktioniert, und das Anlegen wäre relativ mühsam. Stattdessen steht Ihnen hier jedoch ein Wizard zur Verfügung, der Ihnen die Arbeit abnimmt, indem er die UI-Elemente erzeugt und die Datenbindung festlegt.

Diesen **Web Dynpro Code Wizard** erreichen Sie über die Schaltfläche .



Abbildung 48: Template Gallerie [sic!]: SAP-System-Screenshot

Wählen Sie im erscheinenden Fenster (siehe Abbildung oben) das Template **Form** durch Doppelklick aus. Klicken Sie im nächsten Fenster auf  **Context**, um dort den Kontextknoten **FLIGHTDATA** auszuwählen.

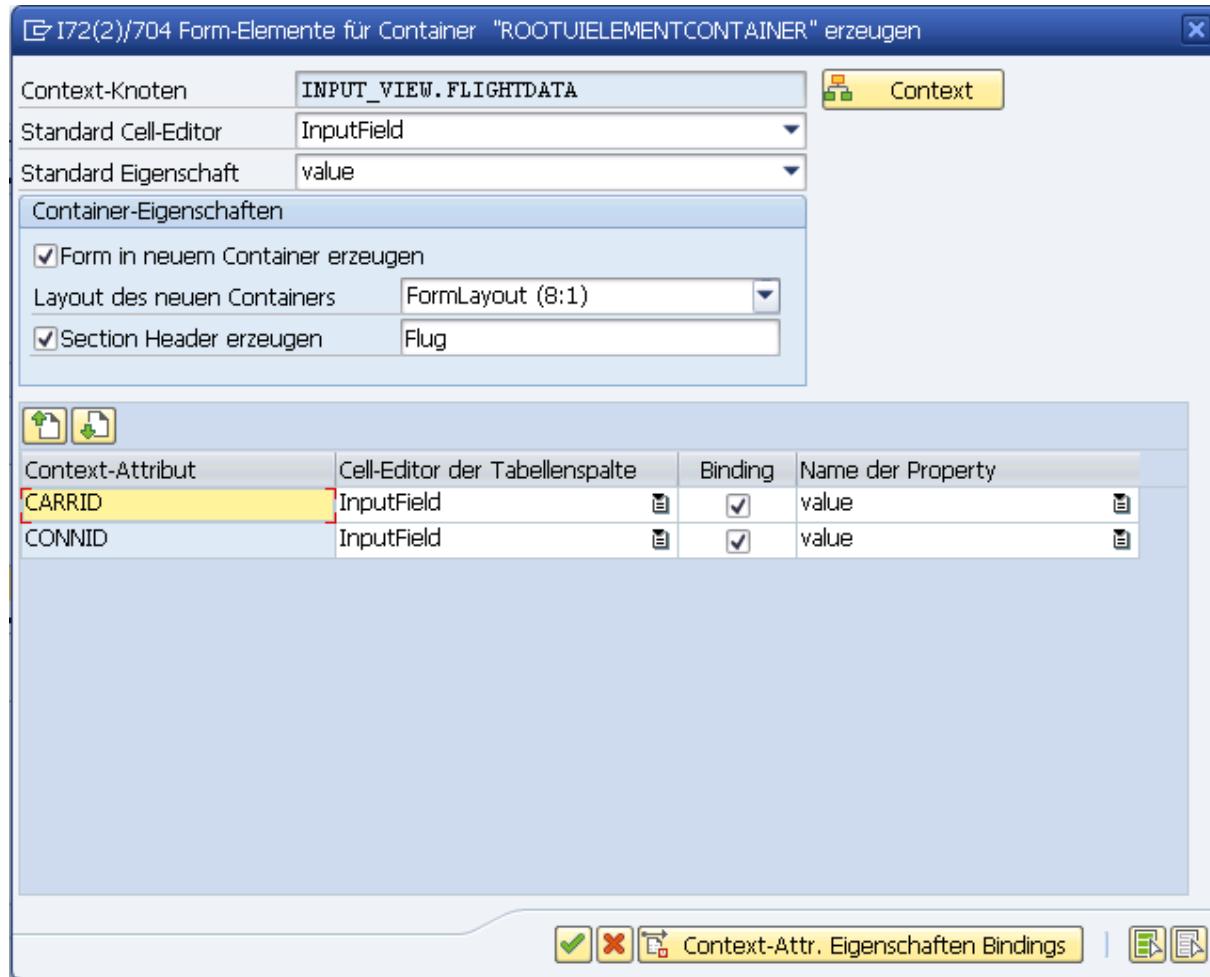


Abbildung 49: Context-Binding erzeugen: SAP-System-Screenshot

Stellen Sie zudem sicher, dass **Form in neuem Container erzeugen** und **Section Header einfügen** angehakt sind, und geben Sie eine passende Überschrift im Feld neben **Section Header einfügen** ein. Bestätigen Sie die Eingaben. Sichern und Aktivieren Sie. Ihre View enthält nun wie erwartet das gewünschte Formular:

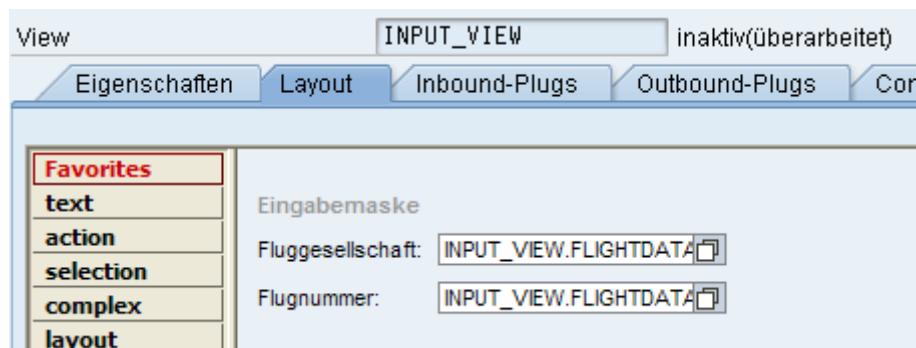


Abbildung 50: Eingabeformular der INPUT_VIEW: SAP-System-Screenshot

Sollte das Formular nicht sichtbar werden, öffnen Sie die andere View und kehren Sie dann wieder zur ersten View zurück. Verfahren Sie analog für die View **OUTPUT_VIEW**. Setzen Sie dort jedoch für die Textfelder die Eigenschaft **readOnly**.

Beim Prüfen der Views erhalten sie möglicherweise einen Hinweis zur AccessibilityDescription. Dieser bezieht sich auf die Barrierefreiheit, die durch die genannte Eigenschaft verbessert werden soll. Es handelt sich nur um eine Information, so dass eine Korrektur nicht zwingend nötig ist, sie können aber die Eigenschaft mit einer kurzen

Beschreibung des UI-Elements füllen. In den weiteren Übungen wird dies nicht mehr explizit erwähnt.

Verbinden Sie nun die beiden Views durch eine entsprechende Navigation. Legen Sie hierzu die benötigten Inbound- und Outbound-Plugs sowie Buttons auf den Views an. Vergessen Sie nicht sicherzustellen, dass beide Views Teil des Windows sind. Versuchen Sie, diese Schritte selbstständig durchzuführen, um sicherer im Umgang mit diesen Konzepten zu werden. Wenn Sie bei diesem Vorgehen nicht weiter kommen, blättern Sie zurück zur letzten Übung. Legen Sie schließlich auch hier eine Web Dynpro-Anwendung an, und testen Sie diese nachdem Sie alle anderen Objekte gesichert und aktiviert haben. Testen Sie die Anwendung anschließend. Beachten Sie, dass auch eine Werthilfe angeboten wird, deren Aufbau den gewohnten Komfort von herkömmlichen SAP-Oberflächen bietet.

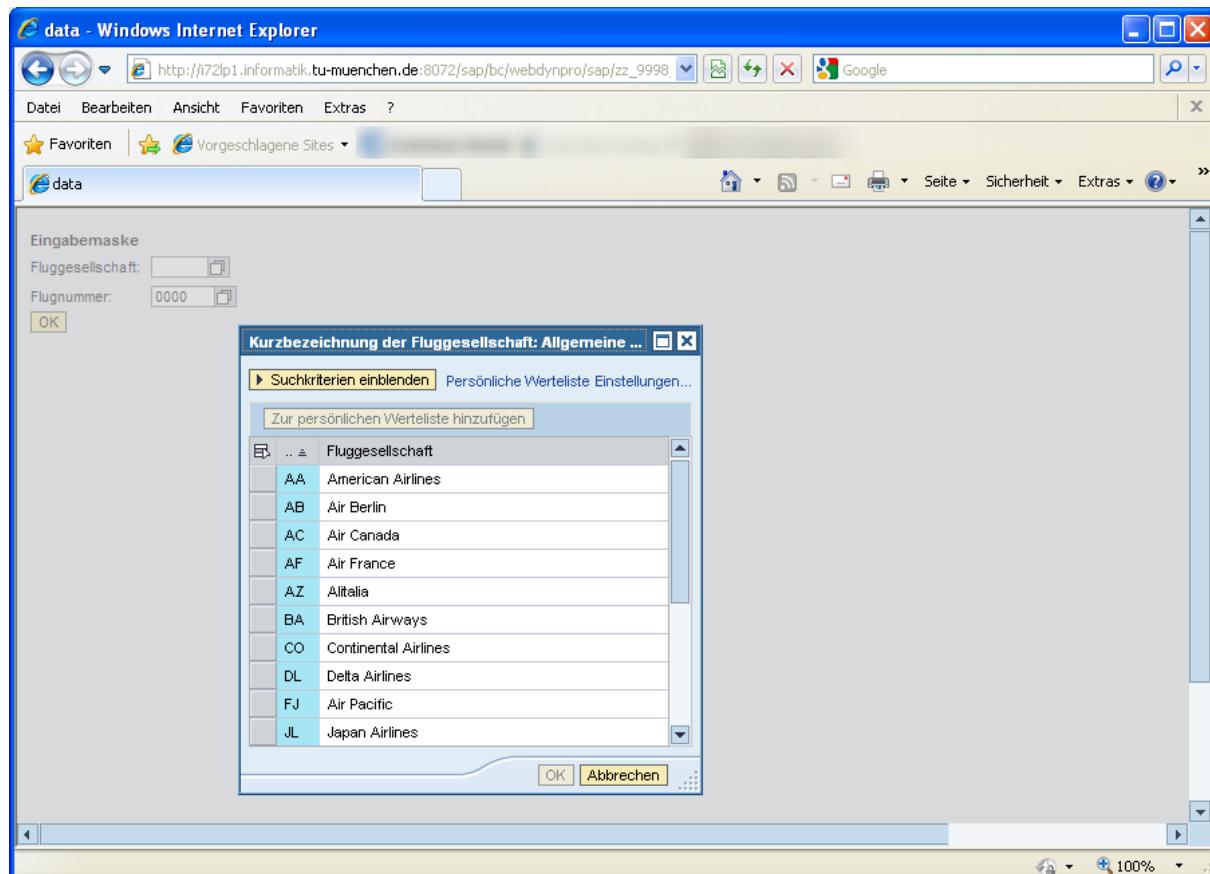


Abbildung 51: Eingbehife in Web Dynpro-Oberflächen: SAP-System-Screenshot

Auf der zweiten Maske sind die eingegebenen Daten der ersten Maske sichtbar. Sie haben nun also den Context benutzt, um Daten in mehreren View-Controllern verwenden zu können. Zur Definition des Layouts und die Datenbindung haben Sie sich auf den Wizard verlassen. Um die Möglichkeiten der Oberflächengestaltung genauer zu verstehen, werden diese im folgenden Unterkapitel näher erläutert.

7.7 Web-Dynpro-Oberflächengestaltung

Die Oberfläche wird im Web Dynpro-Konzept in Form von Metadaten abgelegt. Aus diesen Metadaten (insb. UI-Elemente und deren Eigenschaften) wird dann zur Laufzeit das tatsächliche Layout als HTML-Code generiert. Hierbei kann die Umsetzung unterschiedlich sein, je nachdem mit welchem Client die Seite aufgerufen wird. Derzeit wird als Client in Form eines Webbrowsers verwendet, eine andere Abbildung der Metadaten für weitere Clients (z. B. mobile Endgeräte) wäre aber für die Zukunft auch umsetzbar. Die Eigenschaften der UI-Elemente können im Code des Controllers manipuliert werden. So lassen sich dynamische Oberflächen realisieren.

Sie haben den Begriff des **UI-Elements** bereits kennengelernt. Formal handelt es sich dabei um jede grafische Entität, die einen Platz in einem View-Layout einnimmt. Aus einfachen UI-Elementen wie Textelementen werden auch komplexere Elemente wie **TABLE** oder **TREE** zusammengesetzt. Nicht alle Elemente, die auf einer View definiert sind, erscheinen auch für den Anwender sichtbar. Einige Elemente werden nur zur Strukturierung der Oberfläche verwendet, etwa der **TransparentContainer**, von dem in der letzten Übung vom Wizard eine Instanz angelegt wurde. Weiterhin können Elemente als unsichtbar markiert werden. Hierfür findet sich in deren Eigenschaften ein Eintrag **visible**. Ist ein UI-Element so als nicht sichtbar markiert, belegt es dennoch den theoretisch nötigen Platz. Auch zur Laufzeit kann die Sichtbarkeitseigenschaft verändert werden. Durch die beschriebene Platzreservierung führt ein Ausblenden nicht dazu, dass daneben befindliche Elemente plötzlich aufrücken.

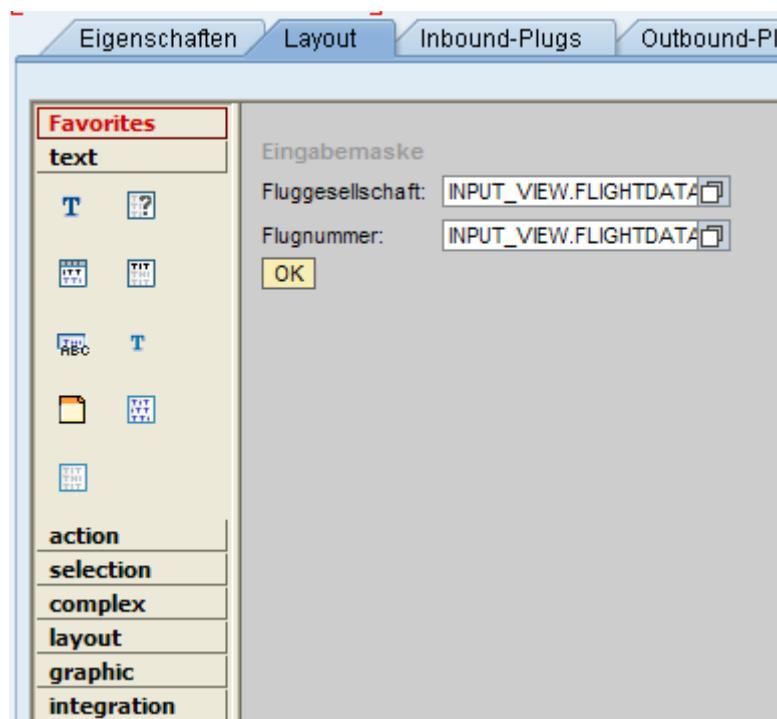


Abbildung 52: Kategorien von UI-Elementen: SAP-System-Screenshot

Beim Bearbeiten des Layouts stehen Ihnen auf der linken Seite die verschiedenen UI-Elemente zur Verfügung. Diese können auch durch Klicken und Ziehen verwendet werden. Sie sind in verschiedene Kategorien gegliedert. So enthält die Kategorie **text** etwa Elemente, die Text auf der Oberfläche anzeigen oder eingeben lassen (z. B. TextView). Unter **action** befinden sich Elementen mit denen Aktionen ausgelöst werden können (z. B. Button). **selection** enthält Elemente die Auswahlmöglichkeiten ermöglichen (z. B. CheckBox oder RadioButton). Unter **complex** befinden sich komplexe Elemente, die sich aus anderen Elementen

zusammensetzen (z. B. Table). Unter **layout** finden sich allgemeine Elemente zur Gestaltung des Layouts der Oberfläche (z. B. Group). Grafische Elemente befinden sich unter **graphic**, und Elemente des Kategorie **integration** umfassen bspw. Dateiup- und Downloads.

Beachten Sie, dass die Kategorieeinteilung Releaseabhängig ist. Je nach System können Sie auch eine Einteilung in die Kategorien Standard Simple, Standard Complex, Standard Container, Active Component, Adobe, BusinessGraphics, BusinessIntelligence, OfficeIntegration und Pattern vorfinden.

7.7.1 Praxis: Testen von UI-Elementen

Um UI-Elemente unabhängig von einer eigenen Anwendung testen zu können, stellt SAP eine Testanwendung **WDR_TEST_UI_ELEMENTS** bereit. Diese erreichen Sie, indem Sie die Eingabefelder oberhalb des Navigationsbaums im Object Navigator verwenden, um die gleichnamige Component zu öffnen und dort die Anwendung auswählen und testen.

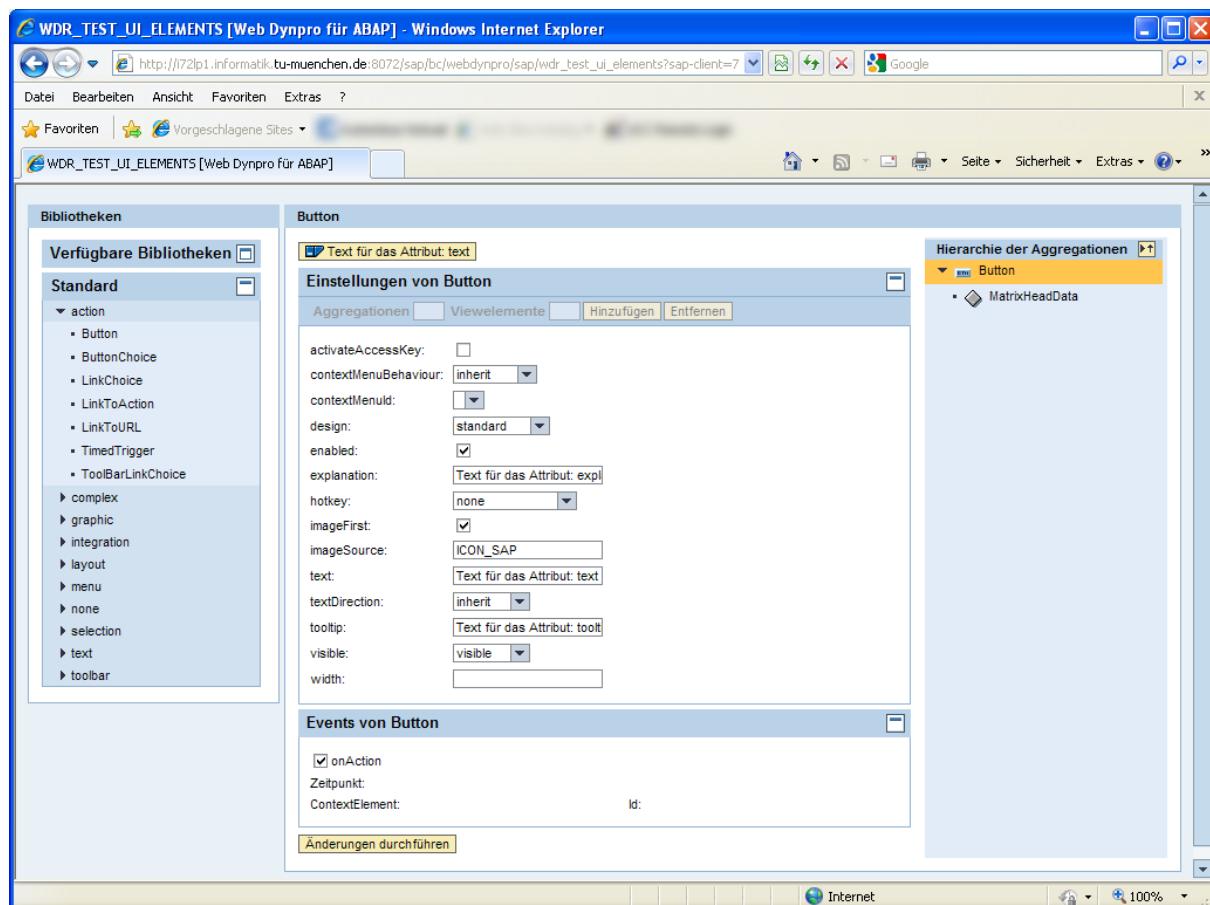


Abbildung 53: Testanwendung für UI-Elemente: SAP-System-Screenshot

Nutzen Sie diese Anwendung, um sich mit den Elementen vertraut zu machen. In der Abbildung sehen Sie das Beispiel eines Buttons. Der gerenderte Button ist oben mittig sichtbar. Darunter können die Eigenschaften des Buttons verändert und mit der Schaltfläche **Änderungen durchführen** übernommen werden.

7.7.2 Hierarchien von UI-Elementen

Elemente der Web Dynpro-Oberfläche sind hierarchisch angeordnet. Dies ist bereits bei Ihren zuletzt entwickelten Views erkennbar:

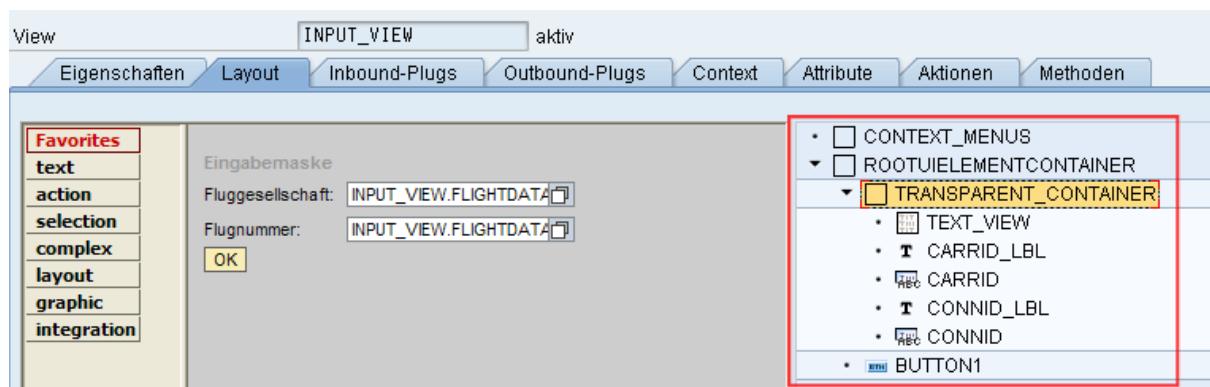


Abbildung 54: Darstellung der Elementhierarchie: SAP-System-Screenshot

Auf der rechten Seite der Layoutdefinition erkennen Sie eine entsprechende Baumdarstellung. Die von Ihnen angelegten UI-Elemente sind hierbei alle einem Element

ROOTUIELEMENTCONTAINER untergeordnet. Dieses Element ist stets vom Typ **TransparentContainer**.

Containerelemente (wie hier der TransparentContainer) sind UI-Elemente, die Unterelemente besitzen können. Sie nehmen einen rechteckigen Platz auf der View ein. Die einzelnen Unterelemente sind Teil dieses rechteckigen Platzes. Das Containerelement besitzt eine spezielle Eigenschaft **Layout**, mit der festgelegt wird, wie die Unterelemente angeordnet werden sollen. Dafür wird das UI-Element einem **Layout-Manager** zugeordnet. Die Unterelemente erben Layouteigenschaften des Container-Elements, die dann im Eigenschaftsbereich **Layoutdaten** zur Verfügung stehen und die Positionierung der Elemente steuern.

Als Layout stehen die folgenden sieben Möglichkeiten zur Auswahl:

- FlowLayout
- RowLayout
- MatrixLayout
- GridLayout
- FormLayout
- FormLayoutAdvanced
- TileLayout

Das FlowLayout ist der Standard-Layout-Manager. Hierbei werden alle UI-Elemente des Containers in einer Zeile angeordnet. Reicht der Platz in der Zeile nicht aus, wird ein Zeilenumbruch durchgeführt. Dies geschieht zum Beispiel dann, wenn das Browserfenster entsprechend verkleinert wird. Reichen auch zwei Zeilen nicht aus, wird analog weiter umgebrochen.

Container-Eigenschaften

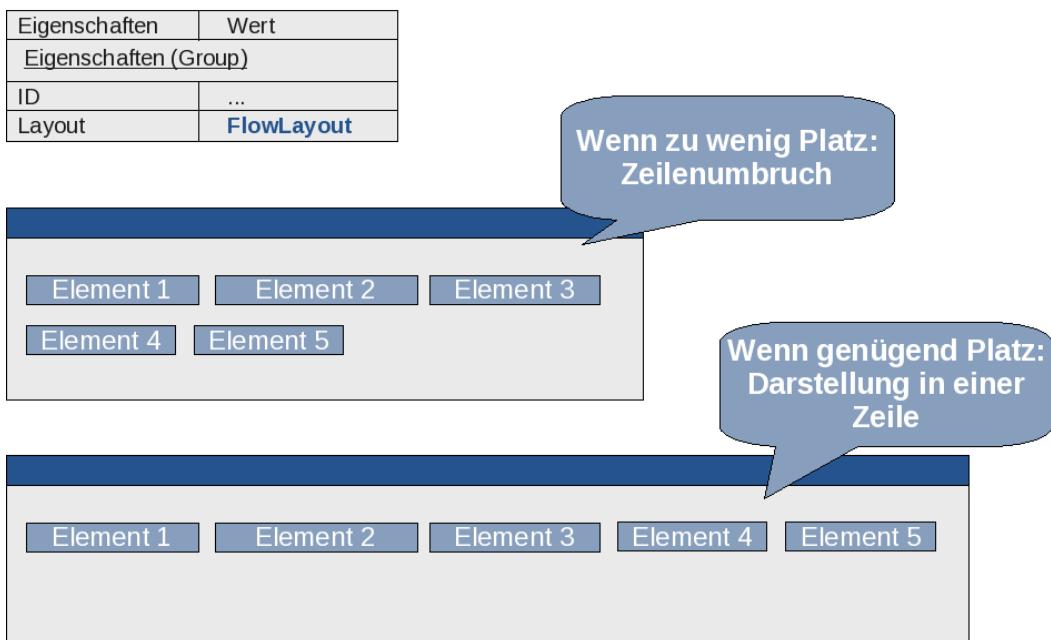


Abbildung 55: FlowLayout

Das FlowLayout ist gut geeignet, um Containerelemente anzurichten und so die Darstellung an die Fenstergröße anzupassen. Die Positionierung einzelner UI-Elemente kann aber leicht unordentlich wirken. Die Umbrüche können in den Eigenschaften unterbunden werden.

Das **RowLayout** ist dem FlowLayout recht ähnlich. Auch hier werden die Elemente in Zeilen angeordnet, ohne dass untereinander stehende Elemente als Spalten ausgerichtet werden:

Container-Eigenschaften

Eigenschaften	Wert
Eigenschaften (Group)	
ID	...
Layout	RowLayout

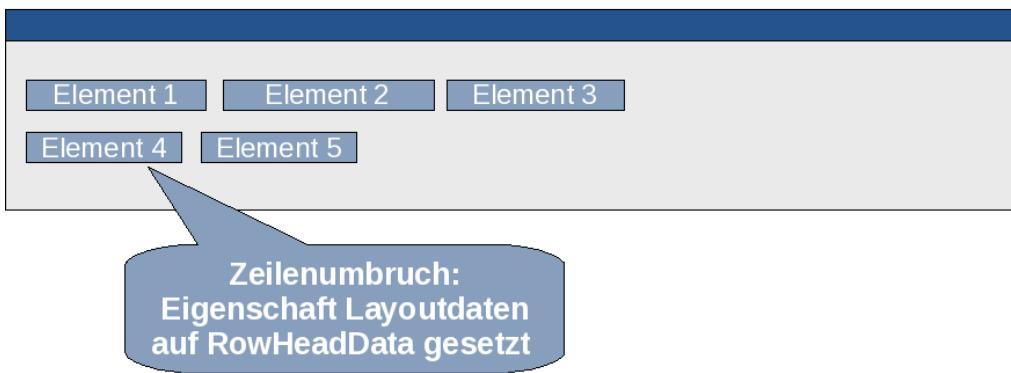


Abbildung 56: RowLayout

Im Unterschied zum FlowLayout geschieht der Zeilenumbruch hier aber nicht automatisch. Stattdessen erben die untergeordneten Elemente die Eigenschaft **Layoutdaten**. Diese kann die Werte **RowData** oder **RowHeadData** annehmen. Ist bei einem Element der Wert von Layoutdaten als RowHeadData gesetzt, findet ein Zeilenumbruch statt, d. h. das Element steht

als erstes Element in einer neuen Zeile. Andernfalls findet kein Zeilenumbruch statt – auch nicht, wenn der zur Verfügung stehende Platz nicht ausreicht. Die Breite der Elemente ist jeweils durch die Eigenschaft **width** einstellbar.

Sollen Elemente der verschiedenen Zeilen auch in Spalten untereinander angeordnet werden, kommen MatrixLayout und GridLayout infrage.

Das MatrixLayout erzeugt den Zeilenumbruch ähnlich wie das RowLayout. Die untergeordneten Elemente des Layouts erben ein Attribut Layoutdaten, welches hier die Werte MatrixData oder MatrixHeadData annehmen kann. Das setzen des Wertes MatrixHeadData bewirkt, dass das Element als erstes Element in einer neuen Zeile positioniert wird, während der Wert MatrixData ein Einfügen in derselben Zeile bewirkt. Durch dieses explizite Erzeugen des Umbruchs müssen die Zeilen nicht dieselbe Anzahl von Elementen enthalten. Die vorhandenen Elemente sind in Spalten angeordnet, wie die folgende Abbildung veranschaulicht:

Container-Eigenschaften

Eigenschaften	Wert
Eigenschaften (Group)	
ID	...
Layout	MatrixLayout

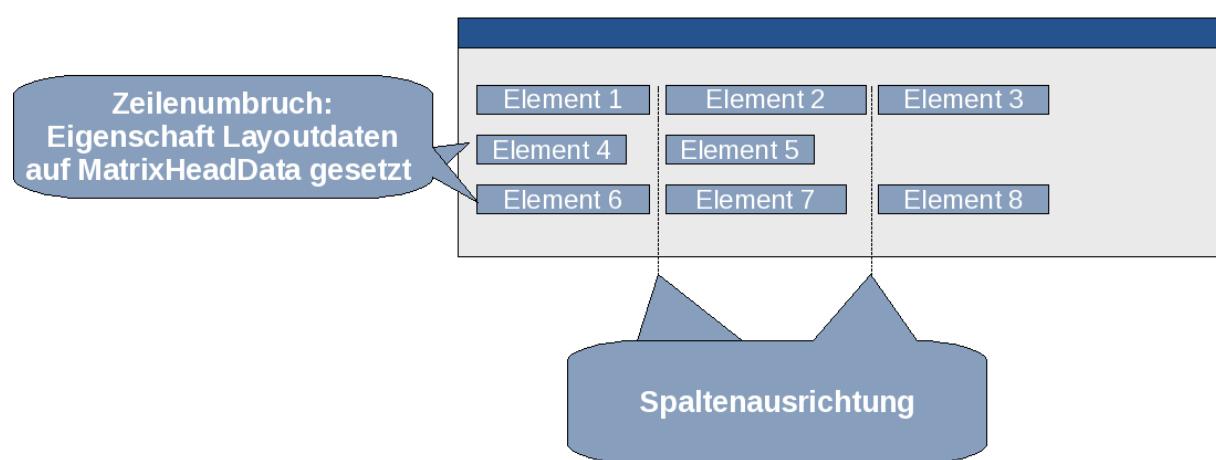


Abbildung 57: MatrixLayout

Durch die Eigenschaft **colSpan** kann festgelegt werden, dass ein Element mehr als eine Spalte einnimmt:

Verwendung von 2 Spalten durch setzen der colSpan-Eigenschaft

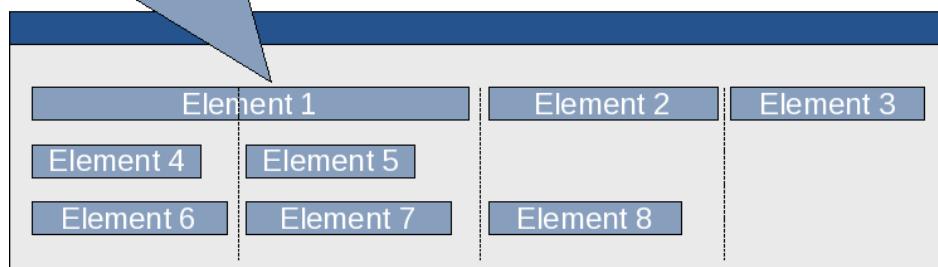
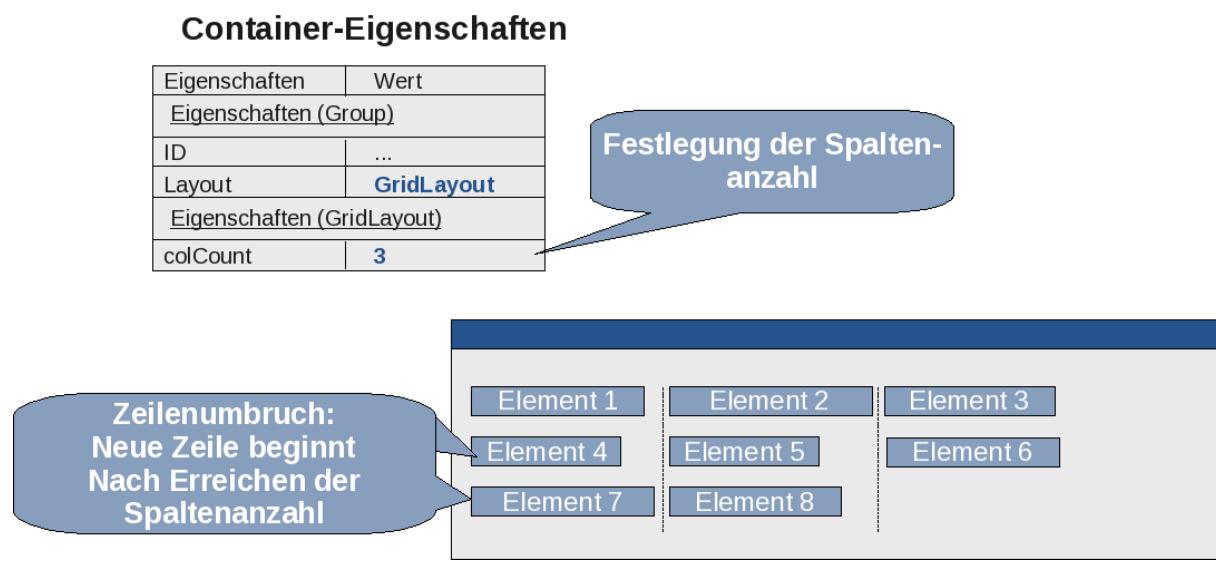


Abbildung 58: Die colSpan-Eigenschaft

Die Anzahl der Spalten hängt hier von den Elementen ab und wird nicht durch den Container definiert. Die Zeilen können unterschiedlich viele Elemente enthalten.

Beim **GridLayout** wird die Spaltenanzahl hingegen im Containerelement festgelegt. Es ist nicht möglich, durch setzen der Eigenschaft **colCount** im untergeordneten Element einen Zeilenumbruch zu erzwingen. Stattdessen erfolgt der Umbruch automatisch anhand der festgelegten Spaltenanzahl.



Auch beim GridLayout kann durch die Eigenschaft **colSpan** in einem Element festgelegt werden, dass dieses sich über mehr als eine Spalte erstreckt.

Mit dem **FormLayout** können Elemente in Spalten wie in einer Zeitung angeordnet werden. Um eine neue Spalte zu erzeugen wird ein Element als FormTopData konfiguriert. Im **FormLayoutAdvanced** bestehen zusätzliche Layoutmöglichkeiten über Containergrenzen hinweg.

Das **TileLayout** ähnelt dem FlowLayout, wobei hier die Spalten die gleiche Breite aufweisen. So kann ein Kachelartiges Layout generiert werden.

7.7.3 Einfügen von UI-Elementen

Bislang haben Sie zum Einfügen von Elementen das Kontextmenü der Elementhierarchie benutzt. Dies geschah durch Rechtsklick auf das übergeordnete Element in der Hierarchie (Voraussetzung hierbei: Es muss sich um ein Element handeln, das Unterelemente besitzen kann). Sie können allerdings auch die kategorisierte Elementliste des View-Editors benutzen. Voraussetzung dafür ist, dass die Layout-Vorschau eingeblendet ist. Diese kann durch ein- bzw. ausgeblendet werden.

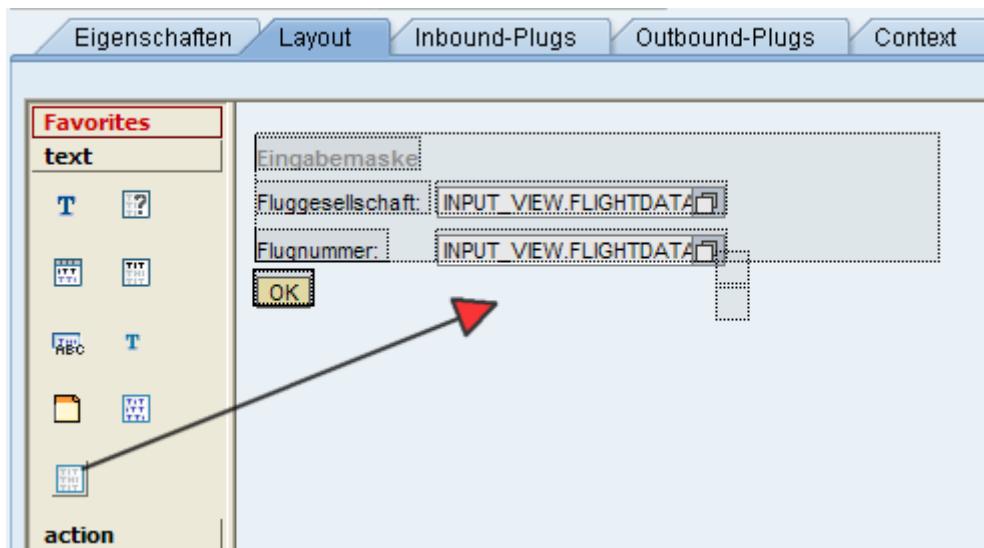


Abbildung 60: Elemente einfügen: SAP-System-Screenshot

Aus der Elementliste lassen sich die Elemente mit der Maus nach rechts in die Layout-Vorschau ziehen (siehe obige Abbildung).

Um die Position eines Elementes innerhalb der View zu verändern, kann ebenfalls „Drag’n’Drop“ verwendet werden. Entweder das Element wird in der View-Vorschau bewegt, oder in der Elementhierarchie. Dort steht Ihnen zusätzlich die Möglichkeit zur Verfügung, aus dem Kontextmenü die Einträge **nach oben** bzw. **nach unten** aufzurufen und das Element dadurch um jeweils einen Schritt zu verschieben.

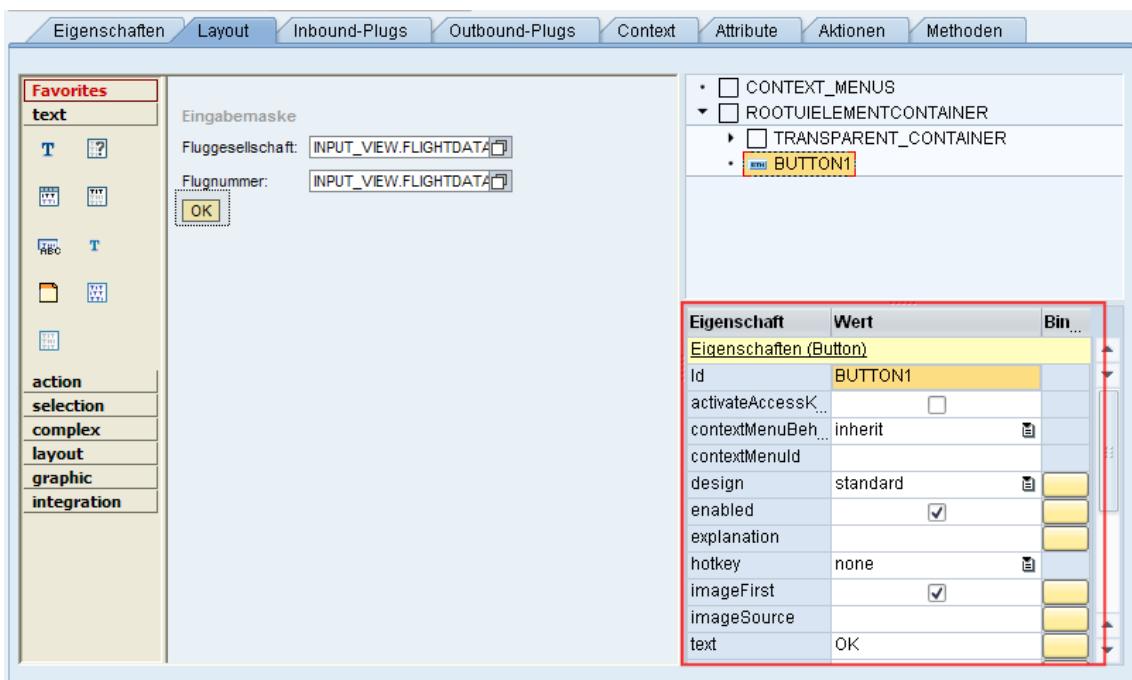


Abbildung 61: Elementeigenschaften: SAP-System-Screenshot

Unterhalb der Elementhierarchie finden Sie die bereits erwähnten Eigenschaften. Um die Eigenschaften eines bestimmten Elements aufzurufen, muss dieses ausgewählt werden. Dies kann entweder in der Layout-Vorschau oder in der Elementhierarchie geschehen. Zu den Eigenschaften zählen auch die Client-seitigen Ereignisse, wie bei einem Button das Ereignis `onClick`. Die Namen der Ereignisse beginnen mit **on**. Diesen werden hier Aktionen zugeordnet, wie Sie es bereits im praktischen Teil getan haben. Es ist nicht möglich, weitere

Client-seitige Ereignisse zu definieren, etwa durch implementieren von JavaScript-Code. Es kann ausschließlich Gebrauch von den vordefinierten Ereignissen gemacht werden.

7.7.4 Datenbindung

Die Datenbindung haben Sie in der letzten Übung dazu verwendet, den Inhalt der Textfelder an entsprechende Kontextattribute zu binden. Die Datenbindung erlaubt aber nicht nur ein Belegen von Texten. Es können auch andere Eigenschaften des UI-Elements gebunden werden. Hierfür muss im Kontext (desselben View-Controllers) ein Knoten oder Attribut mit passendem Typ vorhanden sein. Die Kontextdaten werden dann zur Versorgung der Eigenschaft des UI-Elements verwendet. Umgekehrt wird auch der Context aktualisiert, wenn es sich um eine Eigenschaft handelt, die der Benutzer verändern kann. Durch diese Möglichkeiten ist ein direkter Zugriff aus dem Programmcode auf UI-Elemente in der Regel nicht erforderlich. Stattdessen werden alle Steuerungsaufgaben durch Bearbeiten des Contexts erfüllt. Hierfür ist es erforderlich, dass alle Eigenschaften der UI-Elemente, die zur Laufzeit angepasst werden sollen, an Kontextknoten bzw. -Attribute gebunden werden.

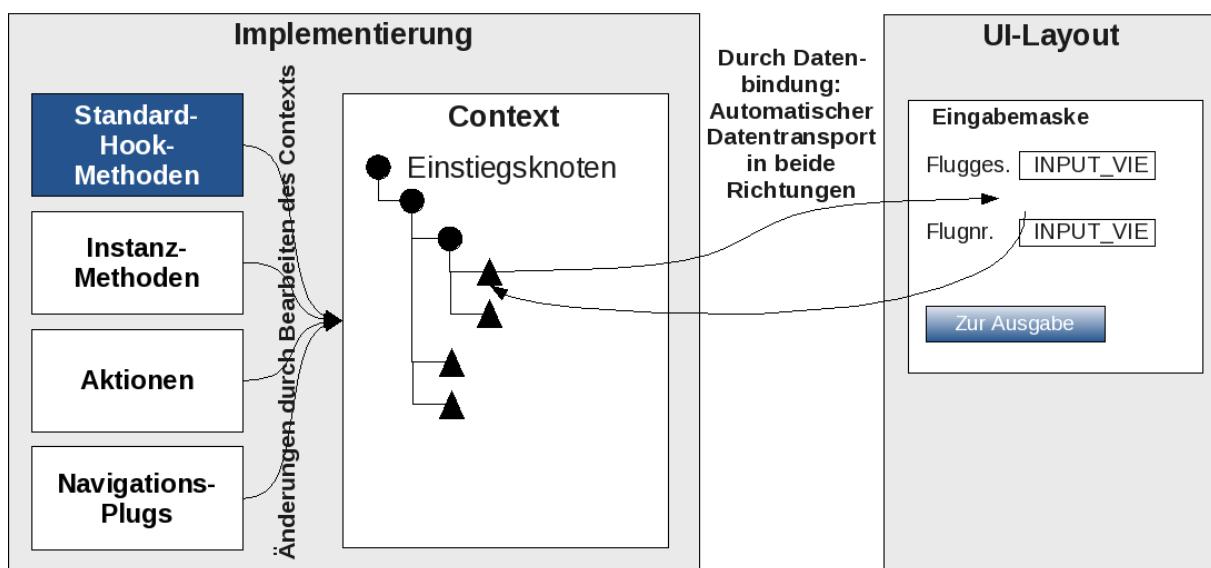


Abbildung 62: UI-Elemente per Context bearbeiten

Die wichtigste Datenbindung ist die von Ihnen in der Übung verwendete Bindung der **value**-Eigenschaft. Nur UI-Elemente, bei denen diese Eigenschaft gebunden ist, können überhaupt Daten anzeigen. Die Eigenschaft ist bei fast allen Elementen vorhanden. Ausnahmen sind z. B. das UI-Element **HorizontalGutter**.

Um also Daten auf einer View sichtbar zu machen, werden ein Kontextknoten oder -Attribut, ein UI-Element und eine Bindung benötigt. Der Kontextknoten bzw. das Kontextattribut werden auf der Registerkarte **Context** der View angelegt (es kann sich um einen abgebildeten Knoten handeln), das UI-Element auf der Registerkarte **Layout**. Die Datenbindung wird ebenfalls auf dieser Registerkarte hergestellt, indem im Bereich der Eigenschaften des UI-Elements auf die Schaltfläche in der Spalte **Binding erzeugen...** geklickt wird. Diese führt zu einem Fenster, in dem die Knoten bzw. Attribute des Contexts angezeigt werden, die für eine Bindung geeignet sind (d. h. einen passenden Typ haben). Diese können dort markiert werden. Anschließend wird der Kontextpfad als Wert der Eigenschaft sowie in der Layout-Vorschau im entsprechenden Element angezeigt und die Schaltfläche in den Eigenschaften ändert ihr Aussehen von zu .

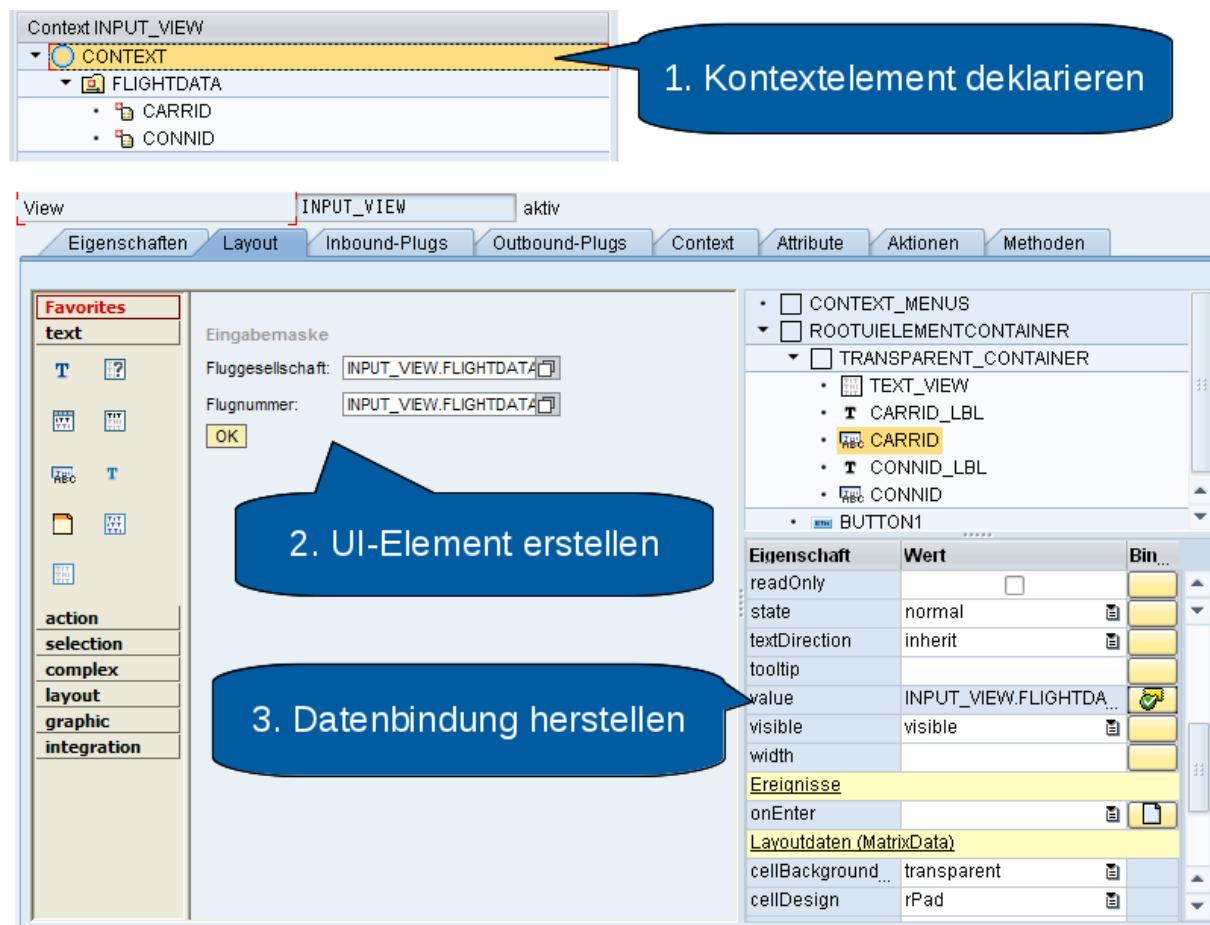


Abbildung 63: Schritte zur Anzeige von Daten: SAP-System-Screenshot (Montage)

Die Datenbindung ist insgesamt ein mächtiges Werkzeug, das weit mehr leistet als nur Daten in UI-Elemente zu schreiben. Eine Vielzahl weiterer Eigenschaften der UI-Elemente ist durch Datenbindung bequem steuerbar. Außerdem funktioniert die Datenbindung nicht nur in eine Richtung. Sie haben bereits in der Übung gesehen, dass die Eingaben in das UI-Element übernommen wurden, d. h. die Daten werden sowohl vom Context zum UI als auch vom UI zum Context transportiert.

Der Rücktransport vom UI zum Context geschieht durch Interaktion des Benutzers und Initiierung eines HTTP-Roundtrips, etwa indem der Benutzer etwas in ein Feld eingibt und eine Schaltfläche anklickt, woraufhin ein Request zum Server geschickt und eine neue HTML-Seite an den Browser zurückgeliefert wird. Bei der Behandlung von Aktionen stehen bereits die aktualisierten Daten im Context und können verarbeitet werden. Hierfür ist kein Programmieraufwand erforderlich, es muss lediglich deklarativ die Beziehung durch das

definieren einer Datenbindung hergestellt werden. Der eigentliche Transport der Daten wird dann vom System automatisch durchgeführt.

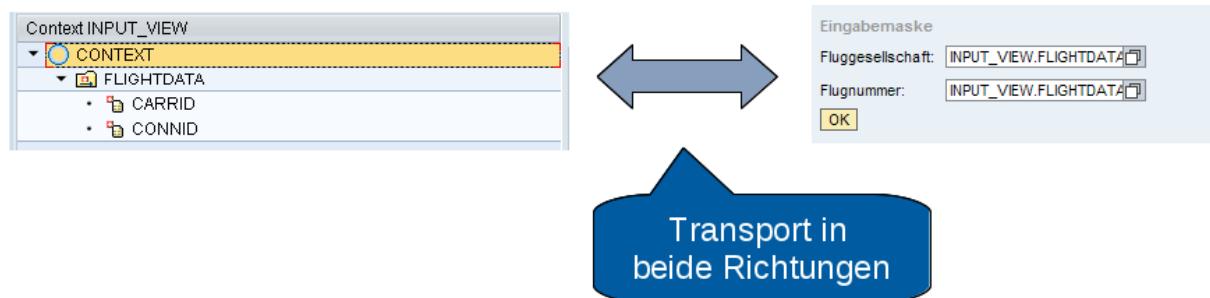


Abbildung 64: Datentransport - eine wechselseitige Beziehung: SAP-System-Screenshot (Montage)

Zum Designzeitpunkt können den Eigenschaften der UI-Elemente Werte zugewiesen werden. Wird auf die Datenbindung verzichtet, ist es nicht mehr möglich, diese zur Laufzeit über den Context anzupassen. Die Möglichkeit der Anpassung ist dadurch aber nicht völlig ausgeschlossen. Aus dem Programmcode des View-Controllers kann direkt, also ohne eine dazwischen geschaltete Datenbindung, auf die UI-Elemente zugegriffen werden. Dieser Zugriff ist nur über die Hook-Methode **wddomodifyview** möglich, die eine Referenz auf die Hierarchie der UI-Elemente bietet. Es gilt allerdings als schlechter Programmierstil, diesen direkten Weg des Zugriffs zu wählen. Der Grund liegt in der gewünschten Trennung zwischen Ablauflogik und Oberfläche. Durch den direkten Zugriff ist diese Trennung nicht mehr gewährleistet. Daher sollten Sie derartige Zugriffe in ihren Programmen nicht verwenden.

Zusammenfassend gibt es drei Möglichkeiten die UI-Element-Eigenschaften zu belegen:

- Statisch zur Design-Zeit
- Dynamisch über den Context durch Datenbindung
- Dynamisch durch direkten Zugriff auf die Elemente

Damit Sie Eigenschaften der UI-Elemente über den Context durch Datenbindung bearbeiten können, müssen Sie wie oben skizziert ein Kontextattribut im Context des View-Controllers anlegen. Dieses Attribut muss passend typisiert sein, damit eine Datenbindung definiert werden kann. Die entsprechenden Typen finden Sie wie folgt:

Erstellen Sie ein Kontextattribut, und wechseln Sie im dafür erscheinenden Fenster in das Feld **Typ**. Rufen Sie für dieses Feld die Eingabehilfe auf (F4). In der Eingabehilfe befindet sich eine Registerkarte **Typen der Web Dynpro Runtime**. Wenn Sie einen Teil des Namens kennen, können Sie hier mit Wildcards arbeiten, ansonsten führt Sie ein Bestätigen ohne Eingabe zur Liste aller Typen, die in diesem Zusammenhang infrage kommen. Zu jedem Typ ist angegeben, zu welcher UI-Element-Eigenschaft er gehört:

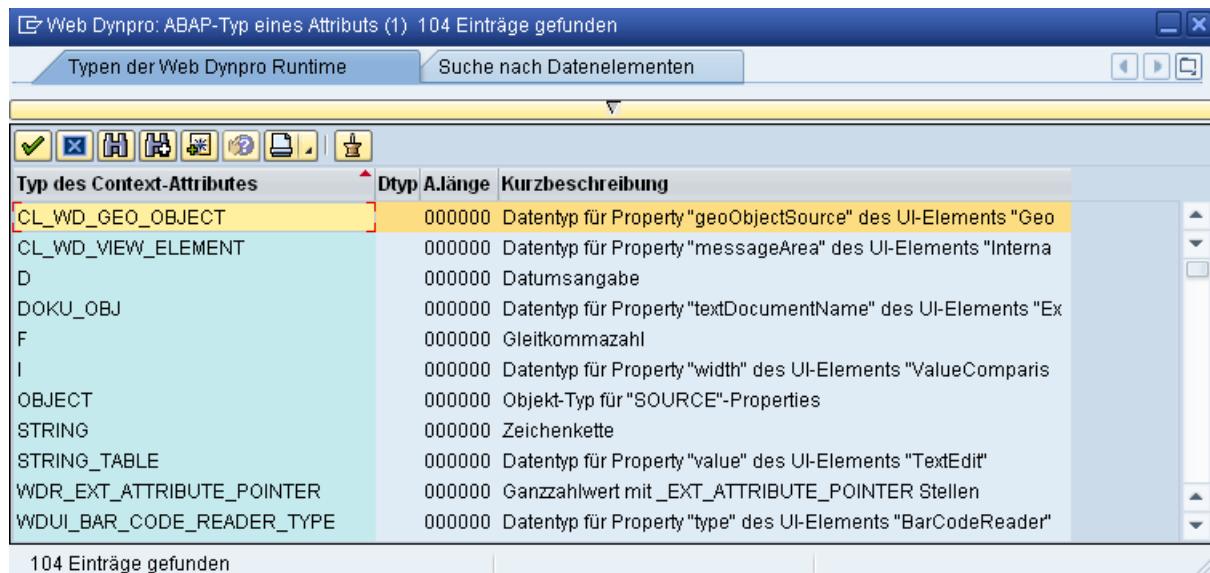


Abbildung 65: Typen der Web Dynpro Runtime in der Suchhilfe: SAP-System-Screenshot

7.7.5 Praxis: Übung zu Datenbindung und UI-Element-Eigenschaften

In dieser Übung werden Sie die Datenbindung für eine andere Eigenschaft als die value-Eigenschaft verwenden. Die Web Dynpro-Component aus der letzten Übung soll so angepasst werden, dass über eine CheckBox (ein „Häkchen“) auf der ersten View festgelegt werden kann, ob auf der zweiten View die Felder eingabebereit sein sollen.

Legen Sie einen neuen Kontextknoten **UI_PROPERTIES** mit der Kardinalität 1..1 im Context des Component Controllers an. Erstellen Sie in diesem Knoten ein Attribut **READ_ONLY**. In diesem Attribut soll die Eigenschaftsausprägung abgelegt werden. Verwenden Sie zur Typisierung dieses Elements den booleschen Typ aus der Liste der **Typen der Web Dynpro Runtime**. Sichern Sie und öffnen Sie den Context der ersten View. Sorgen Sie dafür, dass der Knoten mit dem Attribut per Context-Mapping auch hier zur Verfügung steht. Wechseln Sie anschließend zur Registerkarte **Layout**. Erstellen Sie dort ein neues UI-Element vom Typ **CheckBox**, das Sie an einer geeigneten Stelle auf dem View positionieren. Weisen Sie diesem in den Eigenschaften den Text „Eingaben auf der Ausgabemaske verbieten“ zu und stellen Sie eine Datenbindung zum Kontextattribut **READ_ONLY** her, um die Eigenschaft **checked** der CheckBox zu binden.

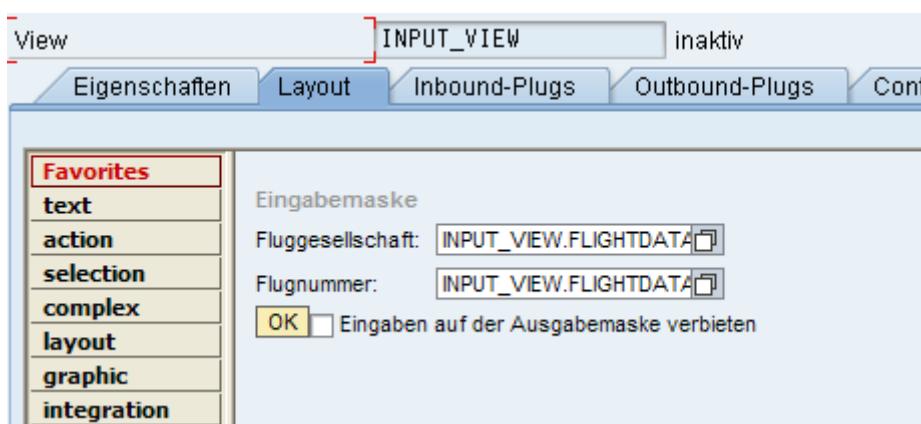


Abbildung 66: Layout mit CheckBox: SAP-System-Screenshot

Sichern Sie die View und öffnen Sie die zweite View. Passen Sie den Context dort analog zur ersten View an. Wechseln Sie anschließend auch hier zur Registerkarte **Layout**. Binden Sie die Eigenschaft **ReadOnly** der InputField-Elemente an das Kontextattribut **READ_ONLY**. Speichern und aktivieren Sie alle inaktiven Entwicklungsobjekte. Testen Sie dann Ihre Web Dynpro-Anwendung. Wenn Sie das Häkchen nicht setzen, sind nun die Felder auf der Ausgabe-View ebenfalls Eingabebereit:



Abbildung 67: Eingabebereite Felder der zweiten View: SAP-System-Screenshot

Bei der Bindung der Eigenschaft der InputField-Elemente wäre es auch möglich gewesen, die Eigenschaft genau umgekehrt zum Wert des Kontextattributs zu setzen. Hierfür ist in den Einstellungen zur Datenbindung das entsprechende Häkchen am unteren Rand des Fensters zu setzen:



Abbildung 68: Invertieren von Eigenschaften: SAP-System-Screenshot

So kann bei Bedarf eine umgekehrte Semantik realisiert werden. Dies wäre hier etwa sinnvoll gewesen, wenn der Name des Kontextattributs statt **READ_ONLY** z. B. **WRITABLE** und die Beschriftung der CheckBox auf der ersten View entsprechend umgekehrt gewesen wäre. Sie haben nun praktisch erfahren, wie Sie Eigenschaften, die über **value** hinausgehen, über den Context binden und so verändern können. Beachten Sie, dass kein Programmcode für die Oberfläche implementiert werden musste, sondern alle Angaben deklarativ erfolgten.

7.7.6 Texte aus dem Dictionary

Bei der Checkbox in der letzten Übung oder zuvor beim Button haben Sie den Text, der angezeigt werden sollte, manuell festgelegt. Die Eingabefelder hingegen wurden mit einer Datenbindung versehen und erhielten ihre Texte so aus dem Context bzw. durch die Eingaben des Anwenders im Browser. Es gibt noch eine weitere Möglichkeit, wie Elemente ihre Texte beziehen können. Im Dictionary sind bekanntlich zu den Datenelementen beschreibende Texte hinterlegt. Diese können auch in UI-Elementen von Web Dynpro verwendet werden.

Implizit haben Sie dieses Konzept bereits verwendet. Der Wizard, mit dem Sie die Eingabefelder angelegt haben, hat für diese Beschriftungen angelegt. Diese UI-Elemente des Typs **Label** haben keinen Wert für die Eigenschaft **text**:

Eigenschaft	Wert	Bin...
Eigenschaften (Label)		
Id	CARRID_LBL	
Layoutdaten	MatrixHeadData	
contextMenuBeh...	inherit	
contextMenuId		
design	standard	
enabled	<input checked="" type="checkbox"/>	
labelFor	CARRID	
text		
textDirection	inherit	
tooltip		
visible	visible	
...		

Abbildung 69: Text aus dem Dictionary: SAP-System-Screenshot

Die Eigenschaft **labelFor** mit ihrem Wert **CARRID** weist jedoch aus, dass es sich bei diesem Element um die Beschriftung für die Fluggesellschaft handelt. Die Beschriftung kann so wie gewohnt zentral im Dictionary gepflegt werden.

7.7.7 Zusammengesetzte UI-Elemente

Es gibt einige komplexere UI-Elemente, die sich aus einfacheren UI-Elementen zusammensetzen. Darunter fallen unter anderem die UI-Elemente **Table** und **Tree**.

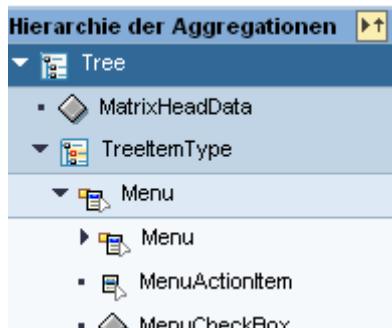


Abbildung 70: Tree als Beispiel für ein zusammengesetztes UI-Element: SAP-System-Screenshot

Die jeweiligen untergeordneten UI-Elemente der zusammengesetzten UI-Elemente werden benötigt, um überhaupt Informationen anzuzeigen. Die zusammengesetzten Elemente können obligatorische Unterelemente haben. Bei den Elementen **Group** und **Tray** ist dies nur das Element **Caption**, mit dem die Beschriftung festgelegt wird, während alle weiteren Elemente vom Entwickler festzulegen sind. Bei Elementen wie **Table** oder **Tree** liegt hingegen eine komplexe Struktur aus obligatorischen untergeordneten Elementen vor. Auch der **TransparentContainer**, den Sie aus den Übungen kennen, ist ein zusammengesetztes Element. Sie sehen in der Elementhierarchie auf der Layout-Registerkarte, dass er Unterelemente besitzt.

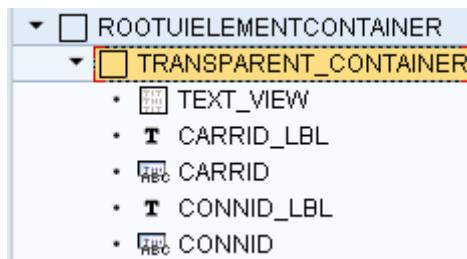


Abbildung 71: TransparentContainer aus der Übung: SAP-System-Screenshot

Das Element Table wird sehr häufig benötigt, da in betriebswirtschaftlichen Anwendungen tabellarische Daten eine große Rolle spielen. Daher wird Ihnen dieses Element ausführlicher vorgestellt.

7.7.8 Das Table-UI-Element

Mit dem **Table**-Element können Tabellen dargestellt werden. Zunächst mag hierfür ein beliebiges Containerelement ausreichend erscheinen, bei dem ein Layout mit Spalten (**MatrixLayout** oder **GridLayout**) gewählt wird. Die Funktionalität des Table-Elements macht dessen Verwendung jedoch sehr viel komfortabler. Das Table-Element setzt sich aus einer Überschrift und Spalten zusammen. Die Überschrift der Tabelle ist als **Caption**-Element umgesetzt. Die Spalten sind hingegen **TableColumn**-Elemente. Diese sind wiederum zusammengesetzte Elemente, die nur mit untergeordneten Elementen funktionieren. Sie bestehen aus einer Spaltenbeschriftung (einem Caption-Element) und einem **Zelleneditor** (**Cell Editor**). Unter dem Zelleneditor werden alle Arten von UI-Elementen verstanden, die als Zellelemente in einer Spalte dienen können. In einfachen Tabellen genügt hier das Standard-Element **Textview**, mit dem die Daten der Tabellenzelle angezeigt werden können. Komplexere Anwendungsfälle, in denen etwa auch Eingaben in die Tabelle erfolgen sollen, erfordern andere Elemente für die Zellen (als Zelleneditor). Darunter fallen etwa die Elemente **InputField**, **DropDownByKey**, **CheckBox** oder **Button**. Jede Tabelle muss mindestens eine Spalte besitzen.

The screenshot shows a table with two columns labeled 'Spalte 1' and 'Spalte 2'. Above the table is a header row labeled 'Überschrift'. To the right of the table is a detailed view of its structure in the layout editor:

- CONTEXT_MENUS**
- ROOTUIELEMENTCONTAINER**
- TABLE** (highlighted in yellow)
 - TABLECOLUMN [Columns]**
 - ZELLENEDITOR_1**
 - CAPTION_1 [Header]**
 - TABLECOLUMN_1 [Columns]**
 - ZELLENEDITOR_2**
 - CAPTION_2 [Header]**
 - CAPTION [Header]**

Abbildung 72: Beispiel für eine Tabelle: SAP-System-Screenshot

Die obige Abbildung zeigt ein Beispiel für eine Tabelle im Layout-Editor. Sie enthält zwei Spalten. Die Tabellenüberschrift wurde im Element **CAPTION [Header]** festgelegt. Die Spaltenbeschriftungen wurden in den entsprechenden **Caption**-Elementen unterhalb der **TableColumn**-Elemente eingegeben. Im Fußbereich der Tabelle (auf der linken Seite der Abbildung) sehen Sie einige Elemente zur schnellen Navigation zwischen den Zeilen der Tabelle.

Ein Table-Element kann als ganzes zur Datenbindung verwendet werden. So können Daten zweidimensional ausgegeben werden. Für die Datenbindung muss im Context ein Knoten gewählt werden, der die Kardinalität **0..n** oder **1..n** besitzt, da eine Tabelle, die technisch nur eine Zeile enthalten kann, nicht sinnvoll ist. Auch die **Lead-Selection** wirkt sich auf die Darstellung der Tabelle aus. Das Element im Context, das als Lead-Selection des Kontextknotens ausgewählt ist, wird bei der Darstellung der Tabelle hervorgehoben. Zum Anlegen der Bindung gibt es zwei Möglichkeiten. Die erste Möglichkeit besteht in der Verwendung des Wizards zum Anlegen der Tabelle. Hierbei wird unter anderem die Datenbindung konfiguriert. Die zweite Möglichkeit ist das manuelle Einrichten der Datenbindung. Es wird hier empfohlen, in der Elementhierarchie mit der rechten Maustaste auf die Tabelle zu klicken und **Binding erzeugen** auszuwählen.

Es ist naheliegend, dass die Spalten der Tabelle an Attribute des Knotens gebunden werden sollten, der für die Bindung der Tabelle verwendet wurde. Jedes Attribut des Knotens mit einem einfachen Datentyp kann für eine Spalte verwendet werden. Es ist aber auch zulässig, ein TableColumn-Element an eine Spalte eines Attributs zu binden, das sich nicht unterhalb des Knotens, an den die Tabelle gebunden wurde, befindet. Dies hat zur Folge, dass die Werte in allen Zeilen gleich sind.

Die Überschrift der Spalte kann wie in Abschnitt 7.7.6 beschrieben aus dem Dictionary bezogen werden. Hierfür muss die Eigenschaft **Text** des Caption-Elementes leer gelassen werden, und das in der Spalte dargestellte Kontextattribut mit einem Dictionary-Typen typisiert sein.

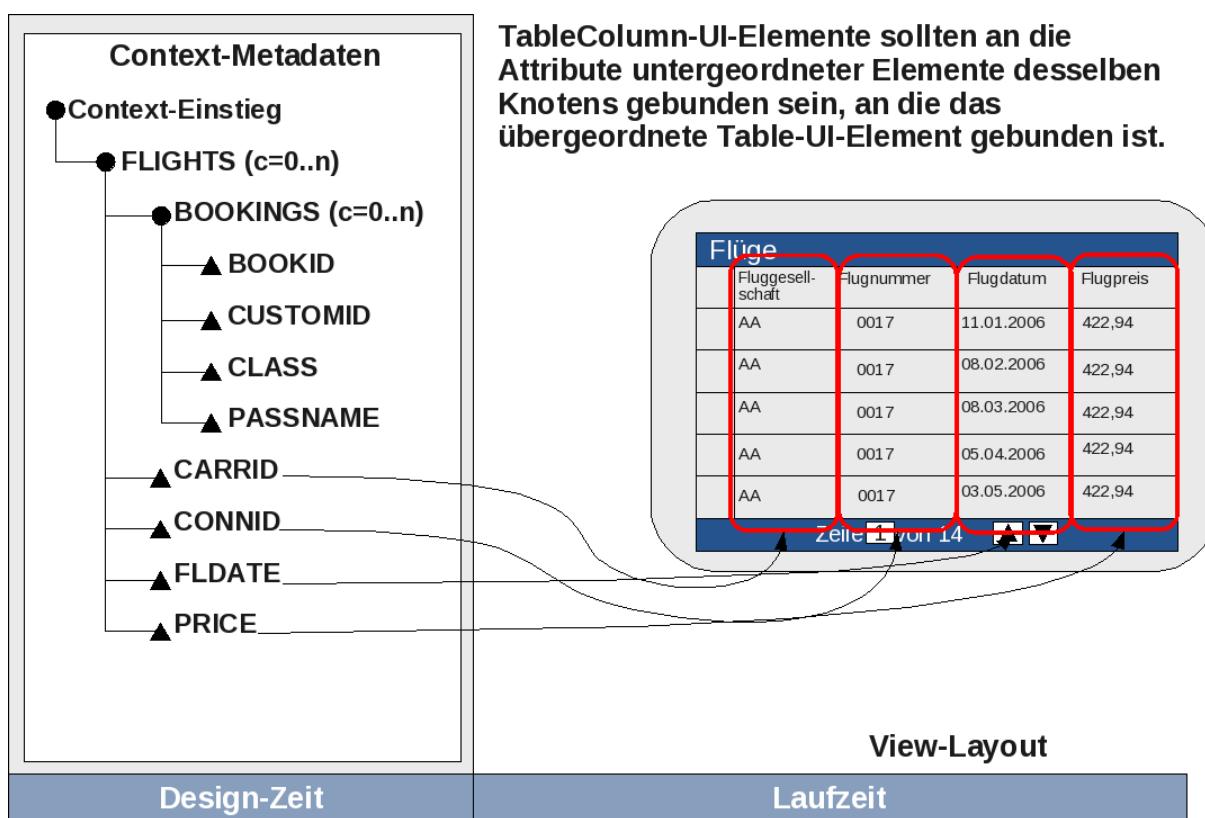


Abbildung 73: Datenbindung von TableColumn-Elementen

Die Abbildung zeigt, wie die Attribute des Kontextknotens **FLIGHTS** für entsprechende Spalten eines Table-UI-Elements verwendet werden. Für jede Spalte ist zu entscheiden, welches UI-Element als Zelleneditor verwendet werden soll. Das Wort Zelleneditor ist etwas verwirrend, da nicht nur editierbare Elemente verwendet werden können, etwa auch das

Standardelement TextView. Ein InputField sorgt dafür, dass die Zellen standardmäßig bearbeitet werden können. Weitere Elemente sind auch möglich, etwa ein Button. Dieser zeigt dann als Text den Feldinhalt an und löst ein Client-Ereignis aus, wenn er angeklickt wird.

Am Beginn jeder Tabellenzeile befindet sich eine Schaltfläche zur Auswahl dieser Zeile. Diese Schaltflächen kennen Sie bereits von Tabellen in der herkömmlichen SAP-Oberfläche. Bei **Table**-Elementen wirkt sich diese Auswahl auf die Lead-Selection des Kontextknotens aus. Der Klick löst einen Roundtrip aus, bei dem die Lead-Selection auf das ausgewählte Kontextelement gesetzt wird.

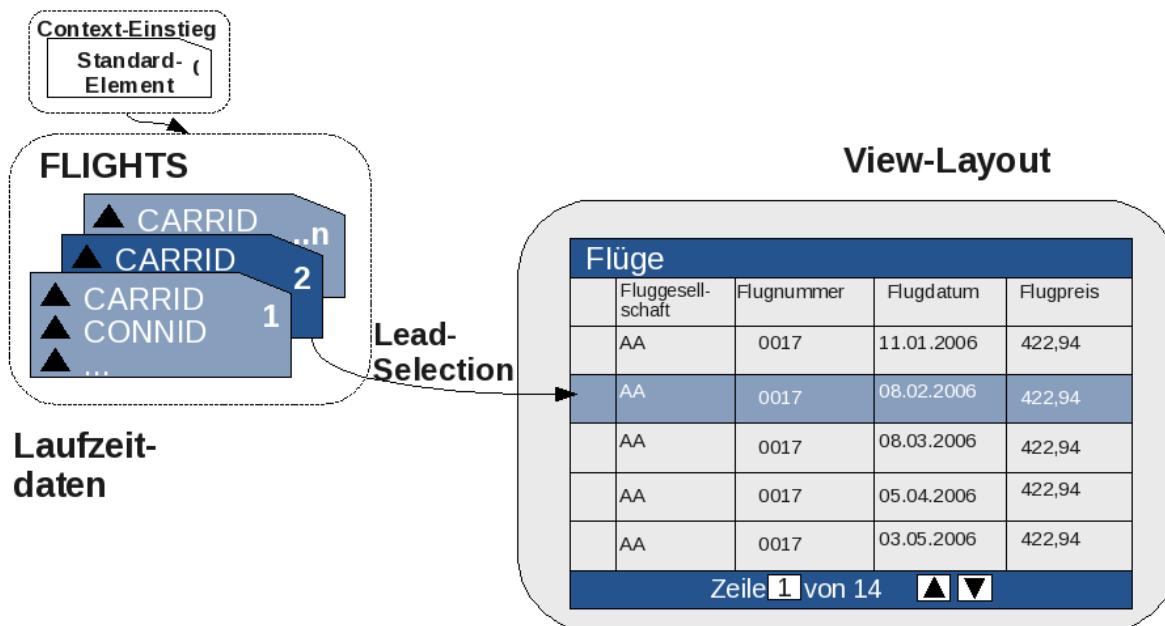


Abbildung 74: Lead-Selection in der Oberfläche

Die obige Abbildung veranschaulicht das Prinzip. Auf der linken Seite sind die Kontextelemente, auf der rechten ist die Tabelle dargestellt. Die Tabelle enthält so viele Zeilen, wie Elemente im Kontextknoten vorhanden sind, an den die Tabelle gebunden ist. Im Beispiel sind nicht alle Kontextelemente explizit sichtbar. Der Anwender klickt hier nun auf die Selektions-Schaltfläche am Beginn der zweiten Zeile. Es wird ein Roundtrip ausgelöst, die Lead-Selection auf das entsprechende Kontextelement gesetzt und in der Tabelle die zweite Zeile hervorgehoben.

Zusätzlich zur Lead-Selection-Änderung können Sie auch eine Aktion definieren, die Sie dem Web Dynpro-Ereignis `onLeadSelect` zuordnen. Hierdurch wird beim Wechsel der selektierten Spalte auch die Hook-Methode `wddobeforeaction` aufgerufen und die entsprechende Aktionsbehandlermethode der View wird ausgeführt. Hier kann zusätzlich benötigte Funktionalität hinterlegt werden.

Es ist auch möglich, mehrere Zeilen einer Tabelle gleichzeitig zu selektieren. Die Voraussetzung dafür ist, dass der gebundene Kontextknoten eine **Auswahlkardinalität** (Eigenschaft **Selection**) besitzt, die dies zulässt. Die möglichen Werte sind hier wie bei der Knotenkardinalität 0..1, 1..1, 0..n und 1..n, wobei die letzten beiden eine Mehrfachauswahl erlauben.

Eigenschaft	Wert
Knoten	
Knotenname	FLIGHTDATA
Dictionary-Struktur	SFLIGHT
Kardinalität	1..1
Selection	0..1
Initialisierung Lead-Selection	<input checked="" type="checkbox"/>
Singleton	<input type="checkbox"/>
Supply-Funktion	
Mapping-Pfad	ZZ_9999_WD_DATA.COMPONENTCONTROLLER.FLIGHDATA

Abbildung 75: Auswahlkardinalität eines Kontextknotens: SAP-System-Screenshot

Der Anwender führt die Mehrfachauswahl durch gedrückt halten der Shift- oder Strg-Taste beim Anklicken der Selektionsschaltflächen durch. Um aus dem Programmcode die ausgewählten Elemente erreichen zu können, stellt jeder Kontextknoten die Methode **get_selected_elements** zur Verfügung. Diese liefert die Elemente in Form einer internen Tabelle des Typs **WDR_CONTEXT_ELEMENT_SET** zurück. Diese besitzt einen Parameter **INCLUDING_LEAD_SELECTION**, der angibt, ob die Lead-Selection im Ergebnis enthalten sein soll oder nicht.

Das Verhalten beim Auswählen einer bzw. mehrerer Zeilen ist durch die Wahl eines Wertes für die Eigenschaft **SelectionMode** des Table-Elements änderbar. Wird hier der Wert **auto** ausgewählt, ist das Verhalten wie zuvor beschrieben. Es sind jedoch alternativ die Werte **single**, **multi**, **none**, **singleNoLead** und **multiNoLead** möglich. Die Auswirkungen beschreibt die folgende Tabelle.

Tabelle 1: Verhalten bei der Zeilenauswahl

SelectionMode	Mindestzahl ausgewählter Zeilen	Höchstzahl ausgewählter Zeilen	Auswahl einer Zeile löst Roundtrip aus	Auswahl einer Zeile setzt Lead-Selection
auto	Anhand Auswahl-kardinalität	Anhand Auswahl-kardinalität	Ja	Ja
single	Anhand Auswahl-kardinalität	1	Ja	Ja
multi	Anhand Auswahl-kardinalität	Anhand Auswahl-kardinalität	Ja	Ja
none	0	0	Nein	Nein
singleNoLead	Anhand Auswahl-kardinalität	1	Nein	Nein
multiNoLead	Anhand Auswahl-kardinalität	Anhand Auswahl-kardinalität	Nein	Nein

7.7.9 Praxis: Übung zum Tray-UI-Element

Ein anderes Beispiel für ein zusammengesetztes UI-Element ist das **Tray**-Element. Hierbei handelt es sich um einen Bereich auf einer View, der ein- und ausgeklappt werden kann. So lassen sich Teile einer View unsichtbar machen.

Ein Tray kann eine Beschriftung besitzen. Diese wird im untergeordneten Element **Caption** festgelegt. Darüber hinaus lassen sich weitere Unterelemente nach Wahl dort unterbringen.

Legen Sie eine neue Web Dynpro-Component an, um die Funktionsweise kennen zu lernen. Die Component soll **ZZ #####_WD_TRAY** heißen und eine View enthalten, deren Namen Sie frei wählen dürfen. Öffnen Sie die View und legen Sie ein UI-Element des Typs **Tray** an. Sie finden dieses im Bereich **layout**. Sie sehen anhand der Elementhierarchie, dass automatisch das untergeordnete Element **Caption** angelegt wurde:

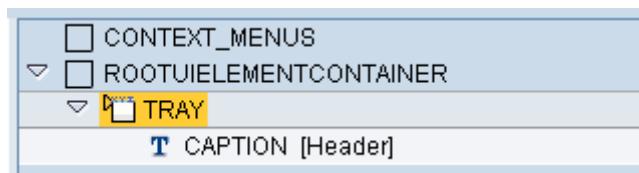


Abbildung 76: Automatisch angelegtes untergeordnetes UI-Element : SAP-System-Screenshot

Wählen Sie dieses Element aus und setzen Sie dessen Eigenschaft **text** auf einen Wert Ihrer Wahl. Innerhalb des Trays soll nun ein Bild angezeigt werden. Dieses soll aus dem **MIME-Repository** bezogen werden. Das MIME-Repository ist eine Ablage für Grafiken, Style Sheets, Symbole usw. im SAP-System. Um die Elemente des MIME-Repositorys zu durchblättern, bietet das System einen Browser. Sie finden diesen im Object Navigator im linken oberen Bereich:

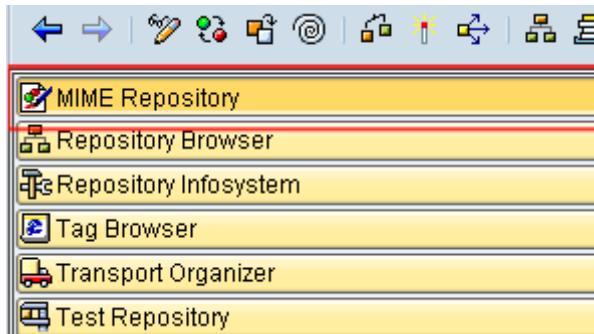


Abbildung 77: Zugriff auf das MIME-Repository: SAP-System-Screenshot

Wenn Sie auf die Schaltfläche klicken, die in der obigen Abbildung rot umrandet ist, erscheint auf der linken Seite ein Navigationsbaum, mit dem im hierarchisch strukturierten Repository auf die Elemente zugegriffen werden kann. Öffnen Sie dort den Pfad **SAP -> PUBLIC -> BC -> Pictograms**. Doppelklicken Sie dort auf airplane.gif (ggf. müssen Sie die Spalte **Name** etwas breiter machen, um dies lesen zu können). Es erscheint daraufhin eine Vorschau des Bildes im linken unteren Bereich des Fensters. Um das Bild in das Tray-Element einzufügen, benötigen Sie zunächst ein **Image**-Element. Erzeugen Sie dieses per Rechtsklick auf das Tray-Element in der Elementhierarchie der View (wählen Sie dort wie gewohnt **Element Einfügen** und wählen Sie einen Namen Ihrer Wahl sowie den Typ **Image**). Ziehen Sie nun das Bild (d. h. den Namen, nicht das Vorschaubild) aus dem Navigationsbaum auf den Namen Ihres Image-Elements in der Elementhierarchie der View. Falls dies nicht funktioniert (dies ist in manchen Releases der Fall), rufen sie stattdessen die F4-Hilfe des **source**-Attributs des **Image**-Elements auf. Im erscheinenden Fenster können sie das Flugzeugbild im Karteireiter **Piktogramme** auswählen.

Speichern Sie die View und aktivieren Sie alle inaktiven Objekte (ignorieren Sie ggf. die Warnung zur Relativität der URL). Erstellen Sie eine Anwendung zu Ihrer Component.

Speichern Sie diese und testen Sie dann die Anwendung. Durch Klick auf (in der folgenden Abbildung rot umrandet) können Sie den Bereich mit dem Bild nun zuklappen und danach durch Klick auf wieder aufklappen.

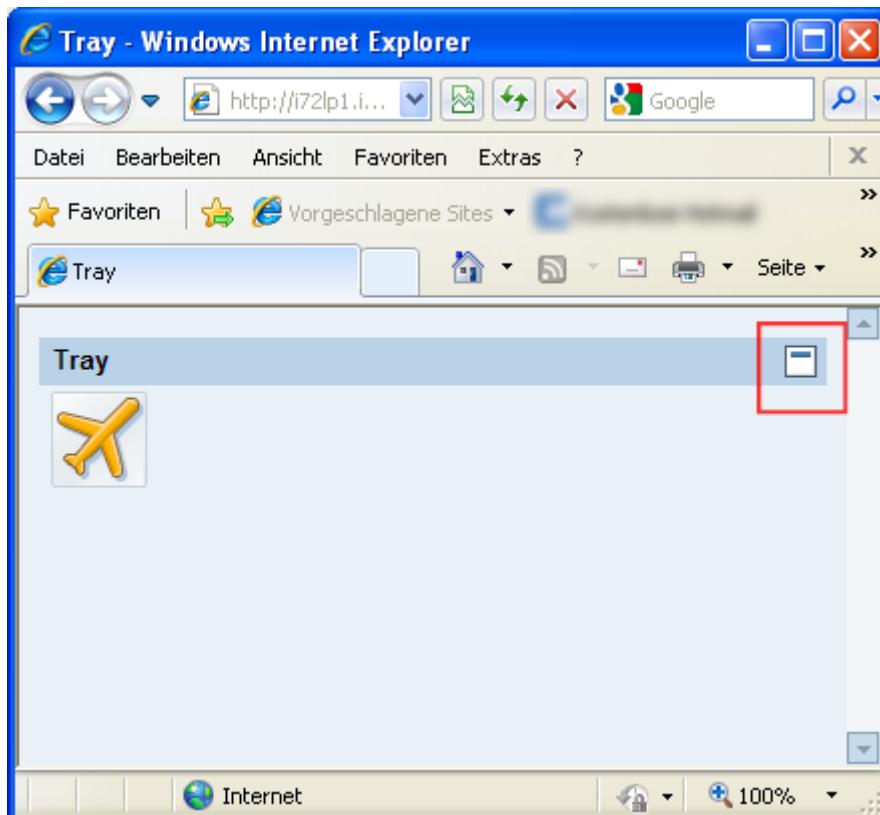


Abbildung 78: Oberfläche mit Tray und Image: SAP-System-Screenshot

7.7.10 Praxis: Übung zur Oberflächengestaltung

In dieser Übung werden Sie eine Web Dynpro-Anwendung erstellen, mit der anhand von Start- und Zielort Flüge aus der bekannten Flugdatenbank gesucht werden. Die Ergebnisse werden in einer Tabelle dargestellt.

Legen Sie für diese Übung eine neue Component mit dem Namen **ZZ_###_WD_FLIGHTS** an. Benennen Sie das Window **MAIN_WINDOW**. Die Anwendung soll zwei Views besitzen (**INPUT_VIEW** und **RESULT_VIEW**). Sorgen Sie dafür, dass beide Views angelegt und im Window eingebettet sind.

Zum Lesen der Flüge steht Ihnen ein Funktionsbaustein **BAPI_FLIGHT_GETLIST** zur Verfügung. Klicken Sie mit der rechten Maustaste auf Ihre Component im Navigationsbaum des Object Navigator. Wählen Sie aus dem Kontextmenü den Pfad **Anlegen -> Service-Aufruf**. Es öffnet sich daraufhin ein Wizard, der Sie unterstützt. Bestätigen Sie das Einstiegsbild des Wizards mit **Weiter**. Wählen Sie im nächsten Bild **Existierenden Controller nutzen** und wählen Sie unter Nutzung der F4-Hilfe als Controller den Component-Controller aus.

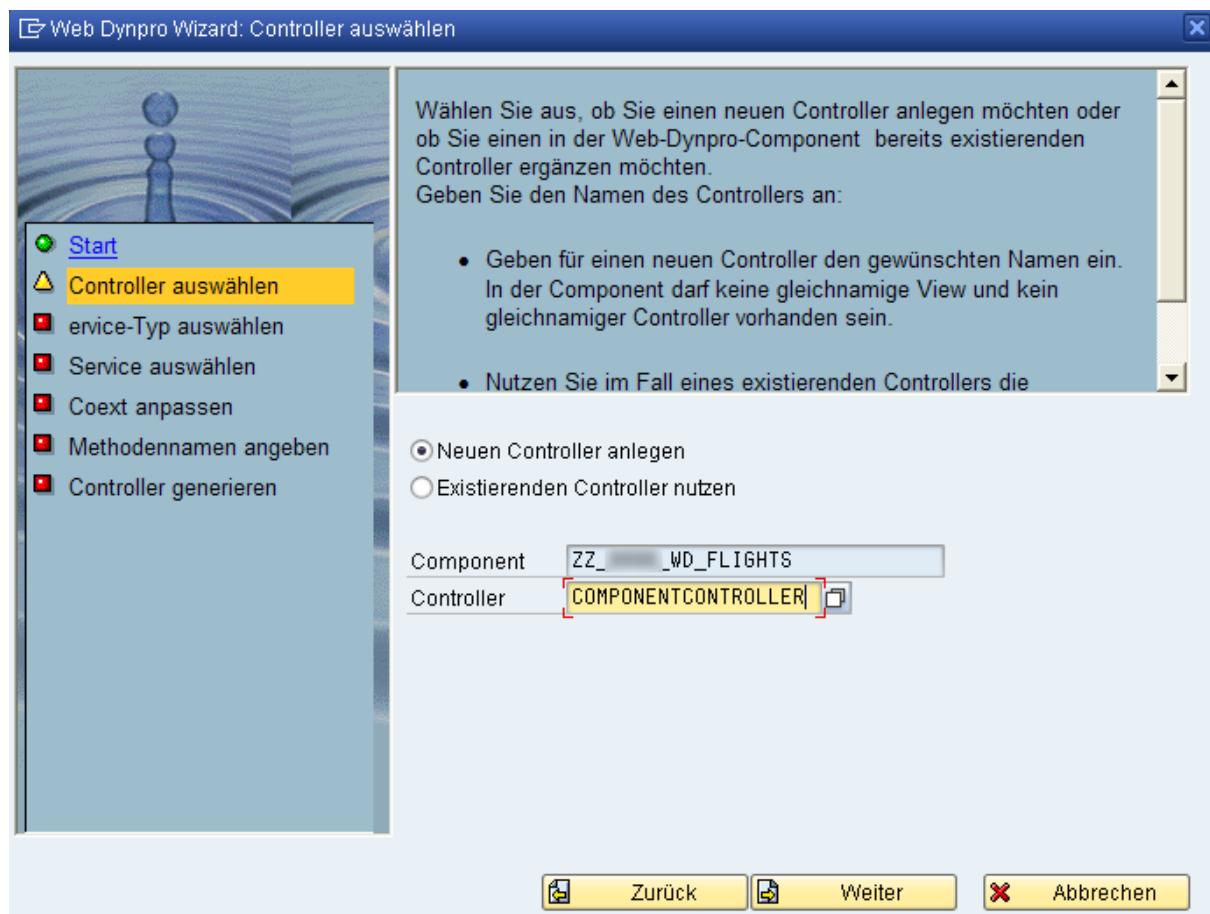


Abbildung 79: Controllerauswahl im Wizard: SAP-System-Screenshot

Fahren Sie fort und wählen Sie auf dem folgenden Bild **Funktionsbaustein**, da es sich bei **BAPI_FLIGHT_GETLIST** um einen solchen handelt. Geben Sie auf dem nächsten Bild dementsprechend **BAPI_FLIGHT_GETLIST** in das Feld **Funktionsbaustein** ein und lassen Sie das Feld Destination leer.

In der nächsten Maske wird die Schnittstelle des Funktionsbausteins auf den Context abgebildet. Die UI-Relevanten Parameter werden hier als Kontextknoten bzw. -Attribute ausgewählt. Die gewünschte Anwendung soll auf der ersten View nach Start- und Zielort fragen, um auf der zweiten View eine Flugliste (als Tabelle) anzuzeigen. Daher werden die Parameter **DESTINATION_FROM**, **DESTINATION_TO** und **FLIGHTLIST** benötigt. Wählen Sie für diese in der Spalte **Objektart** den Eintrag **Context (-Knoten / -Attribut)** aus. Eine Strukturierung durch Knoten für den Funktionsbaustein und Import- / Exportparameter ist bereits vorgegeben. Fahren Sie mit **Weiter** fort.

Akzeptieren Sie auf dem folgenden Bild den Vorschlagswert **EXECUTE_BAPI_FLIGHT_GETLIST** für den Namen der Servicemethode. Fahren Sie mit **Weiter** fort und bestätigen Sie mit **Fertigstellen** die Angaben im Wizard. Schauen Sie sich den Context des Component-Controllers an und klappen Sie alle Knoten aus. Sie sehen die Strukturierung anhand der Schnittstelle des Funktionsbausteins.

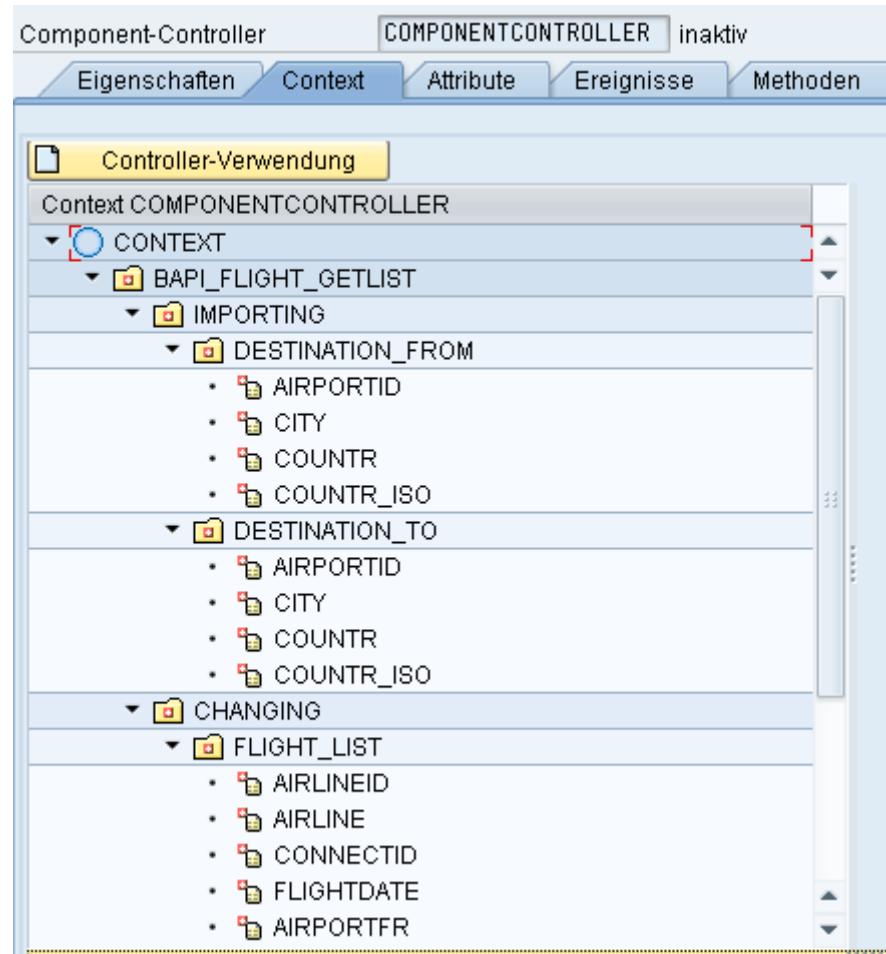


Abbildung 80: Automatisch generierte Kontextknoten und -attribute: SAP-System-Screenshot

Schauen Sie sich auch die Kardinalitäten der Knoten DESTINATION_FROM, DESTINATION_TO und FLIGHT_LIST an. Bei den Parametern DESTINATION_FROM und DESTINATION_TO handelt es sich um Strukturen, deshalb besitzen die Kontextknoten die Kardinalität 1..1. Die Flugliste ist hingegen eine interne Tabelle, weshalb der Kontextknoten die Kardinalität 0..n besitzt.

Wechseln Sie nun zur Registerkarte **Methoden**. Sie sehen, dass dort die Methode **EXECUTE_BAPI_FLIGHT_GETLIST** angelegt wurde. Diese Methode kapselt den Aufruf des Funktionsbausteins und schreibt die Ergebnisse in den Context.

Als nächstes soll die View **INPUT_VIEW** gestaltet werden. Um die Eingabemaske zu erzeugen, ist es sinnvoll, den Wizard zu benutzen. Dafür muss zunächst der Context um die benötigten Knoten **DESTINATION_FROM** und **DESTINATION_TO** ergänzt werden. Übernehmen Sie diese wie in den vergangenen Übungen aus dem Context des Component Controllers und definieren Sie das Context Mapping. Wechseln Sie dann zur Registerkarte **Layout**. Rufen Sie den Web Dynpro Code Wizard auf, und wählen Sie das Template **Form**. Wählen Sie auf der folgenden Maske über die entsprechende Schaltfläche aus dem Context den Knoten **DESTINATION_FROM**. Auf der View soll nur ein Feld für den Ort erscheinen. Entfernen Sie daher für die anderen Felder das Häkchen in der Spalte **Binding**. Stellen Sie zudem sicher, dass **Form in neuem Container erzeugen** und **Section Header einfügen** angehakt sind, und geben Sie eine passende Überschrift im Feld neben **Section Header einfügen** ein. Bestätigen Sie anschließend.

In der Elementhierarchie der View finden Sie wie erwartet einen TransparentContainer mit den entsprechenden untergeordneten Elementen. Setzen Sie im untergeordneten **TextView**-

Element die Eigenschaft **Text** auf einen sinnvollen Wert. Klicken Sie danach auf **ROOTUIELEMENTCONTAINER** (oben rechts in der Elementhierarchie). Diese Auswahl dient dazu, dass das Formular für die Zielstadt später auf der richtigen Ebene eingefügt wird. Erstellen Sie anschließend auch das Eingabefeld für die Zielstadt unter Zuhilfenahme des Wizards.

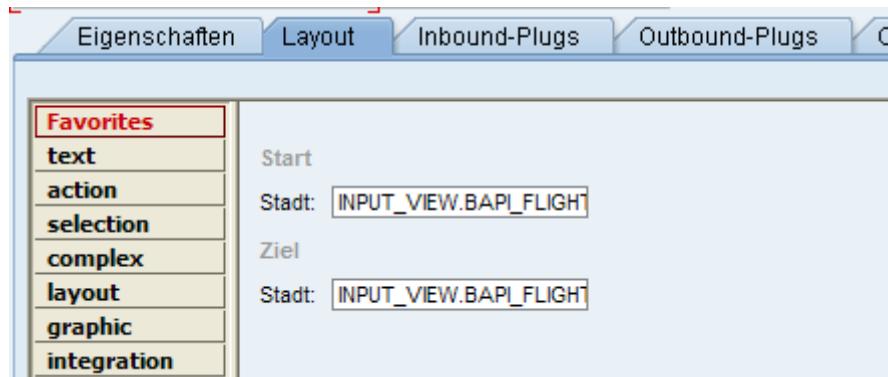


Abbildung 81: Vom Wizard generierte Felder: SAP-System-Screenshot

Wenn Ihnen das Aussehen der Maske nicht gefällt, können Sie auch die Texte der Labels der Eingabefelder anpassen. Sichern Sie anschließend die View und öffnen Sie die View **RESULT_VIEW**. Sorgen Sie dort zunächst dafür, dass der Kontextknoten **FLIGHT_LIST** auch im Context der View zur Verfügung steht und ein Mapping zum Component Controller besteht. Wechseln Sie dann zur Registerkarte **Layout**.

Rufen Sie hier den Wizard auf, um die Tabelle anzulegen. Wählen Sie als Template nicht wie zuvor **Form**, sondern **Table** aus. Wählen Sie den Kontextknoten **FLIGHT_LIST** aus. Der Zelleditor ist auf TextView voreingestellt. Da es sich um eine reine Anzeigemaske handelt, kann diese Einstellung beibehalten werden. Weiterhin sollen alle Felder gebunden sein, was auch der Voreinstellung entspricht. Bestätigen Sie den Wizard.

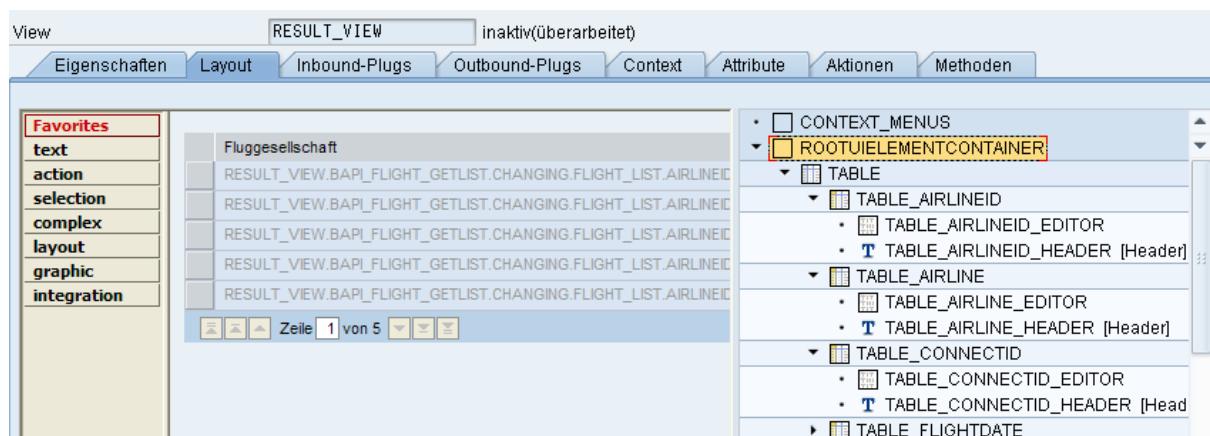


Abbildung 82: Durch den Wizard generierte Tabelle: SAP-System-Screenshot

Der Wizard hat nun eine Tabelle angelegt. Klappen Sie die Elementhierarchie aus, um die untergeordneten Elemente sehen zu können.

Es fehlt bislang noch eine Navigation zwischen den Views. Legen Sie hierzu entsprechende Buttons, Plugs und Navigationslinks an.

Für eine funktionierende Anwendung muss nun die Methode, die die Daten liefert, auch aufgerufen werden. Damit dies beim Aufruf der View **RESULT_VIEW** geschieht, wird der

zugehörige Methodenaufruf in der Behandlermethode des Inbound-Plugs untergebracht. Öffnen Sie dazu die Registerkarte **Methoden** dieser View, und klicken Sie doppelt auf den Methodennamen (dieser beginnt mit **HANDLE**). Sie gelangen so zur Implementierung der Methode. Erzeugen Sie den Methodenaufruf zwischen `method` und `endmethod`. Klicken Sie dazu wieder auf die Schaltfläche  um den Wizard aufzurufen. Wählen Sie dort **Methodenaufruf im verwendeten Controller** (auf dem Tab **Allgemein**), geben Sie als **Controller-Name** den Component-Controller und als **Methodenname** den Namen Ihrer Service-Methode an. Nutzen Sie hierbei die Werthilfe (eine normale Eingabe ist nicht möglich, da die Felder gesperrt sind).



Abbildung 83: Wizard für den Methodenaufruf: SAP-System-Screenshot

Der Wizard generiert daraufhin den folgenden Aufruf:

```
DATA lo_componentcontroller TYPE REF TO ig_componentcontroller .  
lo_componentcontroller = wd_this->get_componentcontroller_ctrl( ).  
  
lo_componentcontroller->execute_bapi_flight_getlist( ).
```

Abbildung 84: Generierter Code: SAP-System-Screenshot

Sie sehen hier, dass zunächst eine Referenzvariable definiert wird, über die der Component-Controller referenziert wird. Hierüber wird schließlich die Methode aufgerufen. Sie benötigt keine Parameter, da Sie direkt mit dem Context arbeitet.

Sichern Sie die View und aktivieren Sie alle Objekte. Erstellen Sie anschließend eine Anwendung für die Component und testen Sie diese.

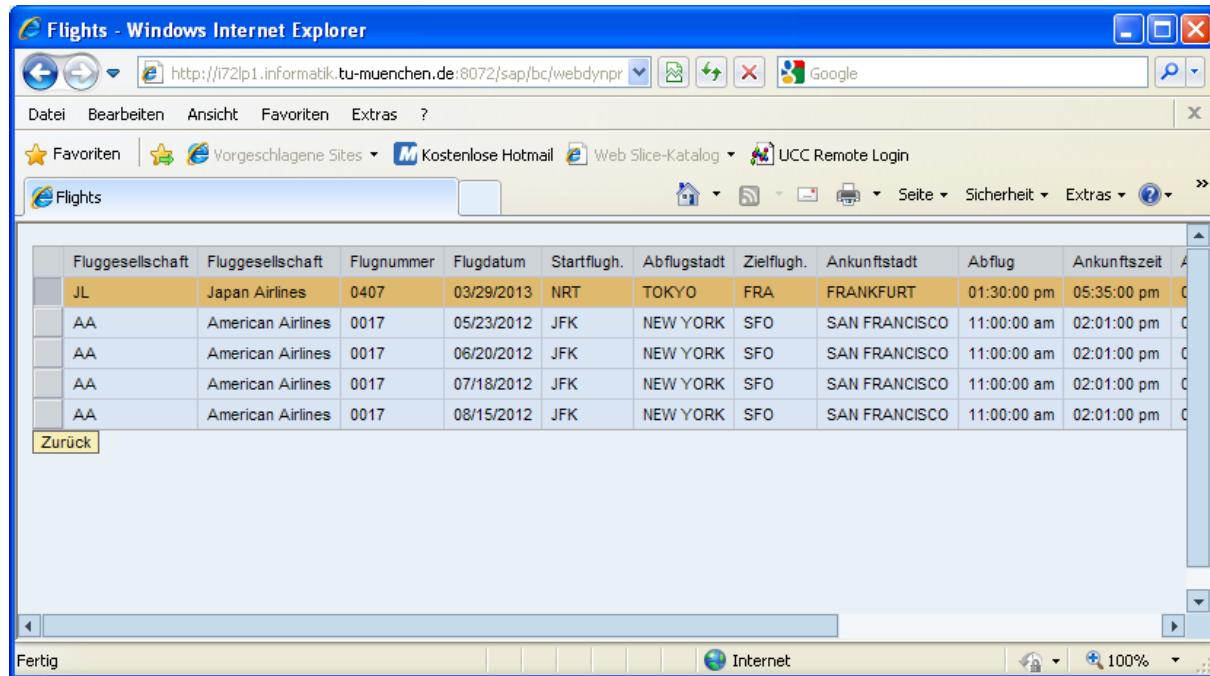


Abbildung 85: Ausgabe des Programms: SAP-System-Screenshot

Sie haben in dieser Übung gelernt, wie Sie Tabellen in Views erzeugen können. Weiterhin haben Sie eine Service-Methode angelegt, um Daten in den Context zu schreiben. Erstmals sind Sie im Zusammenhang mit Web Dynpro tatsächlich mit ABAP-Code konfrontiert worden. Bei der Übung konnten Sie jedoch die großzügige Unterstützung durch den Wizard genießen, der viel manuelle Arbeit abnimmt. Die Programmierung wird im folgenden Unterkapitel vertieft werden.

7.8 Programmierung in Web Dynpro-Components

Nachdem Sie nun schon mit den ersten Zeilen ABAP-Code im Web Dynpro-Konzept konfrontiert wurden, wird dieses Thema nun vertieft. So widmet sich dieses Kapitel den Hook-Methoden, die Sie im Theorieteil zur Web Dynpro-Architektur bereits gesehen haben. Darüber hinaus sind aber auch eigene Methoden und Attribute ein Thema. Ein Schwerpunkt des Unterkapitels liegt auf Zugriffen auf den Context.

7.8.1 Methoden von Controllern

Technisch gesehen sind alle Web Dynpro-Controller Klassen. Diese Klassen werden beim Anlegen von Components bzw. Controllern automatisch erzeugt und der Quellcode zur Implementierung der Controller generiert. Eine wichtige Rolle für die Implementierung der Logik einer Web Dynpro-Anwendung nehmen die sogenannten **Hook-Methoden** ein. Hierbei handelt es sich um vordefinierte Methoden, die zu bestimmten Zeitpunkten zur Laufzeit einer Web Dynpro-Anwendung aufgerufen werden. Der Code der Methoden wird vom Entwickler der Anwendung implementiert, beim Anlegen eines Controllers sind die Hook-Methoden leer. Außer den Hook-Methoden können mit Supply-Funktionen, Ereignisbehandlernmethoden und Instanzmethoden weitere Methoden durch den Entwickler angelegt werden.

Zwei Hook-Methoden stehen in allen Controller-Typen zur Verfügung. Die Methode **wddoinit** wird als erste Methode in der Lebensdauer des Controllers aufgerufen. Der Aufruf

erfolgt nur einmal im Lebenszyklus. Sie bietet sich an, um einmalige Initialisierungen vorzunehmen, die vor allen anderen Methodenaufrufen erledigt sein müssen. Die Methode **wddoexit** stellt das Gegenteil dazu dar: Diese Methode ist die letzte, die in der Lebensdauer des Controllers aufgerufen wird. Auch bei dieser Methode erfolgt pro Lebenszyklus nur ein einziger Aufruf. Die Methode eignet sich, um Bereinigungen durchzuführen. Beide Methoden sind bei Erzeugung des Controllers zunächst leer. Sie werden erst dann aufgerufen, wenn sie auch Code enthalten.

Methoden				
Methode	Methoden-...	Interface	Beschreibung	
WDDOAPPLICATIONSTATEC	Methode	<input type="checkbox"/>	Behandlung bei Suspend und Resume einer...	
WDDOBEFORENAVIGATION	Methode	<input type="checkbox"/>	Fehlerbehandlung vor der Navigation durch ...	
WDDOEXIT	Methode	<input type="checkbox"/>	Cleanup-Methode des Controllers	
WDDOINIT	Methode	<input type="checkbox"/>	Initialisierungs-Methode des Controllers	
WDDOPOSTPROCESSING	Methode	<input type="checkbox"/>	Vorbereitung der Ausgabe	

Abbildung 86: Methoden WDDOEXIT und WDDOINIT eines Controllers: SAP-System-Screenshot

Der Component-Controller enthält drei weitere Methoden, von denen zwei relevant sind. Die Methode **wddobeforenavigation** kommt beim Auslösen eines Client-Ereignisses ins Spiel. Sie wird nach dem Abarbeiten der Aktionsmethode aufgerufen, die dem Client-Ereignis zugeordnet ist, und direkt vor dem Abarbeiten des Ereignisses in der Navigationswarteschlange durch das Framework. Hier können Fehler behandelt werden. Die Methode **wddopostprocessing** wurde für besonders komplexe Web Dynpro-Anwendungen konzipiert, bei denen vor einem Schritt im Geschäftsprozess Daten aus mehreren Components validiert werden müssen. Sie ist die letzte Methode, die vor dem Senden der Seite an den Client abgearbeitet wird.

Methoden				
Methode	Methoden-...	Interface	Beschreibung	
WDDOAPPLICATIONSTAT...	Methode	<input type="checkbox"/>	Behandlung bei Suspend und Resume einer...	
WDDOBEFORENAVIGATION	Methode	<input type="checkbox"/>	Fehlerbehandlung vor der Navigation durch ...	
WDDOEXIT	Methode	<input type="checkbox"/>	Cleanup-Methode des Controllers	
WDDOINIT	Methode	<input type="checkbox"/>	Initialisierungs-Methode des Controllers	
WDDOPOSTPROCESSING	Methode	<input type="checkbox"/>	Vorbereitung der Ausgabe	

Abbildung 87: Die Methoden WDDOBEFORENAVIGATION und WDDOPOSTPROCESSING: SAP-System-Screenshot

View-Controller verfügen über zwei spezifische Hook-Methoden. Die Methoden **wddobeforeaction** aller View-Controllers der zuvor dargestellten View-Komposition werden nach dem Auslösen eines Client-Ereignisses als erste Methoden ausgeführt, und somit noch vor der Behandlermethode für das Client-Ereignis. Die Methode **wdomodifyview** ist die einzige Methode, die einen Zugriff auf die UI-Elementenhierarchie des View-Controllers erlaubt. Sie kann für dynamische Änderungen der Elementenhierarchie benutzt werden. Sie besitzt zwei Importparameter, die vom Framework übergeben und in der Methode verwendet werden

können: VIEW dient als Referenz auf die Elementhierarchie, während FIRST_TIME mit einem Boolean-Wert aussagt, ob die Methode bereits einmal im Lebenszyklus des View-Controllers ausgeführt wurde oder nicht.

Methode	WDDOMODIFYVIEW			
Parameter	DeklArt	RefTo	Opt	Bezugstyp
FIRST_TIME	Importing	<input type="checkbox"/>	<input type="checkbox"/>	WDY_BOOLEAN
VIEW	Importing	<input checked="" type="checkbox"/>	<input type="checkbox"/>	IF_WD_VIEW

Abbildung 88: Schnittstelle der Methode WDDOMODIFYVIEW: SAP-System-Screenshot

Außer diesen vordefinierten Methoden können Sie beliebige weitere Methoden in einem Controller anlegen. Dies geschieht auf der Registerkarte **Methoden**. Dort stehen drei Alternativen in der Spalte **Methoden-Typ** zur Auswahl. Der Typ **Methode** kennzeichnet normale Instanzmethoden. Der Methodentyp **Ereignisbehandler** kennzeichnet eine Ereignisbehandlermethode, die in der Spalte Ereignis statisch für jedes Ereignis registriert werden kann, das in einem verwendeten Controller definiert ist. Der Typ **Supply-Funktion** kennzeichnet Methoden, die an Kontextknoten gebunden werden können, um diese in Abhängigkeit von der Lead-Selection des übergeordneten Knotens mit den richtigen Werten zu belegen, wenn dieser als nicht gültig markiert ist und auf ihn zugegriffen wird. Durch das Kennzeichen **Interface** kann eine Methode in das Component-Interface aufgenommen werden und ist dann für andere Components sichtbar. Innerhalb der Component sind die Methoden hingegen stets öffentlich. Andere Controller können somit darauf zugreifen, vorausgesetzt der rufende Controller deklariert den Controller, der die Methode enthält, als verwendeten Controller. Implementierung und Schnittstelle einer Methode werden durch doppelklick auf den Methodennamen erreicht.

7.8.2 Attribute von Controllern

Neben den Methoden besitzt jeder Controller auch Attribute. Die Attribute sind für die Methoden des Controllers sichtbar. Wird das Kennzeichen **Public** gesetzt, können auch andere Controller derselben Component darauf zugreifen.

Jeder Controller (mit Ausnahme des Interface- und Interface-View-Controllers) besitzt mindestens die zwei folgenden Attribute. Das Attribut **WD_THIS** stellt eine Selbstreferenz auf das Controller-Interface dar. Im Gegensatz zu diesem Attribut sollte die Standard-Selbstreferenz **me** in keinem Quellcode eines Controllers verwendet werden. Über **WD_THIS** kann auf alle Attribute und Methoden des Controllers zugegriffen werden. Das Attribut **WD_CONTEXT** stellt hingegen eine Referenz auf den Kontexteinstiegsknoten dar. Über diese Referenz kann zum gesamten Kontext navigiert werden.

Eigenschaften	Context	Attribute	Ereignisse	Methoden
Attribut	Public	RefTo	Bezugstyp	
WD_CONTEXT	<input type="checkbox"/>	<input checked="" type="checkbox"/>	IF_WD_CONTEXT_NODE	
WD_THIS	<input type="checkbox"/>	<input checked="" type="checkbox"/>	IF_COMPONENTCONTROLLER	

Abbildung 89: Standardattribute WD_CONTEXT und WD_THIS: SAP-System-Screenshot

Die Abbildung zeigt einen Ausschnitt eines Component Controllers, was am Bezugstyp des Attributs **WD_THIS** erkennbar ist. Je nach Controller-Typ steht dort ein entsprechender anderer Typ.

Bei Controllern, die auf ihrer **Eigenschaften**-Registerkarte den Component-Controller als verwendeten Controller ausweisen, etwa die View-Controller in den vergangenen Übungen, weisen ein weiteres Attribut **WD_COMP_CONTROLLER** auf. Dieses Attribut ist eine Referenz auf den Component-Controller. Diese kann für den Zugriff auf die Methoden und öffentlichen Attribute des Component-Controllers verwendet werden.

Attribut	RefTo	Bezugstyp	Beschreibung
WD_CONTEXT	<input checked="" type="checkbox"/>	IF_WD_CONTEXT_NODE	Referenz auf den lokalen Controller-Context
WD_THIS	<input checked="" type="checkbox"/>	IF_MAIN	Selbstreferenz auf das lokale Controller-Interface
WD_COMP_CONTROLLER	<input checked="" type="checkbox"/>	IG_COMPONENTCONTROLLER	Referenz auf den Component Controller

Abbildung 90: Das Attribut WD_COMP_CONTROLLER: SAP-System-Screenshot

Für andere verwendete Controller steht kein solches Attribut zur Verfügung. Um dennoch auf die Komponenten des verwendeten Controllers zugreifen zu können, muss die Referenz über einen Methodenaufruf geholt werden. Die Syntax dafür sieht wie folgt aus:

```
DATA: r_ctrl TYPE REF TO ig_<controllername>.  
r_ctrl = wd_this->get-<controllername>_ctr( ).
```

Hierbei steht <controllername> für den Namen des Controllers, dessen Referenz geholt werden soll.

Wie zuvor bei den Methoden ist es auch bei den Attributen möglich, eigene Definitionen vorzunehmen. Dies erfolgt auf der Registerkarte **Attribute**. Die Sichtbarkeit besteht zunächst nur innerhalb des Controllers, diese kann aber durch setzen des **Public**-Kennzeichens auf andere Controller derselben Component erweitert werden. Ein Bereitstellen der Attribute im Component-Interface ist hingegen nicht möglich. Der Zugriff auf die Attribute erfolgt über die Selbstreferenz **WD_THIS**. Für den Zugriff auf öffentliche Attribute anderer Controller wird eine Referenz benötigt, die wie oben beschrieben geholt werden kann.

7.8.3 Kontextzugriff zur Laufzeit

In den definierten Attributen können Datenobjekte innerhalb eines Controllers sichtbar machen. Eigenschaften von UI-Elementen können jedoch nicht an Attribute des Controllers gebunden werden, sondern benötigen für eine Datenbindung ein entsprechendes Attribut im Context des Controllers. Für den Zugriff auf den Context zur Laufzeit aus Methoden heraus gibt es spezielle Befehle, die Sie in diesem Abschnitt kennenlernen werden.

7.8.3.1 Zugriff auf Knoten und Elemente

Um einen Kontextknoten zu erreichen, wird die Methode **get_child_node** des übergeordneten Knotens verwendet (für den Einstieg also des Kontexteinstiegsknotens). Die folgende Abbildung zeigt dies an einem Beispiel:

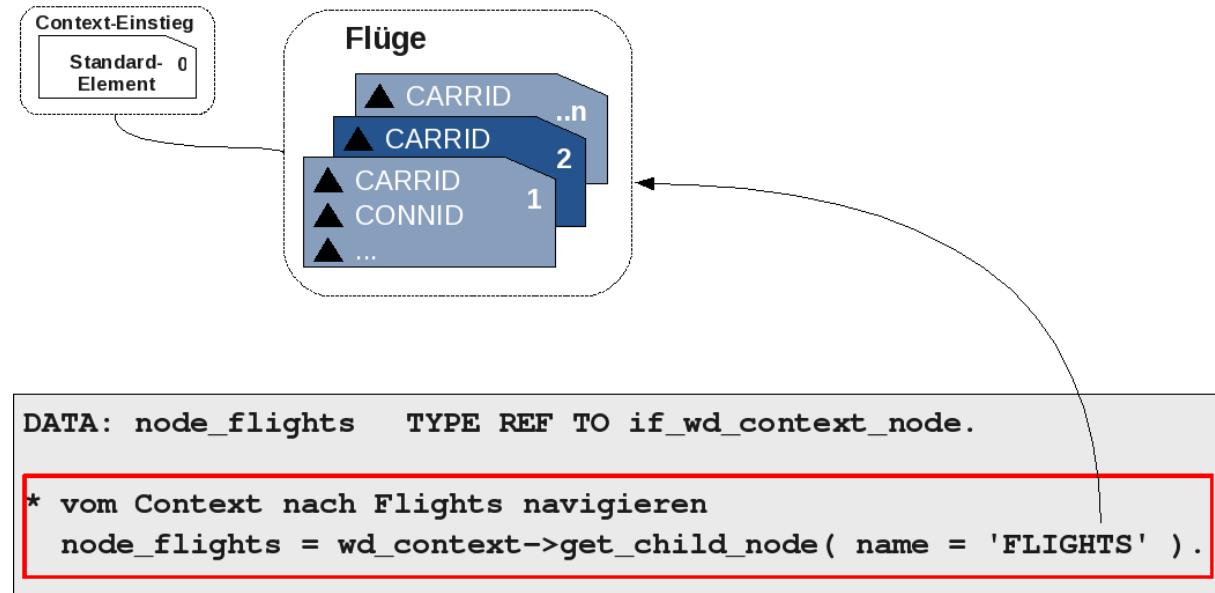


Abbildung 91: Zugriff auf einen Kontextknoten

Der Name des Kontextknotens ist hier in Großbuchstaben anzugeben. Alternativ steht dieser auch über eine Konstante zur Verfügung: Jeder Controller verfügt über ein Interface, dessen Name dem des Controllers mit einem vorangestellten **IF_** entspricht. In diesem Interface befindet sich für jeden Knoten des Controller-Contexts eine Konstante, deren Name dem Knotennamen mit Vorangestelltem **WDCTX_** entspricht. Ihr Wert ist der Knotenname in Großbuchstaben, also so wie er für den Aufruf der Methode **get_child_node** benötigt wird. Die Methode benötigt den Namen des Knotens, der zurückgeliefert werden soll. Optional kann auch ein Index übergeben werden, der angibt, zu welchem Element des übergeordneten Knotens die gewünschte Knoteninstanz gehört. Wird diese Angabe weggelassen, wird stattdessen die Lead-Selection verwendet. Im dargestellten Beispiel wird über den Kontexteinstiegsknoten zugegriffen, der ohnehin nur ein Element enthält.

Wenn eine Referenz auf den gewünschten Knoten geholt wurde, kann auf ein Element des Knotens zugegriffen werden. Hierfür bietet jeder Knoten die Methode **get_element**. Auch dieser Methode kann ein Index mitgegeben werden, um statt der Lead-Selection ein bestimmtes anderes Element zu erreichen. Die folgende Abbildung zeigt ein Beispiel zur Verwendung der Methode.

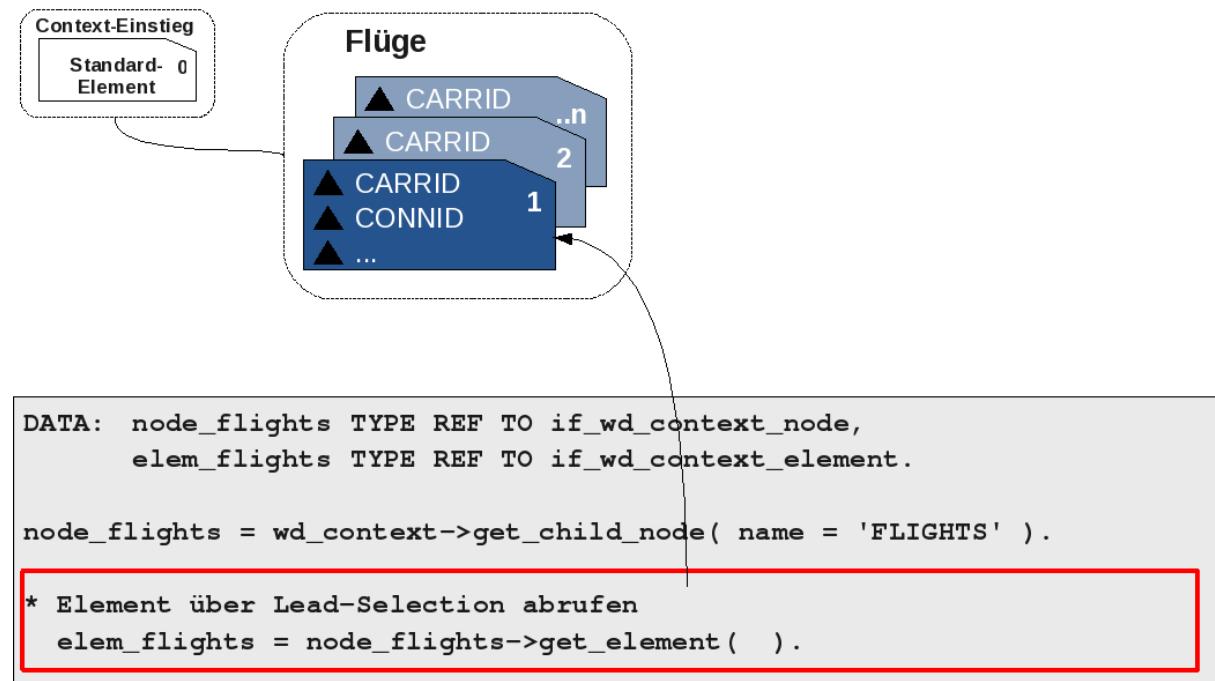


Abbildung 92: Zugriff auf ein Element eines Kontextknotens

Vor dem Zugriff auf das Element sollte noch geprüft werden, ob als Lead-Selection tatsächlich ein Element zurückgeliefert wurde (Prüfung auf `IS INITIAL`).

Der Zugriff auf ein Element anhand des Index hätte im obigen Beispiel so aussehen können:

```
elem_flights = node_flights->get_element( index = n ).
```

Wobei `n` dem gewünschten Index entspricht.

Auch ein Abruf der Elementanzahl ist möglich. Hierfür stellt der Knoten die Methode `get_element_count` zur Verfügung. Deren Aufruf sähe im betrachteten Beispiel so aus:

```
n = node_flights->get_element_count( ).
```

Hierdurch wird die Anzahl der Elemente in `n` gespeichert.

7.8.3.2 Zugriff auf Attribute

Mit Methoden des Web Dynpro-Frameworks kann sowohl auf einzelne Attribute eines Kontextelements als auch auf die Attribute aller Elemente eines Kontextknotens zugegriffen werden.

Für den Zugriff auf Attribute eines einzelnen Elements wird zunächst eine Referenz auf dieses Element benötigt. Das Element stellt eine Methode `get_attribute` bereit, die als Importparameter `name` den Namen des Attributs erwartet, und dessen Wert im Parameter `value` exportiert. Die folgende Abbildung zeigt den Aufruf an einem Beispiel:

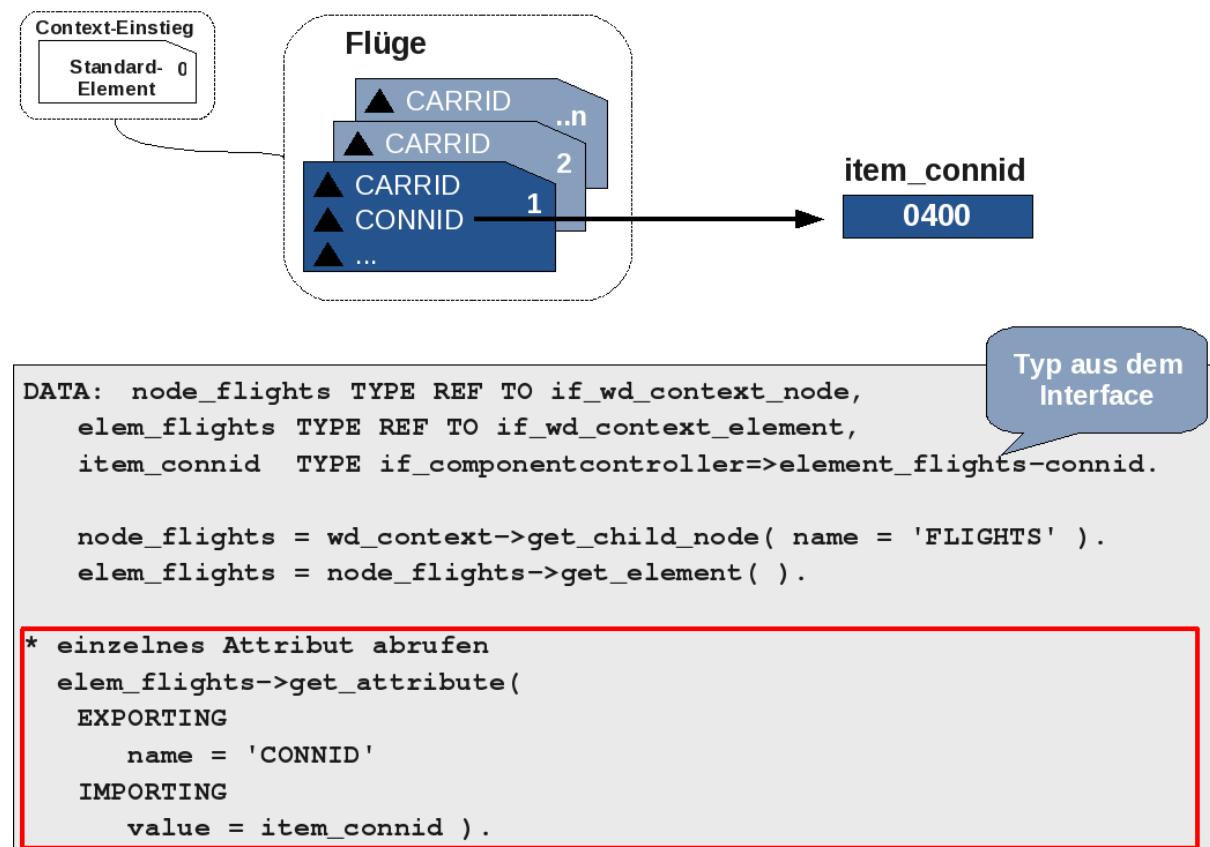


Abbildung 93: Zugriff auf ein einzelnes Attribut

Beachten Sie hier die Typisierung von `item_connid`: Für jeden Knoten im Controller-Context wird automatisch im Interface `if_<controllername>` ein Strukturtyp `element_<knotenname>` angelegt (im obigen Quelltext: `element_flights`). Die Elemente des Strukturtyps entsprechen den Attributen des Knotens. So kann die Variable typisiert werden, an die das Ergebnis des Methodenaufrufs übergeben wird (siehe letzte Zeile in der Abbildung).

Um alle statisch definierten Attribute eines Elements zu erhalten, kann die Methode `get_static_attributes` des Kontextelements verwendet werden. Die Verwendung unterscheidet sich nur unwesentlich vom soeben betrachteten Abruf eines einzelnen Attributs:

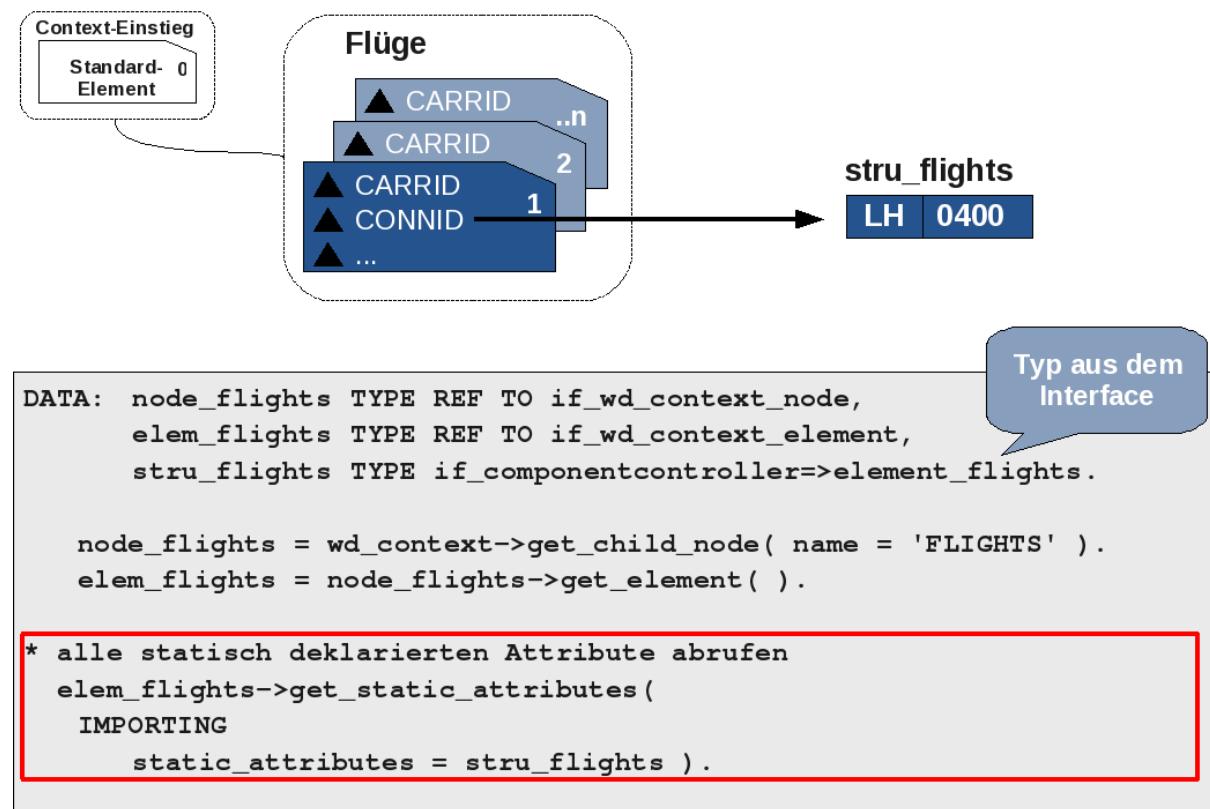


Abbildung 94: Zugriff auf alle statisch definierten Attribute eines Elements

Die Formulierung „statisch definiert“ bezieht sich in diesem Abschnitt darauf, dass die Attribute zur Designzeit angelegt wurden.

Über den Strukturtyp hinaus wird für jeden Knoten im Controller-Context ein Standard-Tabellen-Typ **elements_<knotenname>** im selben Interface erzeugt, dessen Zeilentyp der soeben beschriebene Strukturtyp ist. Mit diesem Typen kann eine interne Tabelle typisiert werden, die Attribute mehrerer Elemente eines Knotens aufnehmen kann. Um diese Elemente zu erhalten, stellt der Knoten die Methode **get_static_attributes_table** bereit, die einen Exportparameter **table** besitzt, der die entsprechenden Ergebnisse enthält. Die folgende Abbildung zeigt den Aufruf:

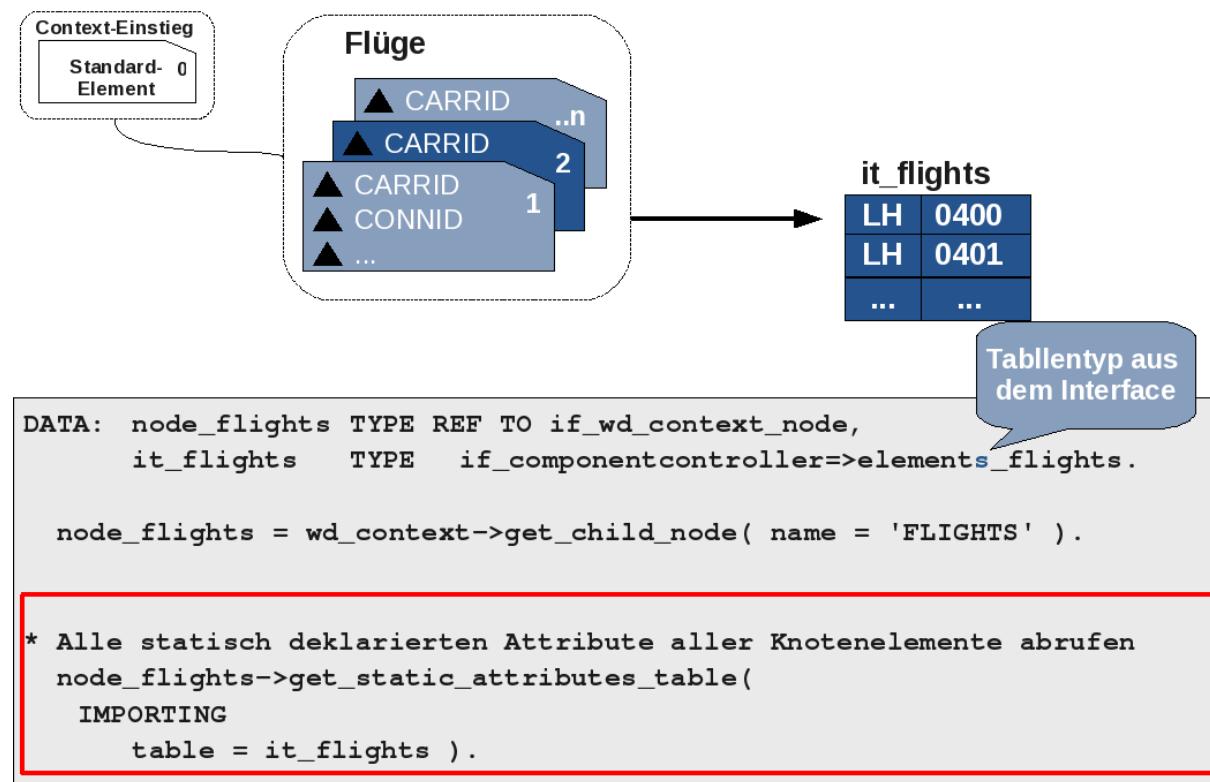
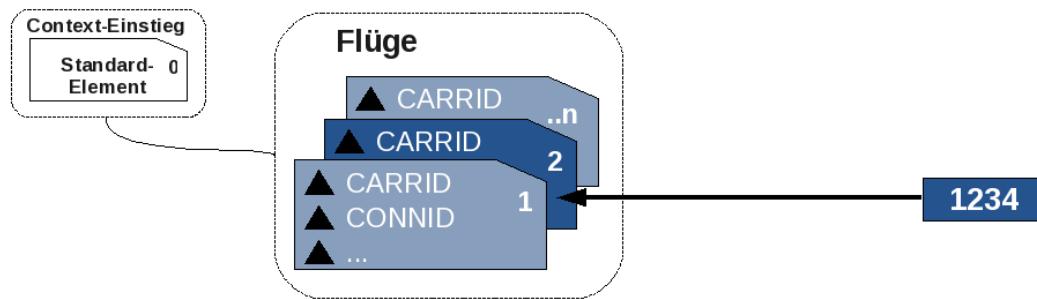


Abbildung 95: Zugriff auf die statisch definierten Attribute aller Elemente eines Knotens

Analog zu diesen get-Methoden, die die Werte von Attributen auslesen, gibt es auch entsprechende set-Methoden, mit denen die Attribute geschrieben werden können. Um ein einzelnes Attribut zu schreiben, wird die Methode **set_attribute** des Elements verwendet. Sie erwartet als Importparameter **name** den Namen des Attributs und als Importparameter **value** den zu setzenden Wert. Die folgende Abbildung zeigt ein Beispiel für die Verwendung.



```

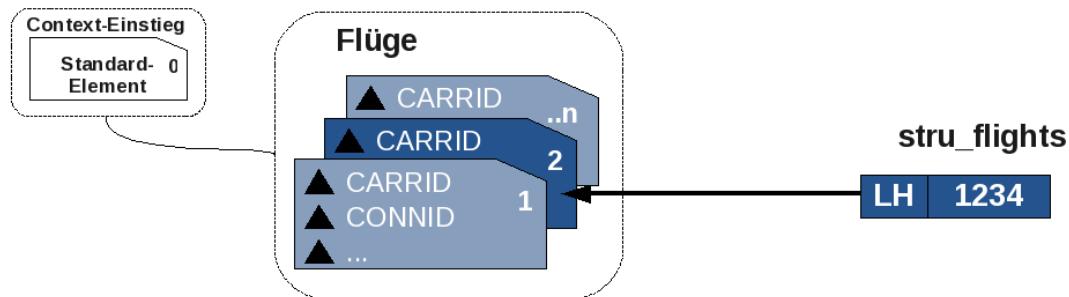
DATA: node_flights TYPE REF TO if_wd_context_node,
      elem_flights TYPE REF TO if_wd_context_element,
      item_connid  TYPE    if_componentcontroller=>element_flights->connid.

      node_flights = wd_context->get_child_node( name = 'FLIGHTS' ) .
      elem_flights = node_flights->get_element( ).

* einzelnes Attribut setzen
      elem_flights->set_attribute(
      EXPORTING
          name = 'CONNID'
          value = '1234' ) .
  
```

Abbildung 96: Setzen eines einzelnen Attributs

Zum Setzen aller statisch definierten Attribute eines Elements bietet dieses die Methode **set_static_attributes**. Diese erwartet als Importparameter **static_attributes** eine Struktur, die über die zuvor bereits beim Lesen der Attribute verwendete Weise typisiert wird:



```

DATA: node_flights TYPE REF TO if_wd_context_node,
      elem_flights TYPE REF TO if_wd_context_element,
      stru_flights TYPE    if_componentcontroller=>element_flights.

      node_flights = wd_context->get_child_node( name = 'FLIGHTS' ) .
      elem_flights = node_flights->get_element( ).

      stru_flights-carrid = 'LH'.
      stru_flights-connid = '1234'.

* statisch deklarierte Attribute setzen
      elem_flights->set_static_attributes(
      EXPORTING
          static_attributes = stru_flights ) .
  
```

Abbildung 97: Setzen aller statisch definierter Attribute

Die folgende Tabelle fasst die vorgestellten Möglichkeiten zum Lesen und Setzen von Attributwerten zusammen.

Tabelle 2: Lesen und Setzen von Attributwerten

Zugriff	Zu verwendender Code
Wert des Attributs <attribute> lesen	<pre>DATA value TYPE if_<controllername>=>wdctx_element_<node>- <attribute>. r_element->get_attribute(EXPORTING name = '<attribute>' IMPORTING value = value).</pre>
Werte aller statischen Attribute lesen	<pre>DATA s_struc TYPE if_<controllername>=>wdctx_element_<node>. r_element->get_static_attributes(IMPORTING Static_attributes = s_struc).</pre>
Werte aller statischen Attribute aller Elemente des Knotens lesen	<pre>DATA t_tab TYPE if_<controllername>=>wdctx_elements_<node>. r_node->get_static_attributes_table(IMPORTING table = t_tab).</pre>
Wert eines Attributs <attribute> auf den Wert von <value> setzen	<pre>r_element->set_attribute(EXPORTING name = '<attribute>' value = <value>).</pre>
Alle statischen Attribute eines Elements setzen	<pre>DATA s_struc TYPE if_<controllername>=>wdctx_element_<node>. s_struc-<attr1> = r_element->set_static_attributes(EXPORTING static_attributes = s_struc).</pre>

7.8.4 Einfügen und Entfernen von Kontextelementen

Über den Programmcode können nicht nur vorhandene Elemente gelesen und bearbeitet werden. Es ist auch möglich, neue Elemente einzufügen und bestehende Elemente zu entfernen. Dies ist wichtig, um aus dem Programmcode heraus einen Kontextknoten mit Daten versorgen zu können.

Zum Einfügen eines Elements wird dieses zunächst erzeugt und die Attribute können gesetzt werden. Erst danach wird das neue Element zum Kontextknoten hinzugefügt. Dieses Vorgehen ist vergleichbar mit dem Einfügen von Zeilen in interne Tabellen, die auch zunächst als Arbeitsbereich angelegt und mit Werten versorgt werden, um dann in die Tabelle eingefügt zu werden.

Um ein Element zu erhalten, wird die Methode **create_element** des Kontextknotens aufgerufen. Hierfür wird zunächst eine Referenz auf ebendiesen benötigt, die über die bekannte Methode **get_child_node** geholt werden kann.

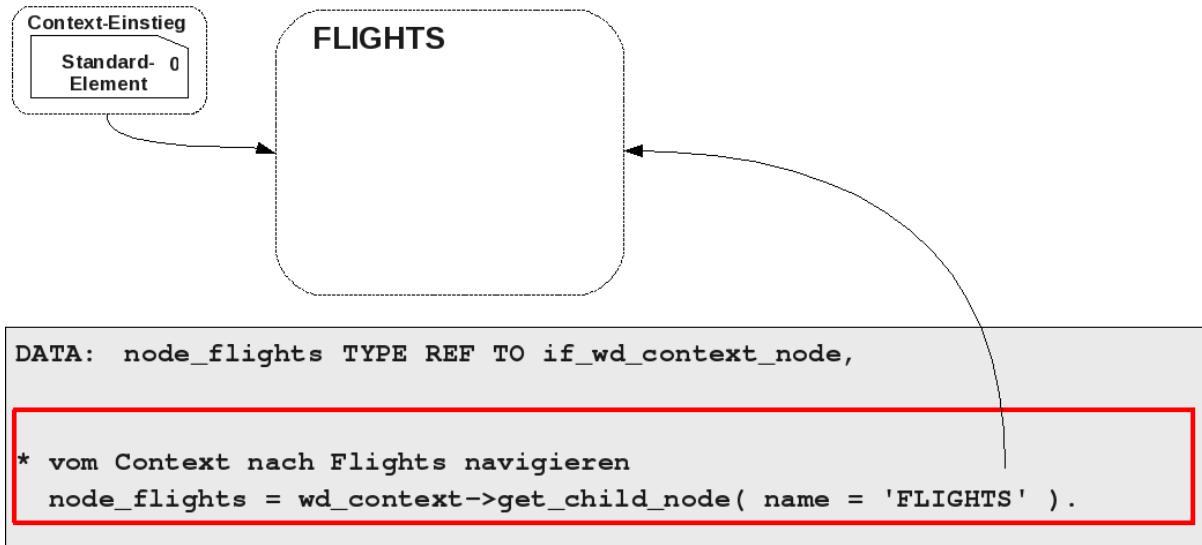


Abbildung 98: Holen der Knoten-Referenz

Mit dieser Referenz kann nun die Methode zur Elementerzeugung aufgerufen werden:

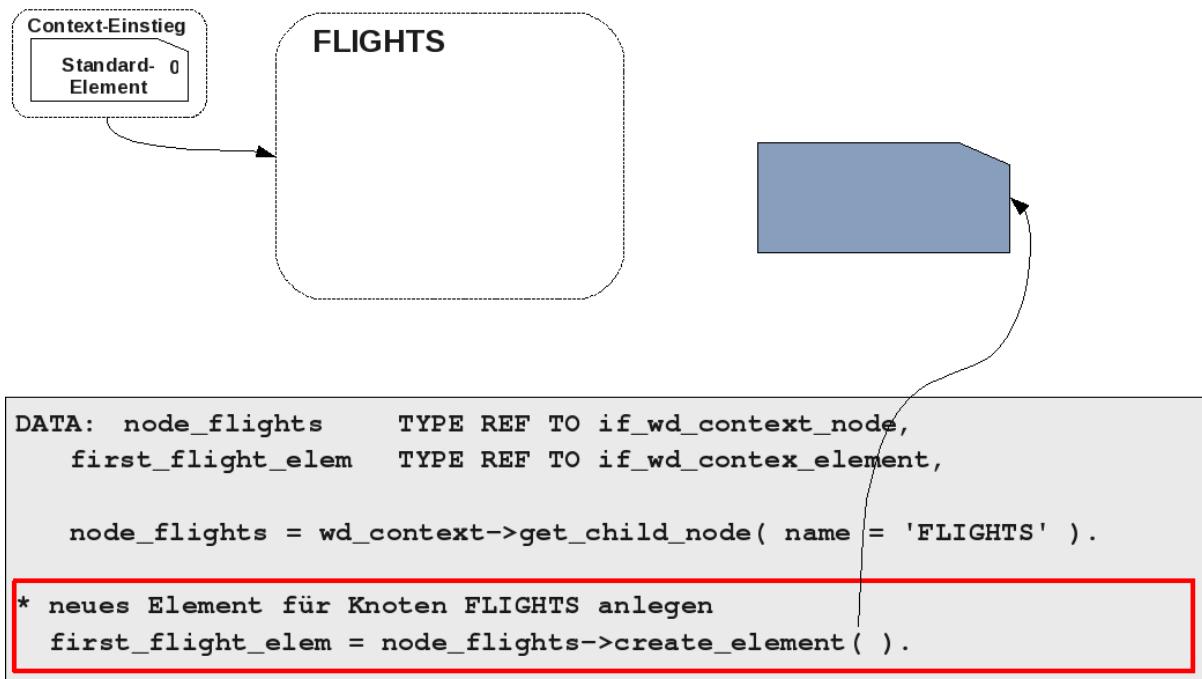


Abbildung 99: Erzeugen eines Elements

Das Element ist hierdurch zwar erzeugt, aber noch nicht Teil des Kontextknotens. Zunächst werden seine Attributwerte gesetzt:

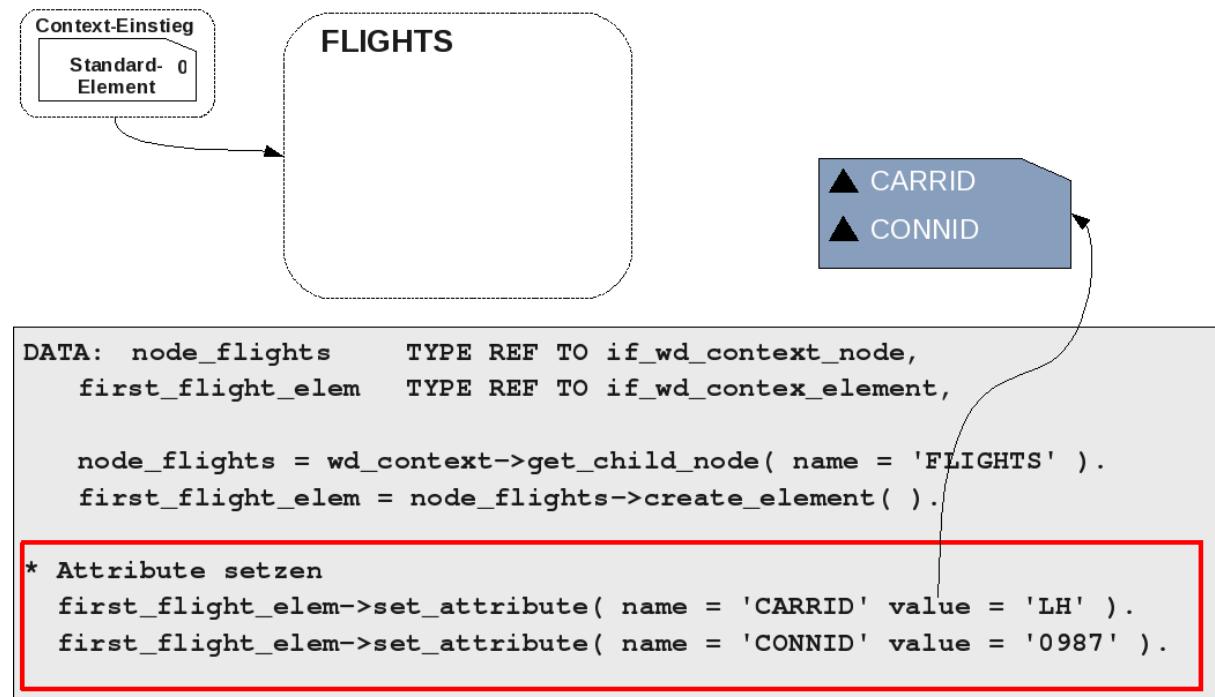


Abbildung 100: Setzen der Attributwerte des Elements

Das „fertige“ Element kann nun zum Knoten hinzugefügt werden. Hierfür wird die Methode **bind_element** des betreffenden Knotens aufgerufen und das neue Element als Parameter **new_item** übergeben. Der zweite Parameter **set_initial_elements** der Methode dient dazu festzulegen, ob durch den Aufruf ein weiteres Element zu den ggf. bereits vorhandenen Elementen hinzugefügt werden soll (Übergabe des Werts **abap_false**), oder ob alle bestehenden Elemente entfernt werden sollen, so dass nur noch das neue Element Teil des Knotens ist (Übergabe des Werts **abap_true**).

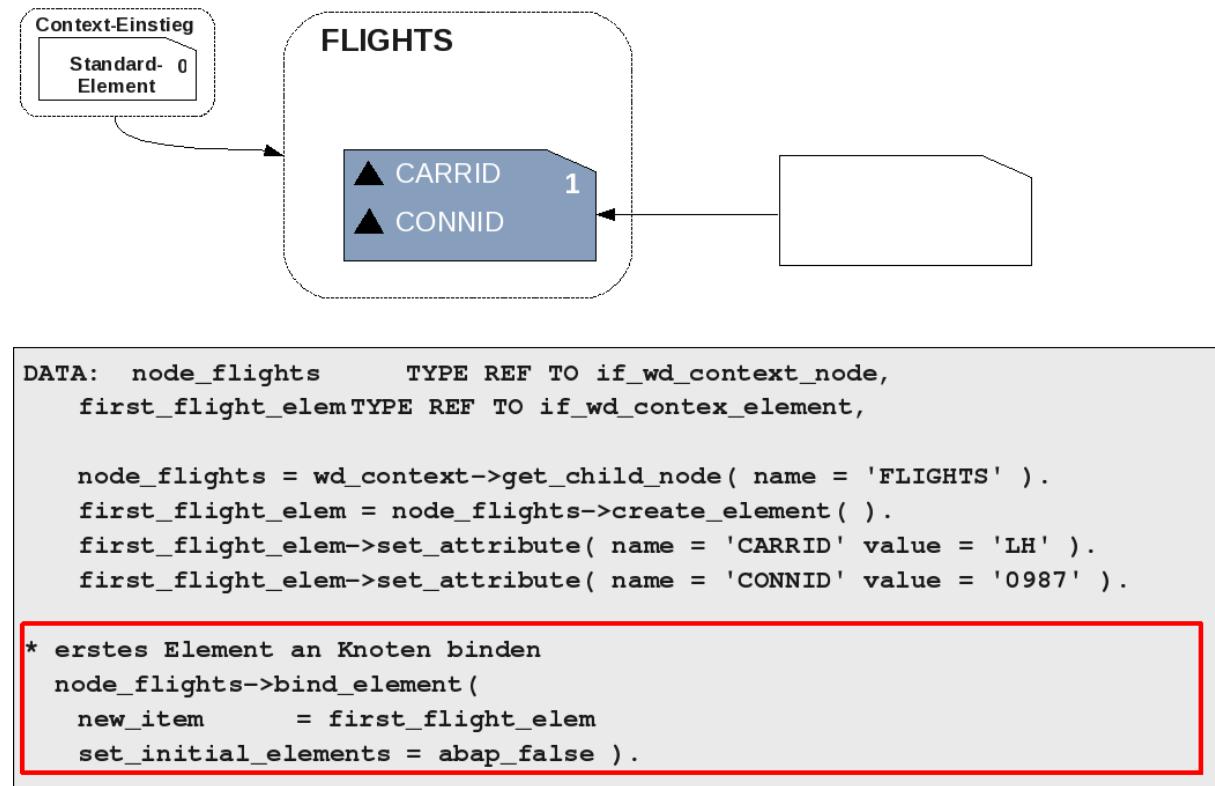


Abbildung 101: Einfügen des Elements

In ABAP-Programmen liegen die Daten häufig in Form von Strukturen vor. Soll ein solcher Datensatz in den Context eingefügt werden, wäre es umständlich, die Strukturkomponenten extra in das Element zu kopieren und dieses dann einzufügen. Es gibt für diese Aufgabe eine Vereinfachung in Form der Methode **bind_structure**. Diese übernimmt in einem Schritt das Anlegen des Elements, das Setzen der Attributwerte des Elements sowie das Einfügen des Elements in den Kontextknoten. Die Methode besitzt den Parameter **new_item** zur Übergabe der Struktur sowie den Parameter **set_initial_elements**, der dem gleichnamigen Parameter der **bind_element**-Methode entspricht. Die folgende Abbildung zeigt ein Beispiel für die Verwendung.

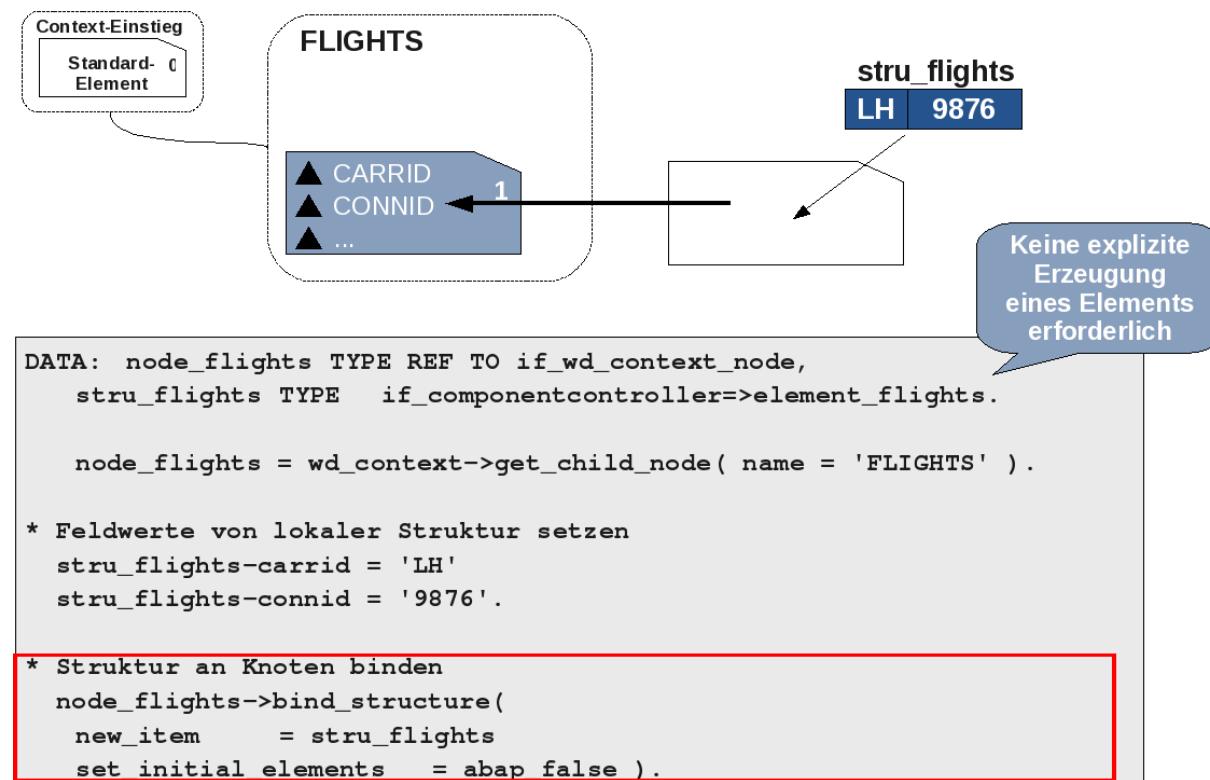


Abbildung 102: Direktes Einfügen von Daten aus einer Struktur

Der naheliegende nächste Schritt ist es, auch interne Tabellen an Kontextknoten zu binden, um so nicht nur einen Datensatz einzufügen zu können, sondern gleich eine ganze Reihe von Datensätzen als einzelne Elemente zum Knoten hinzuzufügen. Eine Methode, die diese Aufgabe erfüllt, steht unter dem Namen **bind_table** zur Verfügung. Deren Schnittstelle ähnelt der Methode **bind_structure**, wobei statt einer Struktur **new_item** hier eine interne Tabelle **new_items** übergeben wird. Die folgende Abbildung zeigt die Verwendung anhand eines Beispiels:

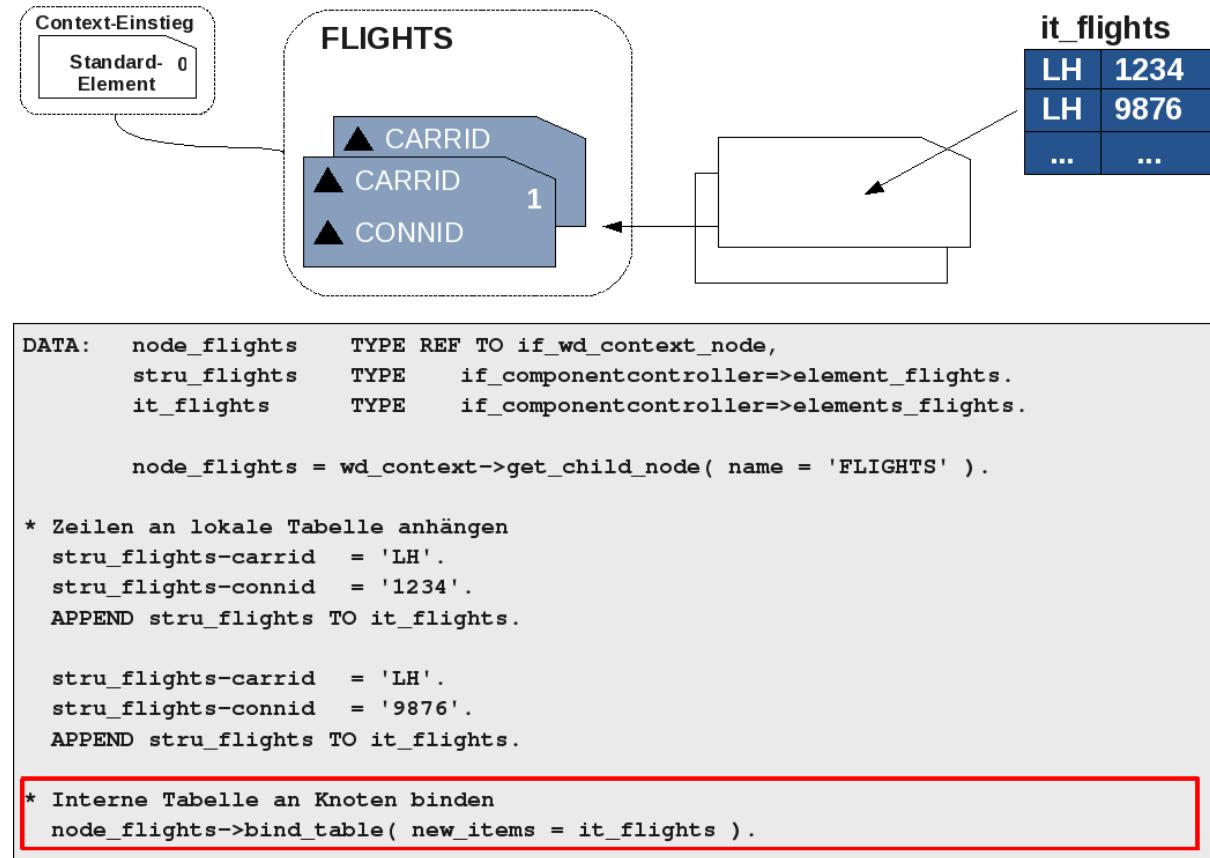


Abbildung 103: Einfügen mehrerer Elemente über eine Interne Tabelle

Zum Löschen eines Elements wird die Methode **remove_element** verwendet. Diese Methode erwartet als Parameter **element** eine Referenz auf das zu löschen Element. Die folgende Abbildung zeigt die Verwendung der Methode.

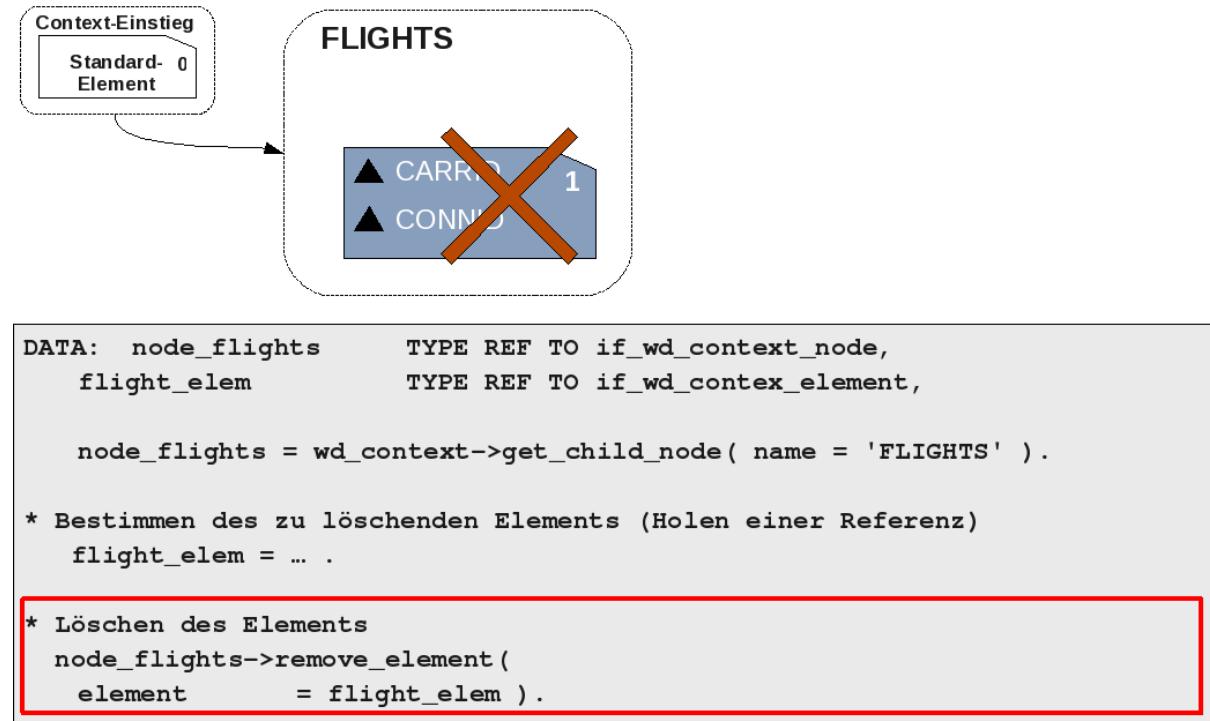


Abbildung 104: Löschen eines Elements

Die folgende Tabelle fasst die Möglichkeiten zum Hinzufügen bzw. Entfernen zusammen, die Sie kennengelernt haben:

Tabelle 3: Hinzufügen und Entfernen von Elementen von Kontextknoten

Zugriff	Zu verwendender Code
Anlegen eines Elements	<code>r_element = r_node->create_element() .</code>
Hinzufügen eines Elements zur Collection eines Knotens	<code>r_node->bind_element(new_item = r_element set_initial_elements = abap_false) .</code>
Binden einer Struktur <code>s_struc</code> an die Collection	<code>DATA s_struc TYPE if <controllername>=>wdctx_element_<node>. ... r_node->bind_structure(new_item = s_struc set_initial_elements = abap_false) .</code>
Binden einer internen Tabelle <code>t_tab</code> an die Collection	<code>DATA t_tab TYPE if <controllername>=>wdctx_elements_<node>. ... r_node->bind_table(new_items = t_tab set_initial_elements = abap_false) .</code>
Entfernen eines Elements aus der Collection	<code>r_node->remove_element(element = r_element) .</code>

7.8.5 Praxis: Übung zum Contextzugriff über eine Hook-Methode

In dieser Übung werden Sie den Context-Zugriff zur Laufzeit anwenden. Erstellen Sie für diese Übung eine Kopie ihrer Component **ZZ_####_WD_FLIGHTS** mit dem Namen **ZZ_####_WD_FLIGHTS2**, indem Sie im Navigationsbaum des Object Navigators mit der rechten Maustaste auf die Component klicken und aus dem Kontextmenü **Kopieren...** wählen. Geben Sie den Namen der Kopie sowie wie gewohnt Paket und Transportauftrag an. Legen Sie auch für die kopierte Component eine Web Dynpro-Anwendung an. Aktivieren Sie die kopierte Component und Vergewissern Sie sich, dass sie funktioniert.

Die Anwendung soll derart angepasst werden, dass das Feld für die Abflugstadt als Vorbelegung die Stadt enthält, von der die meisten Flugverbindungen starten. Öffnen Sie hierzu die erste View und wechseln Sie zur Registerkarte **Methoden**. Die benötigte Funktionalität wird in der Hook-Methode **WDDOINIT** hinterlegt. Durch Doppelklick auf den Methodennamen erreichen Sie den Quellcode (der momentan noch leer ist).

Zum Setzen des Feldes muss das entsprechende Kontextattribut gesetzt werden, an das das Feld per Datenbindung gebunden ist. Um Schreibarbeit zu sparen, können Sie den Wizard verwenden, und dort bei **Operation auf dem Context** (im **Context**-Tab des Wizards) **Setzen** wählen und als **Knoten/Attribut** das Attribut **CITY** des Knotens **DESTINATION_FROM** auswählen.

```

1  method WDDOINIT .
2   DATA lo_nd_destination_from TYPE REF TO if_wd_context_node.
3
4   DATA lo_el_destination_from TYPE REF TO if_wd_context_element.
5   DATA ls_destination_from TYPE wd_this->Element_destination_from.
6   DATA lv_city TYPE wd_this->Element_destination_from-city.
7
8   * navigate from <CONTEXT> to <DESTINATION_FROM> via lead selection
9   lo_nd_destination_from = wd_context->path_get_node( path = 'BAPI_FLIGHT_GETLIST IMPORTING DESTINATION_FROM' ).
10
11  * @TODO handle non existant child
12  * IF lo_nd_destination_from IS INITIAL.
13  * ENDIF.
14
15  * get element via lead selection
16  lo_el_destination_from = lo_nd_destination_from->get_element( ).  

17
18  * @TODO handle not set lead selection
19  IF lo_el_destination_from IS INITIAL.
20  ENDIF.  

21
22  * @TODO fill attribute
23  * lv_city = 1.  

24
25  * set single attribute
26  lo_el_destination_from->set_attribute(
27   name = 'CITY'
28   value = lv_city ).  

29 endmethod.

```

Abbildung 105: Generierter Code zum Elementzugriff: SAP-System-Screenshot

Beachten Sie: In älteren Releases kann der Wizard nur Code zum Lesen generieren, durch Ersetzen des Methodenauftrags von **get_attributes** durch einen schreibenden Zugriff können Sie aber auch dort die gewünschte Funktionalität erreichen.

Sie benötigen nun noch die Stadt, die in das Feld geschrieben werden soll. Nutzen Sie ihre ABAP-Kenntnisse, um Städte und Anzahlen der Flugverbindungen aus der Datenbank zu lesen, bestimmen Sie das Maximum und schreiben Sie diese Stadt in das Kontextattribut. Speichern und aktivieren Sie wie gewohnt und testen Sie dann Ihre Anwendung. Das Feld zur Eingabe der Abflugstadt sollte nun mit der Stadt mit den meisten ausgehenden Verbindungen vorbelegt sein.

7.8.6 Praxis: Übung zu Bindung von Internen Tabellen, Lead-Selection und Supply-Funktionen

In dieser Übung werden Sie die Bindung von internen Tabellen an Kontextknoten anwenden. Sie werden eine Supply-Funktion implementieren, die den Inhalt einer Tabelle in Abhängigkeit von der Lead-Selection einer anderen Tabelle setzt.

Erstellen Sie für diese Übung eine neue Component **ZZ_####_WD_BOOK** mit einem Window **MAIN_WINDOW**, einer View **INPUT_VIEW** und einer View **OUTPUT_VIEW**. Auf der ersten View sollen eine Fluggesellschaft und eine Flugverbindung ausgewählt werden können, auf der zweiten View soll neben den Flügen auch eine Tabelle mit den zugehörigen Buchungen erscheinen.

Hierfür benötigen Sie zunächst die entsprechenden Kontextknoten. Öffnen Sie den Context des **Component-Controllers** und legen Sie einen Knoten **CONNECTION** an. Verwenden Sie die Kardinalität 1..1 (da dieser für die Eingabefelder der ersten View verwendet werden soll), beziehen Sie sich auf die Struktur **SPFLI** aus dem Dictionary und übernehmen Sie die Felder **CARRID** und **CONNID**. Erstellen Sie einen weiteren Knoten unterhalb des Kontexteinstiegsknotens mit dem Namen **FLIGHTDATA**. Dieser soll die Flüge aufnehmen und basiert daher auf **SFLIGHT**. Die Kardinalität beträgt 0..n (da es keinen bis mehrere Flüge zur Verbindung geben kann). Fügen Sie auch hier Attribute aus der Struktur hinzu (mindestens **CARRID**, **CONNID** und **FLDATE**, weitere nach Ihrer Wahl). Erstellen Sie danach für die Buchungen einen untergeordneten Knoten **BOOKINGDATA** unterhalb von **FLIGHTDATA**, wählen Sie die Kardinalität 0..n (keine bis mehrere Buchungen zum Flug)

und übernehmen Sie aus der Struktur SBOOK mindestens die Felder **CARRID**, **CONNID**, **FLDATE**, **BOOKID** und **CUSTOMID**. Sichern Sie den Component Controller und öffnen Sie den Context der ersten View. Sorgen Sie dort dafür, dass der Kontextknoten **CONNECTION** auch in diesem Context zur Verfügung steht und ein Mapping zum Knoten des Component-Controllers besteht.

Wechseln Sie dann zur Registerkarte **Layout**. Erstellen Sie über den Wizard ein Eingabeformular für den soeben angelegten Kontextknoten. Vergeben Sie für die Überschrift des Formulars einen Text und stellen Sie sicher dass das entsprechende TextView-Element als **Layoutdaten** die Einstellung **MatrixHeadData** aufweist.

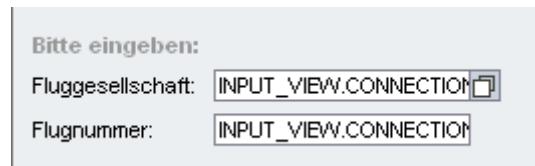


Abbildung 106: Das definierte Eingabeformular: SAP-System-Screenshot

Öffnen Sie nun die zweite View. Wechseln Sie dort zum Context. Sorgen Sie hier wie gewohnt durch „Drag'n'Drop“ dafür, dass alle Kontextknoten übernommen und gemappt werden (siehe nächste Abbildung):

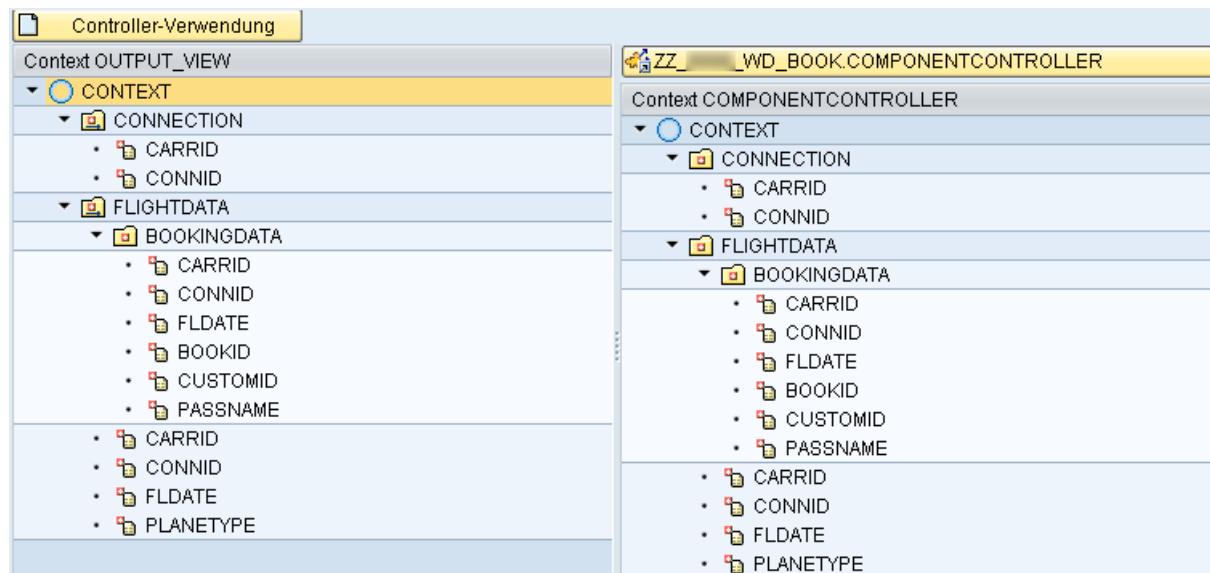


Abbildung 107: Mapping des Contexts: SAP-System-Screenshot

Wechseln Sie nun zur Registerkarte **Layout**. Erstellen Sie dort mit dem Wizard ein Formular analog zur ersten View, bei dem allerdings die Eingabefelder deaktiviert sind. Nutzen Sie ebenfalls den Wizard, um Tabellen für die Flüge sowie für die Buchungen zu erzeugen.

Erstellen Sie als nächstes eine Navigation zwischen beiden Views, indem Sie wie gewohnt Buttons, Plugs und Navigationslinks anlegen. Denken Sie daran beide Views in das Window einzubetten. Aktivieren Sie die Component und ihre Unterelemente, erstellen Sie eine Anwendung und testen Sie, ob die Navigation funktioniert.

Nachdem nun die grundlegende Oberfläche und Navigation existiert, muss für die Versorgung mit entsprechenden Daten gesorgt werden. Erstellen Sie zunächst im Component Controller eine neue Methode **read_flights** und öffnen Sie deren Quelltext. Damit die Flüge gelesen werden können, muss zunächst die Verbindung ermittelt werden. Lesen Sie diese aus dem

Contextknoten **CONNECTION** aus, indem Sie den Wizard zur Generierung der Anweisungen benutzen.

Definieren Sie dann eine interne Tabelle für die Flüge. Nutzen Sie zur Typisierung den Typ `if_componentcontroller=>elements_flightdata`. So können Sie bequem die Tabelle passend zum Context definieren.

Lesen Sie dann die Daten aus der Tabelle **SFLIGHT** in die interne Tabelle. Nutzen Sie hierfür die `INTO CORRESPONDING FIELDS OF TABLE`-Syntax. Für den Where-Teil stehen Ihnen `CARRID` und `CONNID` aus der Struktur `ls_connection` zur Verfügung, die der Wizard definiert und mit den Daten aus dem Kontextknoten **CONNECTION** gefüllt hat. Binden Sie die interne Tabelle anschließend an den Kontextknoten **FLIGHTDATA**. Nutzen Sie hierfür die **bind_table**-Methode. Da es sich um eine Methode des Knotens handelt, müssen Sie zunächst eine Referenz auf diesen Knoten holen. Verwenden Sie dafür die Methode `wd_context->get_child_node`:

```
DATA node_flightdata TYPE REF TO if_wd_context_node.  
node_flightdata = wd_context->get_child_node( name = 'FLIGHTDATA' ).
```

Abbildung 108: Holen einer Referenz des Knotens: SAP-System-Screenshot

Die Syntax für das Binden der Tabelle an den Kontextknoten könnte wie folgt aussehen (passen Sie den Namen der internen Tabelle an den von Ihnen gewählten Namen an):

```
CALL METHOD node_flightdata->bind_table  
  EXPORTING  
    new_items = itab.
```

Abbildung 109: Binden der Tabelle: SAP-System-Screenshot

Speichern Sie die Methode und öffnen Sie die zweite View. Öffnen Sie dort den Quelltext der Methode **handlefrom_input_view** (oder den entsprechenden handle...-Namen, wenn sie die Benennung anders vorgenommen haben). Dies ist die Ereignisbehandlermethode, die bei der Navigation zur zweiten View ausgeführt wird. An dieser Stelle fügen Sie einen Aufruf Ihrer zuvor implementierten Methode im Component-Controller ein. Nutzen Sie dafür den Wizard mit der Auswahl **Methodenaufruf im verwendeten Controller** und der Auswahl des Component Controllers:

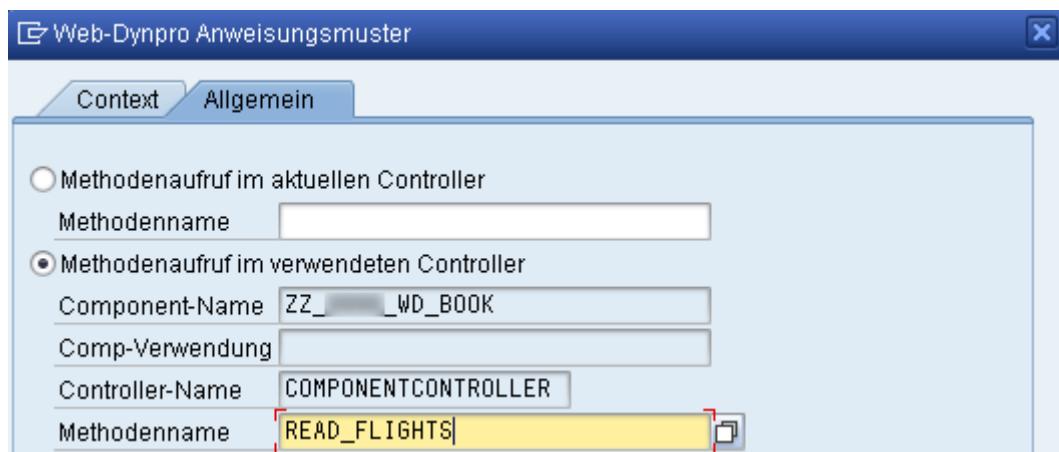


Abbildung 110: Wizard-Verwendung für den Methodenaufruf: SAP-System-Screenshot

Speichern Sie den Controller und aktivieren Sie wie gewohnt. Testen Sie Ihre Anwendung. Bei Angabe einer gültigen Flugverbindung (z. B. LH 401) sollten in der Tabelle der zweiten View nun die gewünschten Daten erscheinen:

Verbindung:				
Fluggesellschaft:	LH	Flugnummer:	0401	
LH	0401	05/25/2012	A319	
LH	0401	06/22/2012	A319	
LH	0401	07/20/2012	A319	
LH	0401	08/17/2012	A319	
LH	0401	09/14/2012	A319	

Abbildung 111: Ausgabe der Flüge: SAP-System-Screenshot

(Anmerkung: Die Darstellung kann von der Vorschau bezüglich der Fußzeile / des Scrollbalken abweichen. Hier gibt es Versionsabhängige Unterschiede)

Die Buchungstabelle ist noch leer. Diese soll nun durch eine Supply-Funktion gefüllt werden, die die Buchungen zur Lead-Selection der Flugtabelle bestimmt.

Öffnen Sie dazu den Context des Component-Controllers. Wählen Sie dort den untergeordneten Knoten **BOOKINGDATA** aus. In den Eigenschaften finden Sie den Eintrag **Supply-Funktion**. Geben Sie dort **READ_BOOKINGS** ein und klicken Sie doppelt auf diesen Text (Vorwärtsnavigation). Dadurch wird die Methode angelegt und Sie gelangen zum Quelltext. Dieser enthält bereits einige Zeilen. Entfernen Sie die Kommentarsterne vor den Code-Zeilen unterhalb von `data declaration`:

```

    ** data declaration
    DATA lt_bookingdata TYPE wd_this->Elements_bookingdata.
    DATA ls_bookingdata LIKE LINE OF lt_bookingdata.

    ** @TODO compute values
    ** e.g. call a data providing FuBa
    *
    ** bind all the elements
    node->bind_table(
        new_items          = lt_bookingdata
        set_initial_elements = abap_true ).
```

Abbildung 112: Vorgegebener Code nach entfernen der Kommentarsterne: SAP-System-Screenshot

Ergänzen Sie die Datendeklaration um eine Struktur vom Typ der Flugtabelle des Kontexts. (Typisieren Sie mit `TYPE if_componentcontroller=>element_flightdata`). Um die Struktur mit den Daten der Lead-Selection zu füllen, benötigen Sie die Methode `get_static_attributes` des Kontextelements. Dieses können Sie über den Parameter `parent_element` erreichen, der vom System zur Verfügung gestellt wird:

```

parent_element->get_static_attributes(
    IMPORTING
        static_attributes = ls_flight ).
```

Abbildung 113: Zugriff auf das Parent-Element: SAP-System-Screenshot

Lesen Sie nun aus der Datenbanktabelle SBOOK die relevanten Buchungen in die (vom Wizard bereits definierte) interne Tabelle lt_bookingdata ein. Verwenden Sie im WHERE-Teil die Spalten CARRID, CONNID und FLDAT.

Den Aufruf zur Bindung der internen Tabelle an den Kontextknoten **BOOKINGDATA** müssen Sie nicht mehr programmieren: Diese Arbeit hat Ihnen der Wizard bereits abgenommen. Speichern, prüfen und aktivieren Sie daher nun und testen Sie die Anwendung.

Durch das Selektieren einer Zeile werden nun in der unteren Tabelle die zugehörigen Buchungen angezeigt. Verschönern Sie die Ausgabe, indem Sie in den Tabellenelementen der zweiten Views noch Überschriften einfügen. Klicken Sie dazu mit der rechten Maustaste auf das Tabellenelement in der Elementhierarchie, wählen Sie **Header Einfügen** (falls dies ausgegraut ist, ist das Header-Element bereits vorhanden und muss nur ausgewählt werden) und pflegen Sie die Eigenschaft **text** des Headers. Um die Oberfläche nachträglich zu vereinfachen, können Sie bei Bedarf auch Spalten aus der Tabelle entfernen.

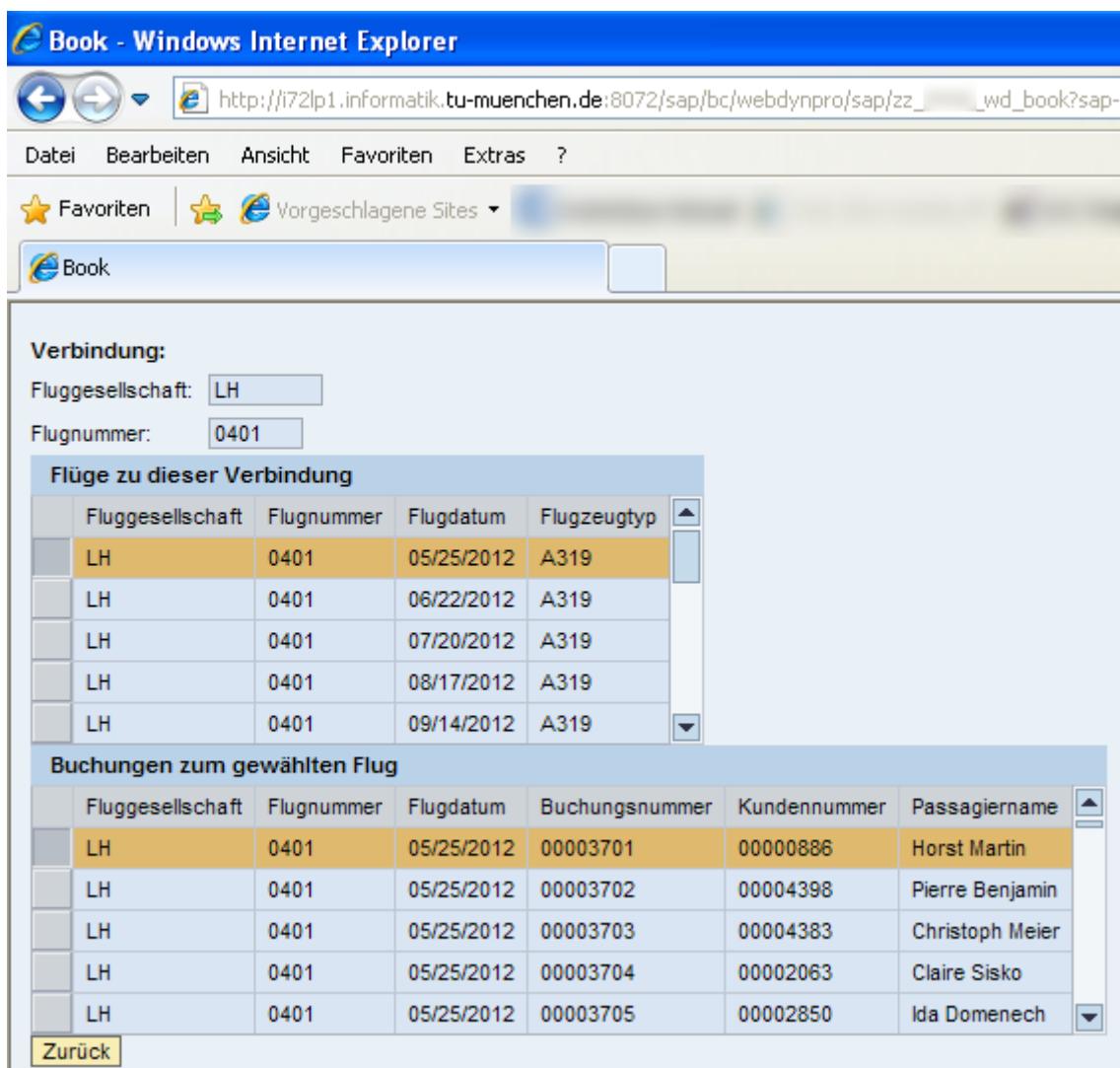


Abbildung 114: Oberfläche der Buchungsanwendung: SAP-System-Screenshot

7.9 Internationalisierung und Nachrichten in Web Dynpro-Anwendungen

Das SAP-System ist konsequent darauf ausgerichtet, mehrsprachige Anwendungen liefern zu können. So sind etwa Feldbezeichner im Dictionary in andere Sprachen übersetzbare, und das System liefert dem Anwender den passenden Bezeichner in seiner Sprache. Auch Web Dynpro-Anwendungen können mehrsprachig angeboten werden. Dafür müssen die Textliterale, die in der Anwendung verwendet werden, übersetzbare gemacht werden. Hierfür stehen drei Möglichkeiten zur Verfügung:

- Definition im ABAP Dictionary
- Definition im Online Text Repository (OTR)
- Definition in einer ABAP-Klasse

Die Übersetzung erstreckt sich natürlich auch auf Nachrichten. Als solche können Texte des OTR, der Datenbanktabelle T100 oder Textelemente aus ABAP-Klassen gesendet werden.

The screenshot displays two versions of a Web Dynpro application side-by-side, separated by a large double-headed arrow labeled "Übersetzung" (Translation).
Left Side (German Version):
- Title: Willkommen bei Web Dynpro!
- Section: Persönliche Daten
- Fields: Vorname (Johann), Nachname (Schmidt)
- Section: Reservierungsdaten
- Fields: Abholdatum (06.08.2004), Rückgabedatum (06.08.2004)
- Field: Fahrzeugtyp (Luxury)
- Section: Region
- Radio buttons: Asien (empty), Europa (selected), Südamerika (empty)
- Field: Abholort
- Buttons: Sichern, Zurücksetzen
Right Side (English Version):
- Title: Welcome to Web Dynpro!
- Section: Personal data
- Fields: Firstname (John), Lastname (Smith)
- Section: Reservation data
- Fields: Pickup Date (8/6/2004), Dropdown Date (8/6/2004)
- Field: Vehicle Type (Luxury)
- Section: Region
- Radio buttons: Asia (empty), Europe (empty), North America (selected), South America (empty), Oceania (empty)
- Fields: Pickup location, Dropdown location
- Buttons: Save, Reset

Abbildung 115: Übersetzung einer Web Dynpro-Oberfläche

Bei der Internationalisierung werden alle sprachenabhängigen Texte so angelegt, dass zur Laufzeit eine Version in der Anmeldesprache des Nutzers verfügbar ist. Die obige Abbildung zeigt ein Beispiel, wie dieselbe Anwendung für Benutzer mit unterschiedlichen Sprachen angezeigt werden kann. Sie sehen dort auch das benutzerspezifische Datumsformat. Auch die hier nicht sichtbaren Quick-Infos sind Beispiele für sprachabhängige Entitäten, und sogar Abbildungen können entsprechend angepasst werden.

Internationalisierung wird häufig als **I18N** abgekürzt. Dies steht für das englische Wort Internationalization, wobei nur der erste und letzte Buchstabe dargestellt wird und die anderen 18 Buchstaben als Zahl 18 wiedergegeben werden.

7.9.1 Möglichkeiten zu Internationalisierung von Texten

Um zu sehen, welche Texte bereits übersetzt werden, testen Sie die zuletzt angelegte Anwendung. Wählen Sie dabei beim Login als Sprache **Englisch** aus:

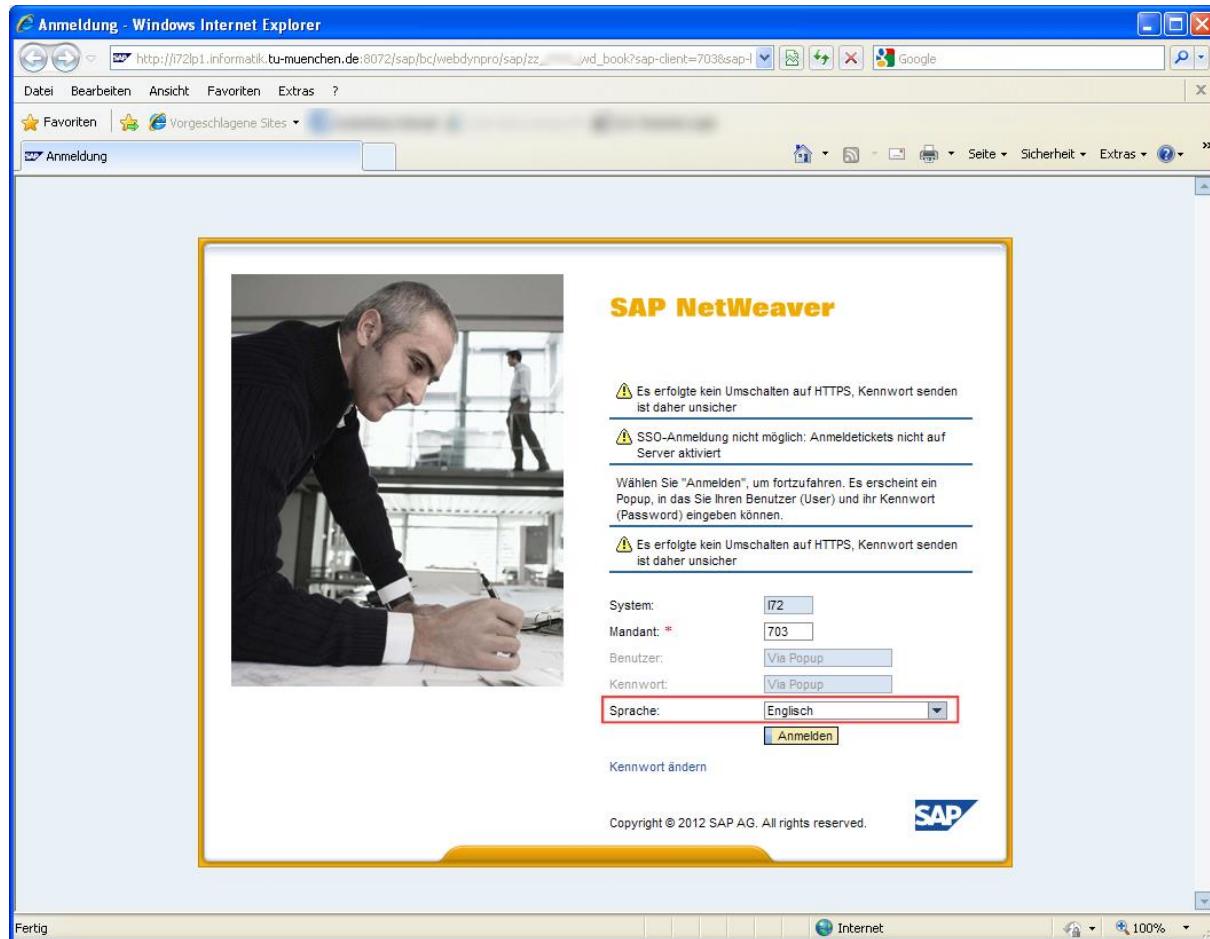


Abbildung 116: Geänderte Sprache beim Login: SAP-System-Screenshot

Sie sehen, dass Teile der Anwendung bereits übersetzt werden, etwa die Feldbeschreibungen und Spaltentitel auf der zweiten View:

Flüge zu dieser Verbindung				
Airline	Flight Number	Date	Plane Type	
LH	0401	05/05/2012	A319	
LH	0401	05/05/2012	A319	

Abbildung 117: Durcheinander der Sprachen: SAP-System-Screenshot

Schauen Sie sich die Eigenschaften der **Label**-Elemente für die Eingabefelder auf der ersten View an. Sie sehen dort, dass diese sich auf ein Eingabefeld beziehen und ihren Text aus dessen Dictionary-Element beziehen. Sie können diesen Bezug explizit konfigurieren. Klicken Sie dazu auf die Schaltfläche zur Datenbindung neben der text-Eigenschaft des Labels. Sie sehen dann im Fenster zur Konfiguration der Bindung im unteren Bereich den Dictionary-Bezug:

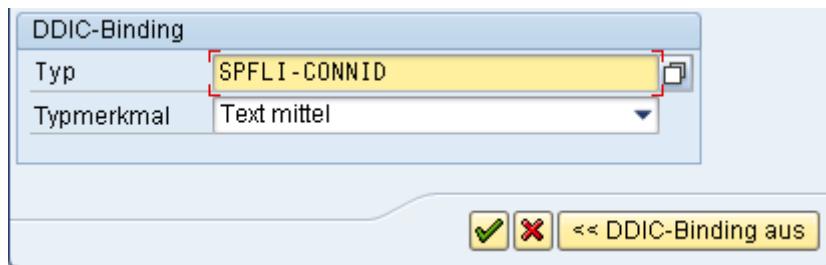


Abbildung 118: Texte aus dem Dictionary: SAP-System-Screenshot

Dort können auch andere Texte (z. B. **Text Lang**) ausgewählt werden oder mit der Schaltfläche << **DDIC-Binding aus** die Bindung deaktiviert werden. Steht für die Sprache des Benutzers der die Anwendung aufruft eine Übersetzung zur Verfügung, wird diese zur Laufzeit verwendet. Es ist auch möglich, aus dem Programmcode mit der Methode **get_available_texts** der Klasse **CL_TEXT_IDENTIFIER** Texte aus dem ABAP Dictionary zu lesen.

Die zweite Möglichkeit, sprachabhängige Texte zu verwalten, stellt das **Online Text Repository (OTR)** dar. Die hier hinterlegten Texte können auch in anderen Anwendungen als Web Dynpro-Anwendungen verwendet werden, etwa in Business Server Pages oder in Klassen. Es gibt im OTR drei Arten von Texten:

- OTR-Langtexte
- OTR-Kurztexte
- Aliastexte

OTR-Langtexte sind in ihrer Länge unbeschränkt, haben aber den großen Nachteil dass Sie nur einmal verwendet werden können. Bei einer weiteren Verwendung muss der Text erneut in allen Sprachen geschrieben bzw. übersetzt werden. OTR-Langtexte werden in Web Dynpro-Anwendungen automatisch beim Setzen von UI-Elementeneigenschaften angelegt. **OTR-Kurztexte** besitzen keine Einschränkung bezüglich der Mehrfachverwendbarkeit. Sie müssen also nicht neu geschrieben werden, wenn Sie an einer anderen Stelle erneut auftauchen sollen. Ihr Nachteil besteht in der Längenbegrenzung von 255 Zeichen. Durch die Mehrfachverwendbarkeit sind sie dennoch sehr zur Verwendung in Web Dynpros zu empfehlen. OTR-Kurztexte können mit der Transaktion **SOTR_EDIT**, oder aber direkt aus der Bearbeitung einer Web Dynpro-View über den Menüpfad **Springen -> Online Text Repository Browser** angelegt werden. OTR-Kurztexte besitzen einen Namen, der sich aus dem Paketnamen, einem Schrägstrich und einem frei wählbaren Alias zusammensetzt. Um bestehende Texte zu suchen oder Übersetzungen anzulegen ist die o. g. Transaktion zu wählen. Um einen angelegten OTR-Kurztext für eine Elementeneigenschaft zu verwenden, muss die Werthilfe der Eigenschaft aufgerufen werden. Daraufhin stehen die OTR-Kurztexte Ihres Pakets und des Pakets **SOTR_VOCABULARY_BASIC** zur Auswahl.

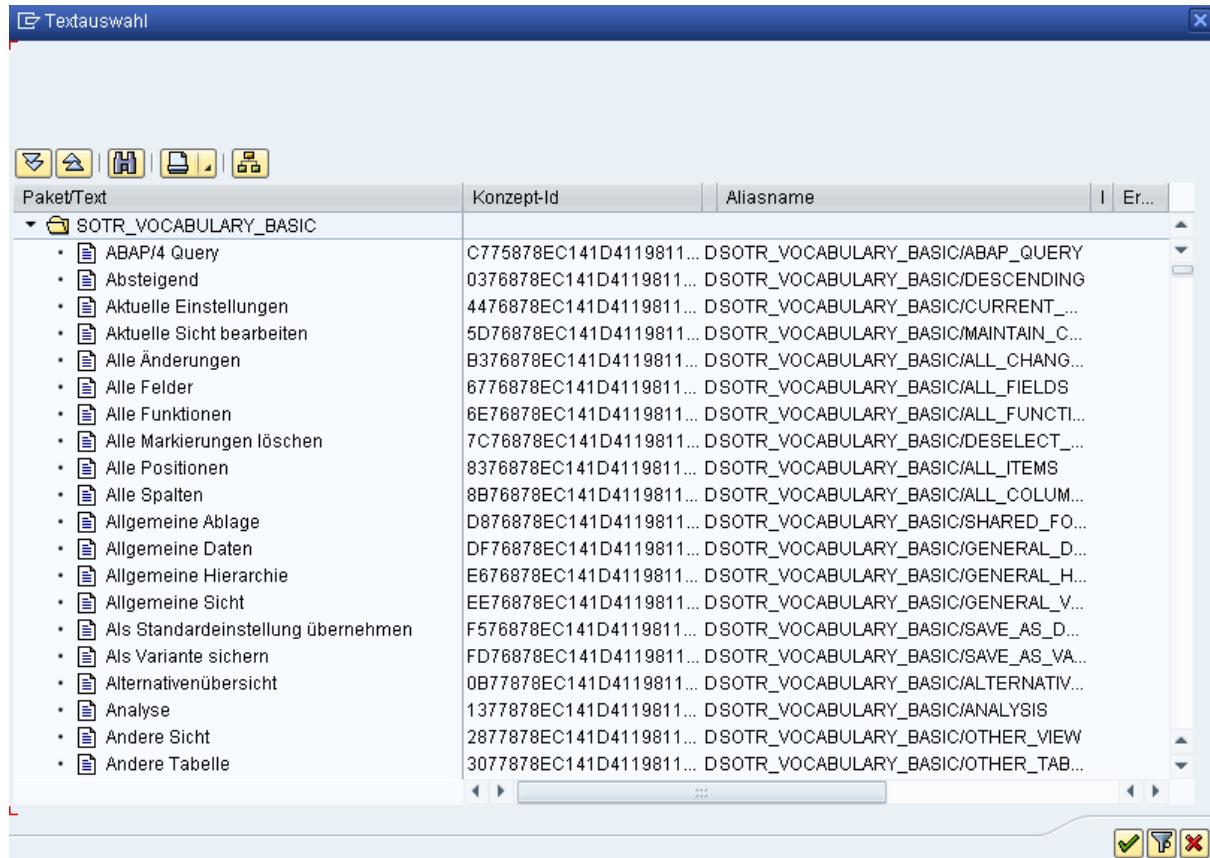


Abbildung 119: Auswahl eines OTR-Kurztextes: SAP-System-Screenshot

Hier kann der gewünschte Text markiert und bestätigt werden. Das System fügt dann eine sog. **OTR-Direktive** in das Feld ein, diese hat den Aufbau **\$OTR:<Paketname>/<Aliasname>** und sorgt dafür dass der OTR-Text verwendet wird.

Aus dem Programmcode des Controllers können auch OTR-Texte abgerufen werden. Hierfür wird die Methode **get_otr_text_by_alias** aus der Klasse **CL_WD_UTILITIES** verwendet. Diese hat den Importparameter **alias**, der in der Form **<Paketname>/<Aliasname>** anzugeben ist. Die Rückgabe erfolgt über den Returning-Parameter **value**, der als String typisiert ist. So ausgelesene Texte können dann dynamisch zur Laufzeit über den Context in die Oberfläche geschrieben werden.

Die dritte Möglichkeit für das Hinterlegen von Texten ist die Verwendung einer ABAP-Klasse. In jeder Klasse können über den Menüpfad **Springen -> Textsymbole** entsprechende Texte definiert werden. Um diese in einer Web Dynpro-Component verwenden zu können, muss die entsprechende Klasse als **Assistance-Klasse** auf der Registerkarte **Eigenschaften** der Component angegeben werden.

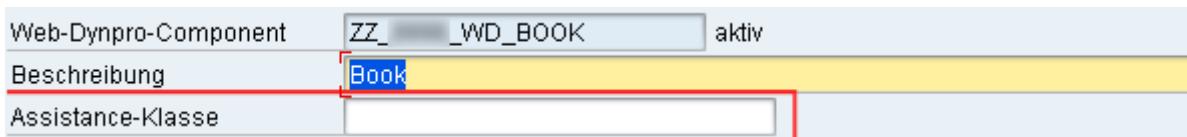


Abbildung 120: Auswahl einer Assistance-Klasse: SAP-System-Screenshot

Wird hier eine Klasse eingegeben, die noch nicht existiert, wird diese angelegt. Sie ist dann Unterklasse der Klasse **CL_WD_COMPONENT_ASSISTANCE**. Durch diese Klasse steht die Methode **get_text** zum Zugriff auf die Textelemente zur Verfügung. Um aus dem Code des eines Controllers der Component auf die Assistance-Klasse und deren get_text-Methode

zugreifen zu können, steht im Controller einer Component, die eine Assistance-Klasse besitzt, das Attribut **wd_assist** zur Verfügung. Diese referenziert eine Instanz der Assistance-Klasse, die automatisch vom System erzeugt wird.

Der Zugriff auf ein Element könnte dann wie folgt aus dem Code eines Controllers erfolgen:

```
DATA: text TYPE string.  
text = wd_assist->if_wd_component_assistance~get_text( key =  
'CAR' ).
```

Bei diesem Aufruf wird ein Textelement über seinen Identifikator 'CAR' geladen. In der Assistance-Klasse kann bei Bedarf ein Alias für die Schnittstellenmethode `get_text` definiert werden, damit der Aufruf nicht so lang ist wie hier gezeigt.

Ebenfalls ist es möglich über beliebige Konstanten auf die Textelemente zuzugreifen, indem diese mit dem Typ **WDR_TEXT_KEY** in der Assistance-Klasse angelegt werden. Der Konstantenwert entspricht dem Textelement-Identifikator in Hochkommata. Ein Beispiel könnte wie folgt aussehen:

```
DATA: text TYPE string.  
text = wd_assist->if_wd_component_assistance~get_text( key =  
wd_assist->carrid ).
```

Beim Anlegen der Textsymbole sollte darauf geachtet werden, dass die angegebene Länge auch für andere Sprachen ausreicht. Dasselbe gilt bekanntlich für die Texte, die im Dictionary hinterlegt wurden.

Bei Betrachtung der Schnittstelle der Methode `get_text` fallen die Parameter **PARA1** bis **PARA4** ins Auge:

Parameter	Art	W...	O...	Typisier...	Bezugstyp	Defaultwert	Beschreibung
KEY	Importi...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Type	WDR_TEXT_KEY		Key des Textes
PARA1	Importi...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Type	CSEQUENCE		optionaler Parameter 1
PARA2	Importi...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Type	CSEQUENCE		optionaler Parameter 2
PARA3	Importi...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Type	CSEQUENCE		optionaler Parameter 3
PARA4	Importi...	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Type	CSEQUENCE		optionaler Parameter 4
TEXT	Return...	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Type	STRING		Rückgabetext

Abbildung 121: Schnittstelle der Methode `get_text`: SAP-System-Screenshot

Mit diesen Parametern können Platzhalter **&PARA1&** bis **&PARA4&** im Text ersetzt werden. Werden diese Parameter nicht angegeben, findet keine Ersetzung der etwaigen Platzhalter statt. Diese Ersetzung muss ggf. später vorgenommen werden, etwa wenn der Text für eine Nachricht verwendet wird.

7.9.2 Auslösen von Nachrichten in Web Dynpro

Auch in Web Dynpro-Anwendungen können Nachrichten verwendet werden, um den Benutzer über Programmereignisse zu informieren. Hierfür wird ein bestimmtes Coding verwendet, das wiederum mithilfe des Wizards angelegt wird.



Abbildung 122: Nachrichten mit dem Wizard: SAP-System-Screenshot

Für das Melden von Nachrichten wird ein Nachrichtenmanager verwendet. Dieser ist für das Sammeln und Senden der Nachrichten aus der Component und etwaigen Untercomponents zuständig. Der erzeugte Code für das Melden von Nachrichten besteht aus allgemeinem Code, Code zum Holen einer Referenz auf den Nachrichtenmanager sowie Code der von der aufzurufenden Methode (zum Senden der Nachricht) abhängig ist.

```

DATA: l_current_controller    TYPE REF TO if_wd_controller,
      l_message_manager       TYPE REF TO if_wd_message_manager.

* Referenz auf den Component-Controller holen
l_current_controller ?= wd_this->wd_get_api( ).

*Referenz auf den Nachrichtenmanager holen
CALL METHOD l_current_controller->get_message_manager
  RECEIVING
    message_manager = l_message_manager.

*Nachricht melden
...

```

Abbildung 123: Generierter Code: Allgemeiner Teil und Holen der Nachrichtenmanager-Referenz

Das Interface if_wd_message_manager enthält eine Vielzahl von Methoden, um Nachrichten zu senden:

Interface **IF_WD_MESSAGE_MANAGER** realisiert / aktiv

Eigenschaften	Interfaces	Attribute	Methoden	Ereignisse	Typen	Aliases						
<input type="checkbox"/> Parameter	<input type="checkbox"/> Ausnahm...											
<input type="checkbox"/> Filter												
Methode	Art	M...	Beschreibung									
REPORT_EXCEPTION	Insta...		Berichtet eine Web-Dynpro-Ausnahme (kehrt eventuell zurück)									
REPORT_SUCCESS	Insta...		Berichtet eine Erfolgsmeldung									
REPORT_WARNING	Insta...		Berichtet eine Warnung									
REPORT_ERROR_MESSAGE	Insta...		Berichtet eine Web-Dynpro-Meldung mit optionalen Parametern									
REPORT_MESSAGE	Insta...		Berichtet eine Meldung									
REPORT_T100_MESSAGE	Insta...		Berichtet eine Meldung mit Hilfe eines T100-Eintrags									
REPORT_FATAL_EXCEPTION	Insta...		Berichtet eine fatale Web-Dynpro-Ausnahme									
REPORT_FATAL_ERROR_MESS...	Insta...		Berichtet eine fatale WDA-Meldung mit optionalen Parametern									
REPORT_ATTRIBUTE_ERROR_...	Insta...		Berichtet eine Web-Dynpro-Ausnahme zu einem Context-Attribut									
REPORT_ATTRIBUTE_EXCEPT...	Insta...		Berichtet eine Web-Dynpro-Ausnahme zu einem Context-Attribut									
REPORT_ATTRIBUTE_T100_M...	Insta...		Berichtet eine Web-Dynpro-Ausnahme zu einem Context-Attribut									

Abbildung 124: Methoden des Interfaces IF_WD_MESSAGE_MANAGER: SAP-System-Screenshot

Alle Methoden zum Melden von Nachrichten der Kategorie **TEXT** haben einheitliche Parameter. Über den Parameter **message_text** wird der Nachrichtentext übergeben. Enthält dieser Platzhalter, muss zusätzlich eine Tabelle für die Ersetzungen übergeben werden, deren Aufbau Sie dem nächsten Codebeispiel entnehmen können. Nachrichten, die sich auf ein bestimmtes UI-Element beziehen (Fehleingabe des Benutzers), muss über den Parameter **element** eine Referenz auf dieses Element sowie über den Parameter **attribute_name** der Name des Attributs im Kontextelement übergeben werden, das den fehlerhaften Wert enthält. Der Attributwert ist an eine Eigenschaft des UI-Elements gebunden und die Nachricht kann so vom System zugeordnet werden.

Die folgende Abbildung zeigt ein Beispiel zum Melden einer solchen Nachricht.

```

DATA: lv_text      TYPE string,
      lt_params TYPE wdr_name_value_list,
      ls_param  TYPE wdr_name_value.

* als Textsymbol in Assistance-Klasse definierten übersetzbaren
* Text holen
lv_text = wd_assist->if_wd_component_assistance~get_text( ... ).

* Parametertabelle füllen
* (Text enthält Platzhalter X1)
ls_param-name = 'X1'.
ls_param-value = 'LH'.
APPEND ls_param TO lt_params.

* Nachricht bezogen auf UI-Element melden
* wd_this->gr_element_flight ist Referenz auf Kontextelement
* FLIGHT in Lead-Selection
CALL METHOD l_message_manager->report_attribute_error_message
  EXPORTING
    message_text      = lv_text
    params            = lt_params
    element           = wd_this->gr_element_flight
    attribute_name    = 'CARRID'.

```

lv_text = 'Fluggesellschaft &X1
nicht gefunden'

Abbildung 125: Melden einer Nachricht

Die Nachricht enthält einen Platzhalter. Zum Füllen dieses Platzhalters wird eine interne Tabelle des Typs **wdr_name_value_list** über einen Arbeitsbereich des Typs **wdr_name_value** mit den entsprechenden Ersetzungen befüllt und beim Aufruf der Methode als Parameter **params** übergeben.

Für Nachrichten der Kategorie **T100**, das sind Nachrichten, die in der Datenbanktabelle T100 definiert sind, stehen zwei Methoden zur Verfügung. Diese erwarten als Parameter die Nachrichtennummer, die Nachrichtenklasse und den Nachrichtentyp. Zum Ersetzen von Platzhaltern stehen je nach Methode die Parameter **p1** bis **p4** oder ein mit einer internen Tabelle von Strings typisierter Parameter zur Verfügung. Die Schnittstellen sind leider nicht einheitlich. Der Nachrichtentext wird jedoch immer über den Parameter **message_text** übergeben. Analog zu den Nachrichten der Kategorie TEXT wird auch hier bei Nachrichten, die sich auf UI-Elemente beziehen, eine Referenz auf das Kontextelement und der Name des Attributs übergeben werden.

```

DATA: ls_message TYPE symsg.

* T100-Nachricht mit Platzhalter P1 melden
CALL METHOD l_message_manager->report_t100_message
EXPORTING
  msgid = 'XYZ'           Nachrichenklasse
  msgno = '001'
  msgty = 'E'
  P1    = 'LH'.           Text der Nachricht
                        z. B. „Fluggesellschaft &1 existiert nicht“

* Nachrichtenfragmente definieren
ls_message-msgid= 'XYZ'.
ls_message-msgno= '001'.
ls_message-msgty= 'E'.
ls_message-msgv1= 'LH'.

*T100-Nachricht bezogen auf UI-Elemente mit Platzhalter P1 melden
CALL METHOD l_message_manager->report_attribute_t100_message
EXPORTING
  msg          = ls_message
  element      = wd_this->gr_element_flight
  attribute_name = 'CARRID'.

```

Abbildung 126: Beispiele für Aufrufe unterschiedlicher Methoden zum Melden von Nachrichten

Eine weitere Kategorie von Nachrichten stellen **EXCEPTIONS** dar. Aus dem Kapitel zu klassenbasierten Ausnahmen wissen Sie, dass ein Ausnahmeobjekt gefangen werden kann, das den zum Fehler gehörenden Text beinhaltet, der in der Ausnahmeklasse festgelegt wurde. Anhand des Ausnahmeobjekts kann eine Web Dynpro-Nachricht gemeldet werden. Den Methoden der Kategorie **EXCEPTIONS** wird über den Parameter **message_objekt** das Ausnahmeobjekt übergeben. Bei Meldungen, die sich auf ein UI-Element beziehen, muss auch hier eine Referenz auf das Kontextelement und der Name des Attributs übergeben werden. Die folgende Abbildung zeigt ein Beispiel für das Melden einer Nachricht dieser Kategorie.

```

DATA: lr_msg_obj    TYPE REF TO cx_root
      lv_result TYPE p DECIMALS 3 LENGTH 3.

*Ausnahmenachricht bezogen auf UI-Element melden
TRY.
  lv_result = wd_this->gv_integer1 / wd_this->gv_integer2.
  CATCH cx_root INTO lr_msg_obj. → Nachrichtentext definiert in Ausnahmeklasse
  CALL METHOD l_message_manager->report_attribute_exception
    EXPORTING
      message_object  = lr_msg_obj
      element         = wd_this->gr_elem_calculator
      attribute_name  = 'INTEGER2'.

ENDTRY.

```

Abbildung 127: Melden einer Nachricht der Kategorie EXCEPTIONS

Das gefangene Ausnahmeobjekt `lr_msg_obj` wird hier an die Methode zum Melden der Nachricht weitergegeben. Außerdem werden die Parameter angegeben, die erforderlich sind, da sich die Nachricht auf ein bestimmtes UI-Element bezieht.

Nachrichten werden häufig im Zusammenhang mit der Validierung von Benutzereingaben gesendet. Diese Validierungen können in den Behandlermethoden der jeweiligen Aktion (**onaction...**) implementiert werden. Wenn es jedoch mehrere Aktionsbehandlermethoden gibt, kann es vorkommen, dass die Validierungen an mehreren Stellen implementiert werden müssen. Es ist dann sinnvoll, die Validierungen in eine eigene Methode auszulagern, um Redundanzen zu vermeiden. Die Hook-Methode **wddobeforeaction** des View-Controllers wird vor allen Aktionsbeandlermethoden ausgeführt. Daher lassen sich hier entsprechende Prüfungen gut unterbringen. Bei geschachtelten Views werden erst alle **wddobeforeaction**-Methoden aufgerufen, bevor eine Aktionsbeandlermethode aufgerufen wird.

Bei Aktionen kann auf der Registerkarte **Aktionen** der View eingestellt werden, ob es sich um eine Standard- oder eine validierungsunabhängige Aktion handelt. Bei **validierungsunabhängigen Aktionen** beeinflussen Nachrichten, die aus der **wddobeforeaction**-Methode heraus gemeldet werden nicht die Navigation oder die Hook-Methoden-Verarbeitung. Bei **Standard-Aktionen** hingegen führen Meldungen über schwerwiegende Fehler sowie Fehlermeldungen über Kontextattribute zum Abbruch der Navigation, außerdem wird die Methode **wddomodifyview** nicht ausgeführt. Meldungen, die nicht in **wddobeforeaction**, sondern in Aktionsbeandlermethoden ausgelöst werden, haben keinen Einfluss auf die Navigation oder die Hook-Methoden-Verarbeitung.

7.9.3 Praxis: Übung zur Internationalisierung

In dieser Übung werden Sie die noch nicht internationalisierten Texte der letzten Web Dynpro-Anwendung in OTR-Elemente überführen. Diese können dann von einem Übersetzer übersetzt werden und stehen daraufhin in der jeweiligen Sprache zur Verfügung, ohne dass Anpassungen an der Anwendung erforderlich sind.

Öffnen Sie die erste View Ihrer Web Dynpro-Component zur Buchungsanzeige. Wählen Sie dort den Menüpfad **Springen -> Online Text Repository Browser**. Klicken Sie im Fenster

das nun erscheint auf um ein neues Element anzulegen (Hinweis: Die View muss im Änderungsmodus geöffnet sein, sonst fehlt dieser Button). Dieses soll die Überschrift auf der ersten View ersetzen. Wählen Sie einen Alias und eine Länge und geben Sie einen passenden Text ein.

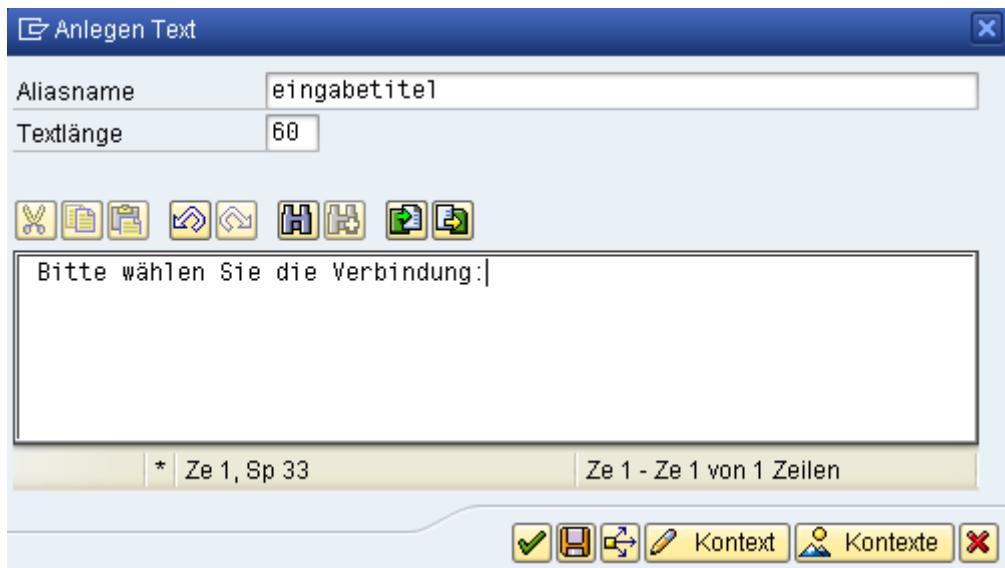


Abbildung 128: Beispiel für einen OTR-Text: SAP-System-Screenshot

Sichern Sie den Text unter Bestätigung des Pakets und kehren Sie zur View zurück. Wählen Sie dort die Überschrift aus und rufen Sie für die Eigenschaft **text** des TextView-Elements die Werthilfe auf. Wählen Sie Ihren soeben angelegten Text aus und fahren Sie fort. Sichern und aktivieren Sie die View.

Überführen Sie analog auch die Überschrift der zweiten View und die Texte auf den Buttons in OTR-Elemente. Sichern und aktivieren Sie auch diese View und Testen Sie die Anwendung. Sie sehen nun die im OTR definierten Texte. Ein Übersetzer könnte nun über spezielle Übersetzungstransaktionen Übersetzungen in andere Sprachen vornehmen.

7.9.4 Praxis: Übung zu Nachrichten

In dieser Übung soll die Buchungs-Web Dynpro-Anwendung um eine Fehlermeldung ergänzt werden, die erscheint, wenn zu den Eingaben auf der ersten View überhaupt keine Flüge existieren. Öffnen Sie dazu die erste View und wechseln Sie zur Registerkarte **Methoden**. Klicken Sie dort doppelt auf die Methode **WDDOBEFOREACTION**. Hier soll die entsprechende Prüfung durchgeführt werden.

Damit nach den Flügen gesucht werden kann, benötigen Sie zunächst die eingegebenen Daten aus dem Context. Um diese zu lesen, öffnen Sie den Wizard und wählen Sie **Auslesen** auf der Registerkarte **Context**. Wählen Sie mit der Werthilfe den Kontextknoten **CONNECTION**.

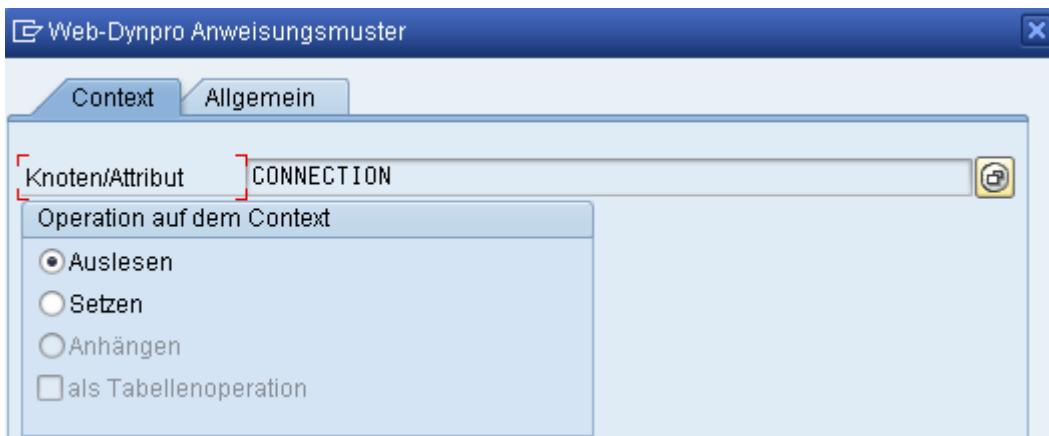


Abbildung 129: Auslesen des Kontextknotens: SAP-System-Screenshot

Der Wizard legt nun den Code an, der die entsprechenden Daten ausliest. Diese stehen nun als Struktur **ls_connection** mit den Komponenten **carrid** und **connid** zur Verfügung.

Nutzen Sie diese Daten, um mit einer SQL-Anfrage zu ermitteln, ob Flüge zu den Eingaben existieren oder nicht. Wenn keine Flüge existieren, soll die im folgenden Schritt programmierte Fehlermeldung ausgelöst werden.

Lassen Sie den Code für die Fehlermeldung ebenfalls über den Wizard generieren, indem Sie dort auf der **Allgemein**-Registerkarte **Meldung erzeugen** wählen und als Methode **report_element_error_message** eingeben.

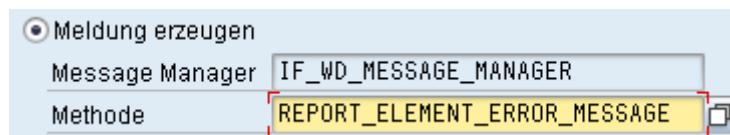


Abbildung 130: Meldung erzeugen: SAP-System-Screenshot

Der Wizard generiert nun den entsprechenden Code. Kommentieren Sie aus dem Aufruf der Methode den RECEIVING-Teil aus. Übergeben Sie als message_text einen passenden Text, der die Platzhalter &carrid und &connid an den Stellen enthält, an denen diese Daten später Teil der Nachricht werden sollen. Als element übergeben Sie der Methode das Kontextelement, das im Wizard-Code zum Auslesen der Benutzereingaben bereits bestimmt wird (`lo_el_connection`).

Es fehlt nun noch die Übergabe der Parameter, die im Nachrichtentext die Platzhalter ersetzen sollen. Hierfür dient der Parameter `params` der Methode. Kommentieren Sie diesen ein, indem Sie das Sternchen am Zeilenanfang entfernen. Als Übergabewert wird hier eine interne Tabelle des Typs **WDR_NAME_VALUE_LIST** erwartet. Definieren Sie daher (vor dem Methodenaufruf) eine interne Tabelle dieses Typs (Tipp: Kein TABLE OF da der Typ bereits ein Tabellentyp ist) sowie einen entsprechenden Arbeitsbereich vom Typ **WDR_NAME_VALUE**. Erzeugen Sie mit dem Arbeitsbereich zwei Zeilen in der internen Tabelle. Die Komponenten der Zeile heißen **name** und **value**. Geben Sie für die erste Komponente '`carrid`' bzw. '`connid`' (mit den Anführungszeichen) und für die zweite Komponente die entsprechende Komponente der Struktur `ls_connection`.

Übergeben Sie die so gefüllte interne Tabelle an den Parameter `params` des Methodenaufrufs. Speichern und aktivieren Sie, und testen Sie Ihre Anwendung. Bei

entsprechenden Eingaben sollte nun Ihre Fehlermeldung erscheinen und die Navigation unterbrochen werden (siehe folgende Abbildung).

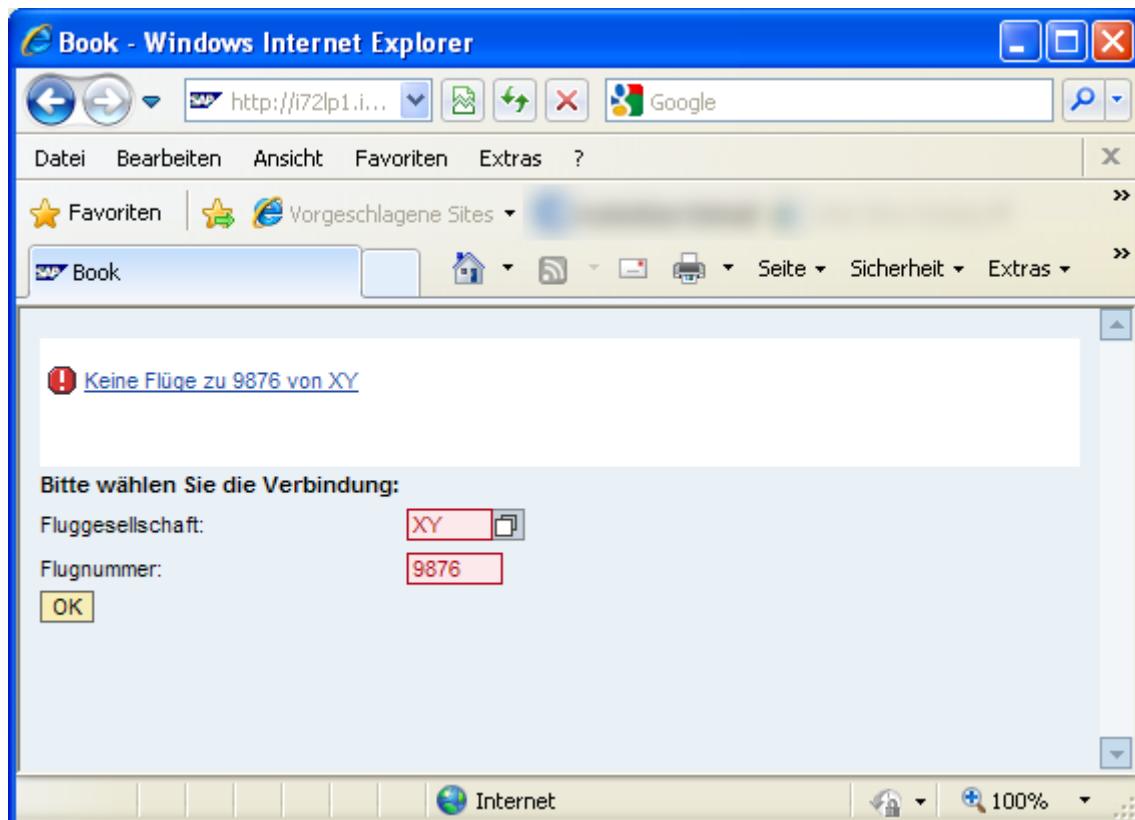


Abbildung 131: Fertige Fehlermeldung: SAP-System-Screenshot

7.10 Kontrollfragen

1. Welche Kardinalitäten kann ein Kontextknoten besitzen?
2. Warum werden View-Controller nicht als Datenquelle verwendet?
3. Kann durch Datenbindung nur die value-Eigenschaft eines UI-Elements über den Context geändert werden?
4. Kann die Datenbindung eines UI-Elements sich nur auf Kontextelemente desselben View-Controllers beziehen?
5. Kann eine UI-Elementeigenschaft an ein Controller-Attribut gebunden werden?
6. Wozu dient eine Supply-Funktion?
7. Was ist ein Containerelement?
8. Welche Werte kann die Layout-Eigenschaft eines Containerelements haben?
9. In welchem Controller befindet sich die Hook-Methode wddobeforeaction?
10. Können OTR-Kurztexte mehrfach verwendet werden?
11. Können bei einem Table-UI-Element mehrere Zeilen gleichzeitig gewählt werden?

7.11 Antworten

1. 0..1, 1..1, 0..n oder 1..n
2. Das MVC-Konzept würde verletzt. Ein verwendender Controller würde so vom View-Controller abhängig.
3. Nein, auch andere Eigenschaften sind für die Datenbindung geeignet.
4. Ja
5. Nein
6. Sie versorgt einen ungültigen Kontextknoten mit Daten, etwa wenn sich die Lead-Selection des übergeordneten Knotens geändert hat
7. Ein Containerelement ist ein UI-Element, das untergeordnete Elemente haben kann.
8. FlowLayout, RowLayout, MatrixLayout oder GridLayout
9. im View-Controller
10. Ja
11. Wenn die Auswahlkardinalität dies zulässt, ja.

7.12 Kapitelabschluss

Sie befinden sich am Ende dieses Abschnitts. Bevor sie die im folgenden Absatz beschriebene E-Mail verfassen, beachten Sie bitte die folgenden Hinweise:

1. Prüfen Sie, ob sie wirklich **alle** Aufgaben seit dem vorangegangenen Abschluss bearbeitet haben. Diese sind mit „Praxis.“ in der Überschrift gekennzeichnet (7.4, 7.5.5, 7.6.5, 7.7.1, 7.7.5, 7.7.9, 7.7.10, 7.8.5, 7.8.6, 7.9.3, 7.9.4).
2. Prüfen Sie bitte noch einmal genau ob alle ihre Repository-Objekte **korrekt funktionieren**.
3. Stellen Sie sicher, dass alle Repository-Objekte **aktiviert** sind. Um Objekte zu finden, die noch nicht aktiviert sind, wählen Sie aus dem Drop-Down-Menü oberhalb des Navigationsbaums im Object Navigator **Inaktive Objekte** aus. Geben Sie anschließend im darunter befindlichen Feld ihren Benutzernamen **USER#-###** ein und bestätigen Sie. Anschließend werden im Navigationsbaum die inaktiven Objekte dargestellt, die noch aktiviert werden müssen. Aktivieren Sie diese nun. Beachten Sie, dass sie die Zweige des Baums ggf. noch aufklappen müssen. Um zu ihrem Paket zurückzukehren, wählen Sie im Drop-Down-Menü wieder **Paket** aus und bestätigen Sie ihren Paketnamen.
4. Stellen Sie weiterhin sicher, dass die **Namen** ihrer Entwicklungsobjekte genau den Vorgaben im Skript entsprechen. Sollten Sie sich vertippt haben, können Sie Programme umbenennen, indem Sie diese mit der rechten Maustaste im Navigationsbaum des Object Navigators anklicken und **Umbenennen...** auswählen.

Wenn Sie den Kurs bis zu dieser Stelle bearbeitet haben, senden Sie bitte eine formlose E-Mail an die vom Kursbetreuer für diesen Kurs genannte Adresse mit dem Betreff „ABAP2: Abschluss Kapitel 7 User ####“ (die Anführungszeichen gehören nicht mit zum Betreff). Sie erhalten dann in Kürze Feedback (je nach Ergebnis entweder über den Fortschrittsbericht, wenn alles in Ordnung ist, oder per E-Mail, wenn noch Korrekturen nötig sind) und können Mängel ggf. noch nachbessern. Bitte achten Sie darauf, den Betreff genau wie angegeben zu formulieren, um eine effiziente Verarbeitung der Mail zu ermöglichen.

Sollten Sie Fragen haben, formulieren Sie diese bitte in einer **separaten E-Mail** mit aussagekräftigem Betreff, da die Kapitelabschluss-mails meist nur über den Betreff verarbeitet werden!