Visual Modeler

Implementierungshandbuch

Release 91

Visual Modeler

Implementierungshandbuch

Release 91

Hinweis

Vor Verwendung dieser Informationen und des darin beschriebenen Produkts sollten die Informationen unter „Bemerkungen" auf Seite 129 gelesen werden.

Inhaltsverzeichnis

Kapitel 1. Implementierungsmethodik

Die Implementierungsmethodik für Visual Modeler besteht aus verschiedenen Phasen, dank derer sichergestellt ist, dass die Implementierung geplant und durchgehend bis zu ihrem Abschluss überwacht und protokolliert werden kann.

In der Tabelle zur Implementierungsmethodik für Visual Modeler finden Sie eine Zusammenfassung dieser Phasen sowie der zur Ausführung der einzelnen Phasen erforderlichen Aktivitäten. Zum Überwachen und Protokollieren der verschiedenen Phasen steht eine Gruppe standardmäßiger Dokumente zur Verfügung.

Implementierungsphase
Beschreibung

Planen
Implementierung planen: Zeitachse und Meilensteine definieren und Risiken und Abhängigkeiten ermitteln

Analysieren
Organisation und Verwaltung, Geschäftsregeln definieren, Benutzerschnittstelle, Nachrichtenprotokolle, Datenquellen, Ablaufplanung E-Commerce, Schulungsbedarf, Rolloutstrategie, Vorbereitung der Umgebung, Planung der Operationen

Entwerfen und konfigurieren
Installation, Konfiguration, Integration, Komponententest und Entwicklung der Schulung

Testen und implementieren
Serverkonfiguration testen, Kommunikation von Unternehmen zu Partner, Kommunikation von Partner zu Unternehmen, auf Produktionssysteme umstellen, Distributorschulung, Dokumentationsbereitstellung, Unterstützung

Verbessern
Aktuelle Erweiterungsaktivitäten, Partnerschulung und Unterstützung

Kapitel 2. Visual Modeler-Integration implementieren

Visual Modeler ist so gestaltet, dass Channel-Partner in ein E-Commerce-Netz integriert werden können. Organisationen im Netz arbeiten als Unternehmen und deren Partner. Von jeder als Unternehmen fungierenden Organisation wird die jeweilige Version des Unternehmensservers installiert, um so nahtlos Informationen an die entsprechenden Channel-Partner übertragen zu können.

Jeder Wiederverkäufer oder Distributor kann mit mehreren Unternehmen zusammenarbeiten. Daher muss es mit der jeweiligen Installation des Unternehmensservers möglich sein, Nachrichten von verschiedenen Unternehmensservern zu empfangen und zu beantworten.

In der folgenden Tabelle sind die wichtigsten Aktivitäten für eine Implementierung von Visual Modeler zusammengefasst:

Implementierungsphase
Task

Planen
Projektanalyse

Analysieren
- Konfigurationsanalyse
- Integrationsanalyse
- Bedarfsanalyse

Entwerfen und konfigurieren
- Vorbereitung der Servlet-Containerumgebung
- Installation der Wissensdatenbank
- Einrichtung der Wissensdatenbank
- Visual Modeler-Konfiguration
- Rollen- und Sicherheitsdefinition
- Systemadministratorauthentifizierung
- XML Schema-Erstellung
- Anpassen von BizAPIs, BLCs und Controllern
- Anpassen von JSP-Seiten

Testen und implementieren
- Produktintegration
- Serverkonfiguration testen
- Kommunikation von Unternehmen zu Partner testen
- Kommunikation von Partner zu Unternehmen testen
- Freigabe für Produktionssysteme

Verbessern
Bewerten und verbessern

Kapitel 3. Implementierungsschritte

Bei der Implementierung von Visual Modeler werden in erster Linie die folgenden Tasks ausgeführt:

- *Projektanalyse*: Zustimmen zu einem Zeitplan für das Implementierungsprojekt und Definieren einer Zeitachse. Definieren von Meilensteinen zum Ermitteln des Verarbeitungsfortschritts bei der Implementierung und Aufdecken von Abhängigkeiten und Risiken, die dazu führen könnten, dass die Implementierung nicht rechtzeitig abgeschlossen werden kann.
- *Konfigurationsanalyse*: Ermitteln einer passenden Visual Modeler-Konfiguration (Anzahl der zu verwendenden Maschinen und deren Positionen in internen Netzen in Relation zu Firewalls und Proxy-Servern). Weitere Informationen zu einer in Gruppen auszuführenden Implementierung finden Sie im Abschnitt zum Thema *Hohe Verfügbarkeit und Lastverteilung* im *VM Installationshandbuch*.
- *Integrationsanalyse*: Ermitteln von Integrationspunkten in vorhandenen E-Commerce-Systemen.
- *Bedarfsanalyse*: Überprüfen der Hardware- und Softwareanforderungen, um so sicherzustellen, dass die Maschinen über ausreichend Leistungspotenzial verfügen, um den zu erwartenden Datenverkehr und die erforderlichen Antwortzeiten unterstützen zu können.
- *Installation von Visual Modeler*: Installieren von Visual Modeler auf der/den dafür vorgesehenen Maschine(n). Weitere Informationen hierzu finden Sie im Abschnitt zum Thema *Installationsübersicht* im *VM Installationshandbuch*.
- *Einrichtung der Wissensdatenbank*:
  1. Installation der Wissensdatenbank: Installieren des Wissensdatenbankschemas auf dem dafür vorgesehenen Datenbankserver.
  2. Einrichtung der Wissensdatenbank: Überprüfen der Konnektivität zum Datenbankserver für die Wissensdatenbank und Bereitstellen aller für E-Commerce erforderlichen Informationen. Dazu gehören (u. a.) die Partnerprofile für Ihre Partner, Ihr Produktkatalog sowie die Informationen zur Preisliste.

     Weitere Informationen hierzu finden Sie im *VM Installationshandbuch*.
- *Visual Modeler-Konfiguration*: Modifizieren der Konfigurationsdateien, um so die Systemkonfiguration in Ihrer Produktionsumgebung zu definieren.
- *Rollen- und Sicherheitsdefinition*: Definieren von Gruppen und Rollen und entsprechendes Modifizieren von Konfigurationsdateien und ACL-Scripts. Damit legen Sie die Zugriffsberechtigungen für die Benutzer Ihres Unternehmensservers fest.
- *Schemaerstellung*: Erstellen des Geschäftsobjektschemas, um so die Datenquelleninformationen bereitzustellen. Über die Datenebene wird der Zugriff zwischen Unternehmensserver und externen Systemen gesteuert.
- *Anpassen von BLCs und Controllern*: Modifizieren der Geschäftslogik- und Controllerklassen zur Unterstützung Ihrer Geschäftslogik. In einigen Fällen müssen Sie die Java-Klassen ändern, um so Geschäftsprozesse speziell für Ihr Unternehmen implementieren zu können.
- *Anpassen von JSP-Seiten*: Modifizieren von Schablonen, um so Ihren Anforderungen in Bezug auf "Darstellung und Funktionsweise", Suchvorgänge und statisches Paging gerecht werden zu können. Die von Visual Modeler

bereitgestellten JSP-Seiten dienen der Anzeige der Browserseiten. Sie können entsprechend den Anforderungen Ihres Unternehmens angepasst werden.

- *Produktintegration*: Importieren von Produktinformationen in die Wissensdatenbank oder Bereitstellen einer Punchout-Integration. Wenn im Rahmen Ihrer Implementierung auch Bestellungen aus einem Produkt eines anderen Anbieters unterstützt werden sollen, müssen Sie einen Weg finden, um die Produktdaten in Visual Modeler zu integrieren.
- *Serverkonfiguration testen*: Bevor Sie Visual Modeler implementieren, müssen Sie das System ausgiebig testen. Mithilfe einiger bereitgestellter Scripts können Sie die wichtigsten funktionalen Komponenten testen.
- *Kommunikation von Unternehmen zu Partner testen*: Senden von Testnachrichten vom Unternehmensserver an andere Unternehmensserver.
- *Kommunikation von Partner zu Unternehmen testen*: Senden von Testnachrichten von anderen Unternehmensservern an Ihren Unternehmensserver.
- *Bewerten und verbessern*: Sobald Visual Modeler implementiert ist, müssen Sie einen fortlaufenden Prozess zur Analyse der Verwendung und Leistung planen.

Kapitel 4. Visual Modeler und Sterling Selling and Fulfillment Foundation verknüpfen

Visual Modeler mit Sterling Selling and Fulfillment Foundation verknüpfen

In manchen Instanzen müssen komplexe Produkte möglicherweise erst konfiguriert werden, bevor sie von den Kunden erworben werden können. In manchen anderen Instanzen verfügen diese Produkte möglicherweise über optionale Komponenten, die die Kunden auf der Basis ihrer Anforderungen konfigurieren können. Mit Visual Modeler können Sie Modelle erstellen, in denen die konfigurierbaren Optionen eines Produkts definiert werden. Außerdem können Sie diesen Modellen Produkte zuordnen. Bei IBM Sterling Configurator handelt es sich um ein Tool, mithilfe dessen den Benutzern die konfigurierbaren Produkte zusammen mit den verfügbaren Optionen angezeigt werden können.

Die Verknüpfung zwischen Visual Modeler und IBM Sterling Selling and Fulfillment Foundation wird benötigt, damit Daten zwischen den beiden Programmen ausgetauscht werden können. Die Verknüpfung ist erforderlich, um sicherstellen zu können, dass zum Definieren der Modelle in Visual Modeler die korrekten Produktinformationen entsprechend der Erfassung in Sterling Selling and Fulfillment Foundation verwendet werden. Die auf die Produkte angewendeten Preise basieren auf der Preisliste und der im Zusammenhang mit dem Gastbenutzer verwendeten Währung. Weitere Informationen zum Zuordnen der Preisliste finden Sie im *Business Center: Preisermittlung Verwaltungshandbuch*.

Für die Verknüpfung von Visual Modeler mit Sterling Selling and Fulfillment Foundation müssen Sie in Visual Modeler und Applications Manager bestimmte Konfigurationen vornehmen.

Visual Modeler-Eigenschaften konfigurieren

Informationen zu diesem Vorgang

Sie müssen die Werte bestimmter Eigenschaften in Visual Modeler konfigurieren, damit das Programm die korrekten Produktinformationen aus Sterling Selling and Fulfillment Foundation anfordern kann.

So konfigurieren Sie die Eigenschaften in Visual Modeler:

Vorgehensweise

1. Zeigen Sie mit Ihrem Browser auf die folgende URL:

   `http://<hostname>:<port>/<kontextstammverzeichnis>/en/US/enterpriseMgr/admin`

   Bei "hostname" handelt es sich um die IP-Adresse, bei "port" um den Empfangsport der Maschine, auf der Visual Modeler installiert ist und bei "kontextstammverzeichnis" um das Kontextstammverzeichnis der per Hosting bereitgestellten Visual Modeler-Anwendung.

   Die Anmeldeseite wird angezeigt.
2. Melden Sie sich als Administrator an. Geben Sie dazu Ihre Anmelde-ID ein und klicken Sie auf **Anmelden**.

3. Klicken Sie auf den Hyperlink **Systemservices**. Die Seite "Systemeigenschaften" wird angezeigt.
4. Klicken Sie auf den Hyperlink **Ausführung**. Daraufhin wird "Eigenschaften für" mit der Seite "Ausführung" angezeigt.
5. Geben Sie für die Eigenschaft "Sterling Order Fulfillment System URL" den Wert `http://<hostname>:<port>/smcfs/interop/InteropHttpServlet` an. Diese URL bezieht sich auf das Interoperabilitätsservlet von Sterling Selling and Fulfillment Foundation.
6. Geben Sie für die Eigenschaft "Sterling Configurator URL" den folgenden Wert an:

   `http://<hostname>:<port>/sbc/configurator/configure.action`

   Bei "hostname" handelt es sich um die IP-Adresse der Maschine, auf der Sterling Selling and Fulfillment Foundation installiert ist und bei "port" um den Empfangsport der Maschine, auf der Sterling Selling and Fulfillment Foundation installiert ist.
7. Definieren Sie die folgenden Eigenschaften entsprechend:
   - Benutzername für das Sterling-Auftragsausführungssystem
   - Kennwort für das Sterling-Auftragsausführungssystem

     Über die Werte dieser Eigenschaften werden der Benutzername und das Kennwort festgelegt, der/das für die Kommunikation mit dem Sterling Selling and Fulfillment Foundation-Server verwendet wird.

Sterling Configurator-Regeln konfigurieren

Informationen zu diesem Vorgang

Damit Sterling Configurator in der Lage ist, die Modellinformationen der Produkte aus Visual Modeler zu empfangen, müssen Sie über Applications Manager die Positionen der Modelle, die Eigenschaften sowie die Regeln angeben, die sich auf Modelle beziehen.

So konfigurieren Sie die Regeln für Sterling Configurator:

Vorgehensweise

1. Melden Sie sich über die Anmeldeseite als Administrator an. Geben Sie dazu Ihre Anmelde-ID und Ihr Kennwort ein und klicken Sie auf **Anmelden**. Die Startseite der Application Console wird angezeigt.
2. Navigieren Sie über die Menüleiste zu **Konfigurationen** > **Applications Manager starten**. Applications Manager wird in einem neuen Browserfenster gestartet.
3. Navigieren Sie über die Menüleiste von Applications Manager zu **Anwendungen** > **Anwendungsplattform**. Das Fenster mit den Anwendungsregeln wird angezeigt.
4. Wählen Sie im Fenster mit den Anwendungsregeln nacheinander **Systemadministration** > **Artikelkonfigurator** aus.
5. Geben Sie die Pfade zu der Position an, an der die Modelle, die Eigenschaftendateien und die Regeln gespeichert werden.

Ergebnisse

Hinweise:

- Alle in Applications Manager für das Modellrepository angegebenen Pfade werden von Sterling Selling and Fulfillment Foundation und Visual Modeler gemeinsam genutzt. Befinden sich Sterling Selling and Fulfillment Foundation und Visual Modeler auf verschiedenen Maschinen, sollten die Pfade zu einem Laufwerk führen, das für beide zugänglich ist. Weitere Informationen zum Modellrepository finden Sie im *Selling and Fulfillment Foundation: Application Platform Configuration Guide*.
- In IBM Sterling Business Center kann der Artikeldefinition eines Paketartikels ein Modell zugeordnet werden. Der Name des Modells wird in der Artikeldefinition gespeichert. Wenn Sie den Modellnamen nach dessen Speicherung in der Artikeldefinition irgendwann ändern, muss die Artikeldefinition entsprechend dem geänderten Modellnamen ebenfalls geändert werden. Zu dieser Situation kann es kommen, wenn ein Benutzer die Modelldefinition in Visual Modeler bearbeitet.

Kapitel 5. Einführung in J2EE-Webanwendungen

In diesem Abschnitt finden Sie eine Überblick über Java 2 Platform, Enterprise Edition (J2EE) und dessen Verwendung beim Implementieren von Webanwendungen. Wenn Sie mit dieser Architektur bereits vertraut sind, können Sie den Abschnitt überspringen.

Architektur

Visual Modeler wurde so entworfen, dass es der Architektur von Java 2 Platform, Enterprise Edition (J2EE) entsprechend der Definition in der *Java 2 Platform Enterprise Edition Specification, v 1.2* von Sun Microsystems Inc. entspricht.

Visual Modeler wird als Webanwendung mit einer Gruppe von Java-Klassen und dazugehörigen Konfigurationsdateien, HTML-Schablonen und JSP-Seiten (JavaServer Pages) implementiert. Das Programm muss in einem dem J2EE-Standard entsprechenden Servlet-Container installiert werden.

Webanwendungen

Eine J2EE-Webanwendung ist so aufgebaut, dass sie einer J2EE-Spezifikation entspricht. Wenn Sie Webkomponenten zu einem J2EE-Servlet-Container hinzufügen, erfolgt das in einem Paket als WAR-Datei (Web application archive). Bei einer WAR-Datei handelt es sich um eine als Java-Archiv (JAR) komprimierte Datei.

Eine WAR-Datei enthält neben Webkomponenten gewöhnlich noch andere Ressourcen wie:

- Serverseitige Dienstprogrammklassen
- Statische Webressourcen (Konfigurationsdateien, HTML-Seiten, Grafik- und Audiodateien etc.)
- Clientseitige Klassen (Applets und Dienstprogrammklassen)

Die Verzeichnis- und Dateianordnung einer als WAR-Datei implementierten Webanwendung folgt einer präzisen Struktur. Eine WAR-Datei verfügt über eine spezielle hierarchische Verzeichnisstruktur. Das Basisverzeichnis einer WAR-Datei ist das *Dokumentstammverzeichnis* der Anwendung. Beim Dokumentstammverzeichnis handelt es sich um das Verzeichnis, unter dem JSP-Seiten, clientseitige Klassen und Archive und statische Webressourcen gespeichert werden. Das Dokumentstammverzeichnis verfügt über ein Unterverzeichnis **WEB-INF/** mit den folgenden Dateien und Verzeichnissen:

- **web.xml**: Implementierungsdeskriptor der Webanwendung. Darin wird die Struktur der Webanwendung beschrieben.
- Deskriptordateien der Tagbibliothek.
- **classes/**: Verzeichnis mit serverseitigen Klassen: Servlets, Dienstprogrammklassen und Java Beans-Komponenten.
- **lib/**: Verzeichnis mit JAR-Archiven von Bibliotheken (Tagbibliotheken und beliebige Dienstprogrammbibliotheken, die von serverseitigen Klassen aufgerufen werden).

Datei web.xml

Jede in einem Servlet-Container implementierte Webanwendung muss über eine **web.xml**-Datei im entsprechenden **WEB-INF/**-Verzeichnis verfügen. Die Struktur dieser **web.xml**-Dateien entspricht einer Dokumenttypdefinition (Document Type Definition, DTD) entsprechend der Veröffentlichung in einer J2EE-Spezifikation.

Zweck einer **web.xml**-Datei ist die Angabe der allgemeinen Konfiguration der Webanwendung entsprechend den Anforderungen des J2EE-Standards. Insbesondere geht es um folgende Angaben:

- Es werden Initialisierungsparameterwerte für die Webanwendung bereitgestellt.
- Es können von der Webanwendung zu verwendende Servletklassen deklariert und benannt werden.
- Jede Servletklasse wird einem oder mehreren URL-Mustern zugeordnet: Wenn durch den Servlet-Container eine Anforderung empfangen wird, deren URL einem in der **web.xml**-Datei definierten Muster entspricht, wird das jeweilige Servlet zum Verarbeiten der Anforderung verwendet.
- Bei Bedarf werden für jedes Servlet Initialisierungsparameterwerte bereitgestellt.
- Sitzungsdaten (wie Zeitlimit)
- Positionen von Bibliotheken mit angepassten Tags, die von den JSP-Seiten verwendet werden.

JSP-Seiten

In früheren Java-basierten Webanwendungen wurden zum Generieren des HTML-Codes, der zurück an die Web-Browser der Benutzer übertragen wurde, nur Servlets verwendet. Nach und nach wurden dann Schablonenmechanismen eingeführt, die es den Webentwicklern ermöglichten, unter Verwendung von Schablonen zum Generieren des HTML-Codes dynamische Inhalte zu generieren. Es stehen zwar zahlreiche dieser Schablonensysteme zur Verfügung, trotzdem hat sich für die J2EE-Architektur zur Anzeige von Inhalten die Verwendung von JSP-Seiten (JavaServer pages) durchgesetzt.

Wenn eine J2EE-Anwendung eine Anforderung vom Browser eines Benutzers empfängt, wird diese Anforderung zunächst dahingehend verarbeitet, Parameter aus der Anforderung zu extrahieren und eine durch die Anforderung initialisierte Geschäftslogik auszuführen. Sobald die Verarbeitung abgeschlossen ist, muss die Webanwendung die Anforderung einer JSP-Seite zuteilen: dazu wird der *Anforderungsdispatcher* verwendet. Gewöhnlich wird im Servletkontext ein Anforderungsdispatcher aufgerufen, indem die Ziel-JSP-Seite an den Dispatcher übergeben wird und anschließend die Anforderungs- und Antwortobjekte vom Anforderungsdispatcher *weitergeleitet* werden.

- Eine JSP-Seite setzt sich aus einer Kombination aus HTML-Code, JSP-Tags und Scriptelementen wie *Scriptlets* zusammen.
- HTML: Eine JSP-Seite kann jeden denkbaren Umfang an normalem HTML-Code enthalten. Dieser Inhalt wird ohne Änderungen direkt an die betreffende Browserseite weitergeleitet.
- JSP-Tags: Über Tags wird der dynamisch generierte HTML-Code mit Werten aufgefüllt, die beim Generieren der Seite berechnet werden. Es gibt Standard-JSP-Tags wie <jsp:getProperty>, <jsp:include> und <jsp:forward>. Diese Tags stehen grundsätzlich zum Erstellen einer JSP-Seite zur Verfügung. Zusätzlich können Sie angeben, dass Ihre Webanwendung eine oder mehrere Bibliotheken mit angepassten Tags verwenden soll. Jede dieser Bibliotheken mit angepassten Tags muss für die Webanwendung in der **web.xml**-Datei deklariert

werden. In der Deklaration muss sowohl die URI für die Tagbibliothek als auch die Position der TLD-Datei (Tag Library Descriptor, Tagbibliothekdeskriptor) angegeben werden.

Anmerkung: In Visual Modeler wird die Verwendung von Tagbibliotheken nicht weiter unterstützt. Unter Leistungsaspekten wird die Verwendung von Scriptlets empfohlen. JSP-Tags können aber weiterhin in einigen bereits vorhandenen Anwendungen oder in speziellen Integrationstasks verwendet werden.

- Scriptelemente: Sie können die HTML- und JSP-Tags auf einer JSP-Seite mit Java-Code zwischen dem Scriptlet-Anfangstag <% (oder <jsp:scriptlet>) und dem Scriptlet-Abschlusstag %> (oder </jsp:scriptlet>) einfließen lassen. Scriptlets werden gewöhnlich dazu verwendet, komplexe Ablaufsteuerungen auf einer JSP-Seite zu betreiben. Berücksichtigen Sie, dass die meisten JSP-Scriptelemente auch über eine entsprechende Kurzform (siehe Tabelle) aufgerufen werden können:

Kurzform	**XML-Form**
<%	<jsp:scriptlet>
<%=	<jsp:expression>
<%!	<jsp:declaration>
<%@	<jsp:directive>

.

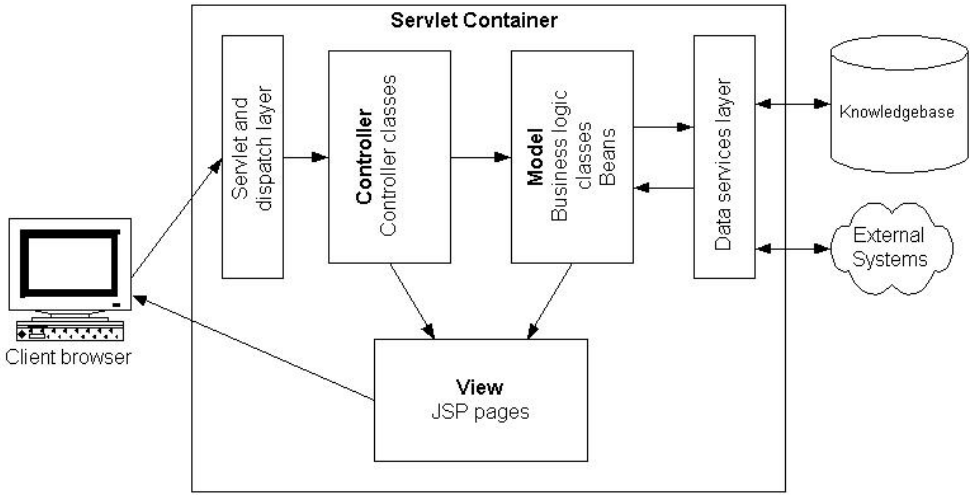
Daten werden über eine Reihe von Mechanismen an eine JSP-Seite übergeben. Wichtig in diesem Zusammenhang sind implizite Objekte und Beans.

- Implizite Objekte: Mit jeder JSP-Seite werden den Webentwicklern Objekte bereitgestellt, die für die Anzeige von Daten auf der generierten HTML-Seite verwendet werden können. Die wichtigsten dieser Objekte sind Seiten-, Anforderungs-, Sitzungs-, Konfigurations- und Anwendungsobjekte.
- Beans: Die Mehrzahl der durch die Geschäftslogik der Anwendung generierten Daten wird zur JSP-Seite hinzugefügt, indem Java-Beans zu einem der oben aufgeführten impliziten Objekte hinzugefügt werden.

Model 2-Architektur

Visual Modeler ist so gestaltet, dass es der Sun “Model 2”-Architektur entspricht. In dieser Architektur wird die Funktionalität der Webanwendung durch die drei funktionalen Komponenten Modell, Ansicht, Controller (Model, View, Controller, MVC) in logisch voneinander zu unterscheidende Komponenten aufgeteilt.

In der folgenden Abbildung wird die Architektur des Modells gezeigt:



- *Modell*: Mit dieser Komponente werden die vom System verwendeten Daten und Geschäftsobjekte verwaltet.
- *Ansicht*: Diese Komponente ist für die Generierung des dem Benutzer angezeigten Inhalts verantwortlich.
- *Controller*: Mit dieser Komponente wird der logische Datenfluss der Anwendung gesteuert. Dabei wird festgelegt, welche Aktionen für das Modell ausgeführt werden und die Kommunikation zwischen den Komponenten Modell und Ansicht wird gesteuert.

Controller

In der "Model 2"-Architektur sind Controller Java-Klassen, die dafür vorgesehen sind, die Verarbeitung einer ankommenden Anforderung zu steuern und die Anforderung dann an die entsprechende JSP-Seite weiterzuleiten. Die Basisstruktur eines Controllers in Visual Modeler hat folgendes Aussehen:

```
public class GenericController extends Controller
{
   public void execute() throws Exception
   {
      //Dispatch some business logic
      BizObjs resultBizObjects = calculate();
      //Generate the beans
      Vector beans = generateBeans(resultBizObjs);
      //Attach the beans to the request
      attachBeans(beans);
      // Dispatch to JSP page
      String pageName = choosePageLogic();
      // Dispatch to JSP page
      Dispatcher rd = request.getDispatcher(pageName);
      rd.forward(request, response);
   }

   protected BizObjs calculate() throws Exception
   {
      //do some processing
      return resultBizObjs;
   }

   protected Vector generateBeans(BizObjs bizObjs)
   {
      //create beans from business objects
      return beans;
   }
```

```
   protected void attachBeans(Vector beans)
   {
      Iterator it = beans.iterator();
      while (it.hasNext())
      {
         DataBean bean= (DataBean) it.next();
         request.setAttribute (beanName, bean);
      }
   }

   protected String choosePageLogic()
   {
      //logic to determine where to forward the request
      return pageString;
   }
}
```

Modell

In der "Model 2"-Architektur werden die Objekte, über die die Daten im System dargestellt werden, mithilfe der Modellkomponente verwaltet. Dabei ist es üblich, Geschäftsobjekte und Beans auf den JSP-Seiten voneinander zu unterscheiden.

Sobald die Geschäftslogik die Erstellung und Transformation der Geschäftsobjekte abgeschlossen hat, werden die Geschäftsobjekte durch die Controllerklasse in die entsprechenden Beans umgewandelt. Anschließend werden die Beans zwecks Darstellung an die entsprechende JSP-Seite weitergeleitet.

Ansicht

Die Benutzerschnittstelle der Webanwendung wird dem Browser über JSP-Seiten bereitgestellt. Daten werden in Gestalt von Beans an die einzelnen JSP-Seiten weitergeleitet. Dies sind Klassen mit definierten Zugriffsmethoden, durch die es der Logik auf der JSP-Seite möglich ist, Werte über Tags in allgemeiner Form abzurufen:

```
<%
DataBean dataBean = request.getAttribute("nameOfBean");
String stringProperty =
dataBean.getNamedProperty("nameOfProperty");
%>
```

Beachten Sie, dass es möglich ist, mit einer Kombination aus Scriptlets, einfachen JSP-Tags und fortschrittlicheren angepassten Tags das Seitenlayout und die Anzeige der Daten zu steuern.

Weitere Literatur

Die zu Webanwendungen, J2EE, Servlets und JSP-Seiten veröffentlichte Literatur ist umfangreich. Bei den folgenden Titeln handelt es sich um empfohlene Bücher zur Vertiefung:

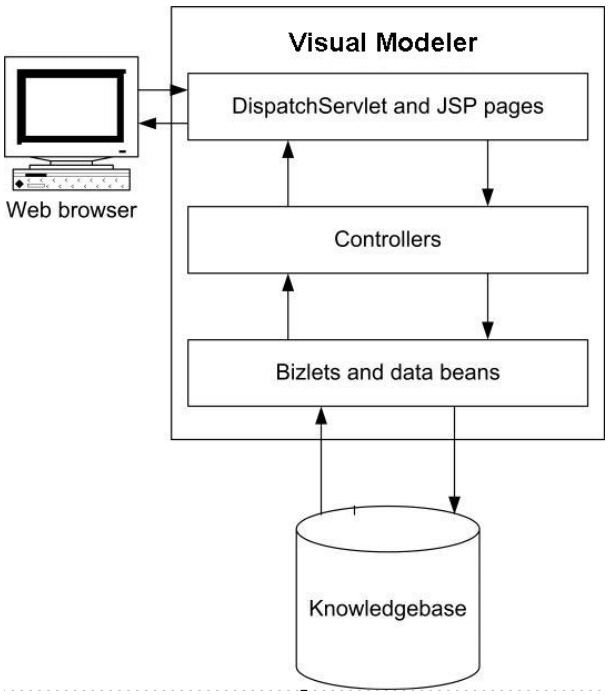
- Hall, *Core Servlets and JavaServer Pages*, Second Edition, Prentice Hall, 2003
- Hunter, *Java Servlet Programming*, Second Edition, O'Reilly, 2001
- Fields and Kolb, *Web Development with JavaServer Pages*, Second Edition, Manning, 2001

Kapitel 6. Systemarchitektur

In diesem Abschnitt wird die Architektur von Visual Modeler beschrieben. Außerdem werden einige der wichtigen Java-Klassen vorgestellt, die von Visual Modeler und seinen Anwendungen verwendet werden. Diesem Abschnitt liegt die Annahme zugrunde, dass Sie über ein profundes Wissen hinsichtlich der J2EE-Architektur verfügen.

Dieser Abschnitt ist dafür vorgesehen, Sie beim Ändern oder Erweitern bereits vorhandener Anwendungen oder beim Schreiben neuer Anwendungen zu unterstützen. Denken Sie daran, dass nicht alle Teile von Visual Modeler dieser Architekturbeschreibung entsprechen.

In der folgenden Abbildung wird die Architektur von Visual Modeler gezeigt.



Visual Modeler - Webanwendung

Wenn Sie Visual Modeler in Ihrem Servlet-Container installieren, erfolgt diese Installation als WAR-Datei **Sterling.war**. Beim Implementieren der WAR-Datei wird diese in ein Verzeichnis mit dem Titel **Sterling/** extrahiert. Die **web.xml**-Datei für die Anwendung finden Sie im Unterverzeichnis **WEB-INF/**.

Die wichtigsten Konfigurationseinstellungen in dieser Datei sind:

- Die Definition der Klassen "InitServlet" und "DispatchServlet":
  - "InitServlet" wird geladen, wenn der Servlet-Container gestartet wird. "InitServlet" sucht in allen Konfigurationsdaten nach Visual Modeler und verwendet den Wert des Elements "propertiesFile": standardmäßig lautet dieser **Comergent.xml**.

  - "DispatchServlet" ist das zum Verarbeiten eingehender Anforderungen verwendete Hauptservlet. Die Mehrzahl der im Servletzuordnungsabschnitt definierten URLs werden in "DispatchServlet" aufgelöst.
- Der Servletzuordnungsabschnitt ordnet die meisten URL-Muster dem Servlet "DispatchServlet" zu. Beachten Sie, dass über “/msg/*” Anforderungen der Klasse "MessagingServlet" zugeordnet werden. So ist sichergestellt, dass eingehende XML-Nachrichten durch diese Servletklasse verarbeitet werden.
- Im Sitzungskonfigurationselement wird ein Sitzungszeitlimitwert von 30 (Minuten) definiert. Für jede Implementierung von Visual Modeler muss ein passender Wert für diesen Parameter definiert werden. Dabei müssen folgende Aspekte berücksichtigt werden:
  - Browser werden möglicherweise unbeaufsichtigt zurückgelassen, wenn die Systembenutzer ihren Platz verlassen. Kann ein skrupelloser Benutzer bei weiterhin aktiver Sitzung auf einen Browser zugreifen, hat er auch Zugriff auf das System.
  - Die Benutzer können im Rahmen der Verwendung von Visual Modeler auch in andere externe Systeme wechseln. Über den Wert des Sitzungszeitlimits muss ausreichend Zeit zur Verfügung gestellt werden, um in andere Systeme wechseln und dann wieder zurückkehren zu können.
  - Jede Sitzung arbeitet mit Systemressourcen. Je größer der Wert für das Sitzungszeitlimit, desto größer die Speicherbelegung durch das System.
- Die Position des "Comergent"-Tagbibliothekdeskriptors (Tag Library Descriptor, TLD) wird bereitgestellt.

Anforderungen verarbeiten

Wenn Visual Modeler eine Anforderung vom Browser eines Benutzers empfängt, muss bestimmt werden, wie die Anforderung verarbeitet werden soll und wie dem Benutzer die Ergebnisse angezeigt werden sollen. Dies geschieht mithilfe der **MessageTypes.xml**-Konfigurationsdateien. Über diese Dateien wird die Zuordnung zwischen einer Anforderung und den verwendeten Verarbeitungsklassen der Logik und den JSP-Seiten festgelegt.

1. Wird eine Anforderung empfangen, wird der Nachrichtentyp ermittelt und der passende Controller aufgerufen.
2. Möglicherweise wird mithilfe einer Geschäftslogik- oder bizAPI-Klasse zusätzliche Geschäftslogik aufgerufen.
3. Vom Controller wird die Anforderung dann zwecks Bereitstellung der Ausgabe für den Browser des Benutzers an die angegebene JSP-Seite weitergeleitet.

Über das Element "messageTypeFilename" des Elements "GeneralObjectFactory" der Datei **Comergent.xml** wird die durch Kommas begrenzte Liste der Datei **MessageTypes.xml** angegeben, die zum Festlegen des Nachrichtentyps verwendet wird. Für jede **MessageTypes.xml**-Datei wird eine (nach Nachrichtengruppen organisierte) Liste mit Nachrichtentypen deklariert.

Für jede Anforderung wird der Nachrichtentyp über den Parameter "cmd" angegeben. Bei einem URL

```
../Sterling/catalog/matrix?cmd=search
```

beispielsweise lautet der Nachrichtentyp “search”.

Jeder Nachrichtentyp wird über das Attribut "Name" des Elements "MessageType" festgelegt. Über das Attribut "Name" wird angegeben, welcher Nachrichtentyp angefordert werden soll, wenn ein Benutzer auf einen URL klickt.

Anmerkung: Sie müssen dafür sorgen, dass Nachrichtengruppen und Nachrichtentypen über eindeutige Namen verfügen. Daher müssen Sie die Gruppe der **MessageTypes.xml**-Dateien dahingehend überprüfen, ob nicht doch Nachrichtengruppen und Nachrichtentypen mit identischen Namen definiert wurden. Informationen zu einer Ausnahme von dieser Regel finden Sie unter „Definitionen des Elements "MessageType" überschreiben". Es wird empfohlen, die Nachrichtentypen innerhalb der Nachrichtengruppen alphabetisch nach Namen zu ordnen, um so schnell doppelt vergebene Nachrichtentypennamen erkennen zu können.

"MessageType"-Elemente verfügen über eines oder mehrere der folgenden untergeordneten Elemente:

- BizletMapping: Wird für die Nachrichtenverarbeitung verwendet. Dabei werden zur Verarbeitung der betreffenden Nachricht eine "Bizlet"-Klasse und eine Methode dieser Klasse zugeordnet.
- ControllerMapping: Ordnet einen Controller für die Verarbeitung der Anforderung zu. Für die Nachrichtenverarbeitung können Sie eine Klasse des Typs "BizRouter" angeben, um so eine "Bizlet"-Klasse zur Verarbeitung der Nachricht aufzurufen.
- JSPMapping: Ordnet eine JSP-Seite zur Anzeige der Ergebnisse der Verarbeitung der Anforderung zu.

In einem "MessageType"-Element kann jede beliebige Kombination dieser drei Elemente angegeben werden.

- Wird kein "ControllerMapping"-Element angegeben, wird standardmäßig die Klasse "ForwardController" verwendet. Bei dieser Klasse wird die Anforderung einfach an die im "JSPMapping"-Element angegebene JSP-Seite weitergeleitet. Wird kein "JSPMapping"-Element gefunden oder fehlt die angegebene JSP-Seite, wird eine entsprechende Fehlerseite angezeigt.
- Wird ein benutzerdefinierter Controller angegeben, kann dieser die Anforderung entweder selbst verarbeiten (vgl. „Controllerklassen" auf Seite 22) oder über die Methode *runAppJob()* der Klasse "AppExecutionEnv" (vgl. „Klasse "AppExecutionEnv"" auf Seite 27) eine entsprechende Geschäftslogikklasse aufrufen.
- Wird kein "JSPMapping"-Element angegeben, muss über die Geschäftslogikklasse oder den Controller angegeben werden, welche JSP-Seite verwendet werden soll.

Jede Anforderung oder Nachricht wird hinsichtlich der erforderlichen Berechtigungen überprüft, um so festzustellen, ob der Benutzer zur Ausführung des betreffenden Nachrichtentyps berechtigt ist. Nicht alle Benutzer dürfen jeden Nachrichtentyp ausführen.

Definitionen des Elements "MessageType" überschreiben

Das Element "MessageType" verfügt über das folgende optionale Attribut: "IsOverlay". Wenn für dieses Attribut der Wert "true" festgelegt wird, werden durch die Definition des Elements "MessageType" alle vorherigen Definitionen dieses Nachrichtentyps in einer beliebigen früheren Datei "MessageTypes.xml" im Element "messageTypeFilename" überschrieben.

Werden für ein und denselben Nachrichtentyp zwei oder mehr Definitionen angegeben, ohne dass der eine mit dem Attribut "isOverlay" versehen ist, wird ein entsprechender Initialisierungsfehler angezeigt und es wird die erste Nachrichtentypdefinition verwendet.

Beachten Sie, dass die Position des Elements "MessageType" durch das Attribut "IsOverlay" nicht verändert wird; diese wird weiterhin entweder durch die Nachrichtengruppe bestimmt, zu der die erste Definition gehört oder durch das Element "MessageTypeRef", durch das Bezug auf den Nachrichtentyp genommen wird.

Möchten Sie beispielsweise die Definition des Nachrichtentyps "adirectLogin" überschreiben, können Sie wie folgt ein Element definieren:

```
<MessageType Name="adirectLogin" IsOverlay="true">
<ControllerMapping>
com.comergent.apps.common.controller.MyLoginController
</ControllerMapping>
<JSPMapping>../common/adirectPageLoader.jsp</JSPMapping>
</MessageType>
```

Sie können das Attribut "IsOverlay" auch für "MessageGroup"-Deklarationen verwenden, um die Definition einer Nachrichtengruppe zu überschreiben. Allerdings wird davon abgeraten.

Standardelemente

Für jede Nachrichtengruppe können die Standardelemente "BizletMapping", "ControllerMapping" und "JSPMapping" angegeben werden. Diese Elemente kommen zum Einsatz, wenn für einen zu der betreffenden Nachrichtengruppe gehörenden Nachrichtentyp keine Zuordnung angegeben ist.

Ist in einer Nachrichtengruppe keine Standardzuordnung angegeben, sucht das System gewöhnlich in der übergeordneten Nachrichtengruppe der aktuellen Nachrichtengruppe nach einer Standardzuordnung. Wird in der Nachrichtengruppenbaumstruktur keinerlei Zuordnung gefunden, werden die Werte aus der Nachrichtengruppe "MessageGroupDefaults" verwendet.

Wichtige Java-Klassen

Schematisch betrachtet verfügen alle Visual Modeler-Anwendungen über dieselbe Struktur: sie setzen sich aus Controllern, Geschäftsobjekten und Bizlets sowie JSP-Seiten zusammen.

Wrapperklassen

Viele der in J2EE-Webanwendungen verwendeten Standardklassen wurden in Wrapperklassen eingeschlossen, um so mit untergeordneten Eigenheiten zwischen den unterstützten Servlet-Containern umgehen zu können:

ComergentContext

Diese Klasse wird dazu verwendet, den Servlet-Containerkontext mit einzuschließen. Mithilfe dieser Klasse können Sie das Objekt "Env" für Informationen zur Umgebung abrufen. Beachten Sie, dass jedes definierte Kontextattribut serialisierbar sein muss. Wenn Sie versuchen, ein nicht serialisierbares Attribut zu definieren, kommt es zu einer Ausnahmebedingung.

Bereitgestellt wird die Methode *getResourceAsStream()*: Mit dieser Methode kann auf eine Datei als Datenstrom im Lesezugriff zugegriffen werden. Für den Schreibzugriff auf eine Datei müssen Sie die Methode *adjustFileName()* der Klasse "LegacyFileUtils" verwenden.

ComergentDispatcher

Bei dieser Klasse handelt es sich um einen einfachen Wrapper der standardmäßigen "RequestDispatcher"-Klasse: über sie werden die Methoden *forward()* und *include()* bereitgestellt.

ComergentRequest

In dieser Klasse ist die standardmäßige Klasse "HttpRequest" eingeschlossen. In ihr werden Unterstützungsmethoden zum Auswerten der ankommenden Anforderungen und Nachrichten bereitgestellt.

ComergentResponse

Mit dieser Klasse wird die standardmäßige "HttpResponse"-Klasse mit eingeschlossen. Dabei wird eine *localRedirect()*-Methode bereitgestellt, über die eine Anforderung mit einem neuen Nachrichtentyp übergeben werden kann. Wenn Sie beispielsweise für die Verarbeitung einer Anforderung einen Controller wünschen und das Ergebnis dann an einen anderen Controller übertragen möchten, gehen Sie wie folgt vor:

```
response.localRedirect(request, "messageType");
```

Dies hat zur Folge, dass die Anforderung an die Klasse "DispatchServlet" übergeben wird, als ob sie als HTTP-Anforderung empfangen worden wäre.

ComergentSession

Mit dieser Klasse wird die standardmäßige "HttpSession"-Klasse mit eingeschlossen. Wenn sich ein Benutzer zum ersten Mal anmeldet, wird ein entsprechendes Benutzerdaten-Bean erstellt und zum Objekt "ComergentSession" hinzugefügt. Sie können über die "ComergentSession"-Methode *getUser()* auf Benutzerinformationen zugreifen.

Beispiel: Über

```
session.getUser().getUserKey()
```

wird der Schlüssel des aktuellen Benutzers geliefert, über

```
session.getUser().getPartnerKey()
der Schlüssel des Partners, zu dem der betreffende Benutzer gehört.
```

Das Objekt "ComergentSession" wird zum Speichern von Informationen verwendet, die in Bezug auf mehrere Anforderungen im Rahmen einer Benutzersitzung permanent sein müssen. Verwenden Sie zum Definieren eines Objekts in der Sitzung die Methode *setAttribute(String s, Object o)* und zum Abrufen *getSession(String s)*. Die in der Sitzung gespeicherten Objekte müssen die serialisierbare Schnittstelle implementieren: Alle generierten Data-Beans implementieren diese Schnittstelle und können daher in der Sitzung gespeichert werden.

Die Klasse "ComergentSession" bietet auch eine Methode *logout()*: bei Aufruf dieser Methode wird die Servlet-Container-Sitzung umgehend inaktiviert.

Servlets

Die wichtigsten verwendeten Servlets sind:

- InitServlet: Dieses Servlet wird geladen, wenn der Servlet-Container gestartet wird. Mit seiner Methode *init(ServletConfig config)* wird die Klasse "ComergentAppEnv" initialisiert.
- DispatchServlet: Dieses Servlet wird dazu verwendet, fast alle von Visual Modeler verarbeiteten Anforderungen zu bedienen. Sein wichtigster Methodenaufruf lautet:

  ```
  void dispatch(HttpServletRequest request, HttpServletResponse response)
  This method creates a controller to handle the request with:
  Controller controller createController(ComergentRequest comergentRequest)
  and then invokes:
  controller.init(comergentContext, comergentSession,
  comergentRequest, comergentResponse);
  controller.execute();
  ```

  Beachten Sie, dass es sich bei der über die Methode *createController()* erstellten Instanz der Controllerklasse um eine Funktion der Anforderung handelt. Über den Anforderungsnachrichtentyp wird die Controllerklasse bestimmt, denn der Controller wird über die Klasse "GeneralObjectFactory" erstellt. Die Klasse "GeneralObjectFactory" bedient sich der Datei **MessageTypes.xml**, wenn es darum geht, eine Zuordnung zwischen Anforderungsnachrichtentyp und Controllerklasse vorzunehmen.
- DebsDispatchServlet: Dieses Servlet wird dazu verwendet, XML-Nachrichten zu verarbeiten, die aus einem anderen System an Visual Modeler übergeben wurden. Wenn der Inhaltstyp der Anforderung mit “application/x-icc-xml” oder “text/xml” beginnt, wird zum Verarbeiten der Anforderung die Klasse "MessagingController" aufgerufen.

Controllerklassen

Visual Modeler bietet zwei verschiedene Möglichkeiten, Controller zum Verarbeiten von Anforderungen zu verwenden:

Benutzerdefinierte Controller

Sie können Ihre eigene Controllerklasse definieren, indem Sie die Klasse "com.comergent.dcm.caf.controller.Controller" entsprechend erweitern. In diesem Fall müssen Sie die Anwendungslogik bereitstellen, um die JSP-Seite zu ermitteln, an die die Anforderung weitergeleitet werden soll. Beispiel:

```
boolean processingSuccess = false;
/*
*
* Business logic processes request and sets processingSuccess to
* true if successful.
*/
if (processingSuccess)
{
callJSP("SuccessMessageType");
}
else
{
callJSP("FailureMessageType");
```

```
}
protected void callJSP(String messageType) throws
ControllerException, ICCException, IOException
{
String resource = getJSPName(messageType);
ComergentDispatcher rd =
request.getComergentDispatcher(resource);
rd.forward(request, response);
}
protected String getJSPName(String messageType) throws ICCException
{
JSPObjectID id = new JSPObjectID(messageType);
return GeneralObjectFactory.getGeneralObjectFactory().-
getMapping(id);
}
```

SimpleController

Sie können die Klasse "SimpleController" dahingehend erweitern, dass die Anforderung verarbeitet wird, wenn nur ein Punkt zum Ausstieg aus der Anwendungslogik vorhanden ist. Die Klasse "SimpleController" bedient sich des Nachrichtentyps aus der Anforderung, um die JSP-Seite zu ermitteln, an die die Anforderung weitergeleitet werden soll, wenn die Anwendungslogik beendet ist. Möchten Sie die Klasse "SimpleController" erweitern, müssen Sie dazu die Methode *calculate()* überschreiben.

MessagingController

Diese Klasse wird zum Verarbeiten von XML-Anforderungen (wie Preis- und Verfügbarkeitsanforderungen oder Übertragungsanforderungen anderer Systeme zum Warenkorb) verwendet.

Data-Bean-Klassen

Der Zugriff auf Daten in Visual Modeler wird über Datenobjekte gesteuert: dabei handelt es sich um XML-Dokumente, in denen die Geschäftsentitäten wie Partner, Benutzer, Produkte etc. beschrieben werden. Die Beschreibung bezieht sich auf die Felder des betreffenden Datenobjekts sowie auf Informationen darüber, welche Zuordnungen zu den Datenbanktabellen in der Wissensdatenbank bestehen. Jede XML-Datei eines Datenobjekts wird dazu verwendet, eine entsprechende Data-Bean-Java-Klasse zu entsprechen.

Die "DataBean"-Klassen sind die wichtigsten Klassen zum Darstellen der verschiedenen Geschäftsentitäten in Visual Modeler. Jede Geschäftsentität wie Benutzer, Partner, Produkt etc. wird im Speicher durch eine Instanz der entsprechenden "DataBean"-Klasse abgebildet. Weitere Informationen hierzu finden Sie unter „Data-Beans in Visual Modeler” auf Seite 45. Von einigen traditionellen Anwendungen wird möglicherweise noch die Klasse "BusinessObject" verwendet, aber in der Regel ist die Verwendung der Klasse "BusinessObject" überholt.

Klassen des Typs "DataBean" werden auch dazu verwendet, Daten an JSP-Seiten zu übergeben. Sie können jede beliebige Datenobjektdefinition im XML Schema von Visual Modeler dazu verwenden, eine Klasse "DataBean" zu generieren. Dazu müssen Sie das Zielelement "generateBean" verwenden. (Weitere Einzelheiten hierzu finden Sie unter „Software-Development-Kit zum Anpassen der Visual Modeler-Implementierung verwenden” auf Seite 85.)

Bei der Klasse "DataBean" handelt es sich um eine allgemeine, abstrakte Klasse. Alle generierten Data-Bean-Klassen erweitern diese Klasse. Jede Klasse "DataBean" verfügt über die Methoden *restore()* und *persist()*, über die Daten aus der Datenbank abgerufen bzw. in der Datenbank gespeichert werden.

Einige Anwendungen arbeiten mit Application-Beans. Informationen zum Verwenden dieser Beans finden Sie unter „Application-, Entity- und Presentation-Beans” auf Seite 51.

Klassen "ObjectManager" und "OMWrapper"

Sie sollten keine "DataBean"-Klassen mithilfe ihrer Konstruktoren instanziieren. Verwenden Sie stattdessen die Klassen "ObjectManager" und "OMWrapper", wenn Sie bei Bedarf für Ihre Anwendungen neue Instanzen von Objekten erstellen müssen. Diese Klassen folgen insofern dem Factory-Muster, als sie eine Klasse bereitstellen, die dafür vorgesehen ist, bei Bedarf entsprechende Objektinstanzen zu erstellen. Mit diesen Klassen können Sie von einer Objektklasse in eine andere wechseln, ohne dazu den Anwendungscode ändern zu müssen, durch den die Objekte erstellt und verwendet werden.

Objekte erstellen

Im Allgemeinen sollte eher die Klasse "OMWrapper" und nicht die Klasse "ObjectManager" verwendet werden, doch grundsätzlich können beide Klassen verwendet werden. Mit diesen Klassen erstellen Sie mit folgenden Methoden Objekte:

```
ObjectClass temp_ObjectClass =
(ObjectClass) OMWrapper.getObject("ObjectName");
```

oder

```
ObjectManager temp_ObjectManager = ObjectManager.getInstance();
ObjectClass temp_ObjectClass =
(ObjectClass) temp_ObjectManager.getObject("ObjectName");
```

Objektklassen Objektnamen zuordnen

Bei der Festlegung, welcher Typ von Objekt mithilfe des über die Methode *getObject()* bereitgestellten Objektnamens erstellt werden soll, arbeiten die beiden Klassen "ObjectManager" und "OMWrapper" mit der Konfigurationsdatei **ObjectMap.xml** (im Verzeichnis ***debs_ausgangsverzeichnis*/Sterling/WEB-INF/properties/**).

Anmerkung: Fügen Sie keine Kommentare in die Datei **ObjectMap.xml** ein: Dadurch kann es bei der Initialisierung zu Fehlern kommen.

Jedes Element des Typs "Object" hat folgendes Format:

```
<Object ID="ObjectName">
<ClassName>ObjectClass</ClassName>
</Object>
```

Wenn die Methode *getObject("ObjectName")* aufgerufen wird, wird eine Instanz der Klasse "ObjectClass" ausgegeben. Bei *ObjectName* muss es sich um den Namen einer Java-Klasse oder -Schnittstelle und bei *ObjectClass* um eine Unterklasse der

Klasse *ObjectName* (möglicherweise sie selbst) oder um eine Klasse handeln, durch die die *ObjectName*-Schnittstelle implementiert wird.

Wenn die Datei **ObjectMap.xml** nicht über ein Element des Typs "Object" verfügt, dessen Attribut "ID" dem Parameter "ObjectName" entspricht, wird von der Klasse "ObjectManager" oder "OMWrapper" eine Instanz der Klasse "ObjectName" erstellt. Mit anderen Worten: Das Verhalten entspricht der Existenz eines Elements im folgenden Format:

```
<Object ID="ObjectName">
<ClassName>ObjectName</ClassName>
</Object>
```

```
Angenommen, die Datei "ObjectMap.xml" enthält das folgende Element:

<Object ID="com.comergent.bean.productMgr.ProductBean">
<ClassName>
  com.comergent.bean.productMgr.MatrixProductBean
</ClassName>
</Object>
```

Dann wird über den folgenden Methodenaufruf eine Instanz der Klasse "MatrixProductBean" erstellt:

```
ProductBean temp_ProductBean = (ProductBean)
OMWrapper.getObject("com.comergent.bean.productMgr.ProductBean");
```

Beachten Sie, dass die Klasse "MatrixProductBean" die Klasse "ProductBean" erweitern muss; ansonsten kommt es zur Laufzeit zu einer Ausnahmebedingung bei der Klassenumsetzung. Wenn allerdings kein Element existiert, dessen Attribut "ID" den Wert "com.comergent.bean.productMgr.ProductBean" aufweist, wird über denselben Aufruf eine Instanz der Klasse "com.comergent.bean.productMgr.ProductBean" ausgegeben.

Beschränkungen

Beachten Sie, dass Sie keine "Object"-Definitionen vornehmen können, in denen die im Element "ClassName" in einem Element "Object" angegebene Klasse dem Attribut "ID" in einem anderen Element "Object" entspricht. Die einzige Ausnahme von dieser Regel liegt vor, wenn die Klasse für ein einzelnes Element "Object" als Wert für das Attribut "ID" und das Element "ClassName" verwendet wird. Insbesondere gilt bei der Erweiterung eines Datenobjekts (siehe „Datenobjekte erweitern” auf Seite 58) Folgendes:

1. Definieren Sie ein Element "Object", das die erweiterte Klasse der erweiternden Klasse zuordnet:

   ```
   <Object ID="<Extended class>">
   <ClassName><Extending class></ClassName>
   </Object>
   ```

2. Stellen Sie sicher, dass in allen "ClassName"-Elementen Bezüge auf das erweiterte Datenobjekt durch Bezüge auf das erweiternde Datenobjekt ersetzt werden.

Parameter übergeben

Wenn Sie Parameter an die Objektkonstruktoren übergeben müssen, steht Ihnen dafür die folgende "OMWrapper"-Methode zur Verfügung:

```
ObjectClass temp_ObjectClass = (ObjectClass)
OMWrapper.getObjectArg("ObjectName", Object arg1, ... ,
Object arg10);
```

In diesem Format können Sie bis zu zehn Parameter als Objekte an den Methodenaufruf übergeben. Mit den folgenden "OMWrapper"- und "ObjectManager"-Methodenaufrufen können Sie eine unbegrenzte Zahl an Parametern als Array von Objekten übergeben:

```
ObjectClass temp_ObjectClass = (ObjectClass)
OMWrapper.getObject("ObjectName", Object[] args);
```

oder

```
ObjectClass temp_ObjectClass = (ObjectClass)
temp_ObjectManager.getObject("ObjectName", Object[] args);
```

Nehmen Sie beispielsweise an, die Datei "ObjectMap.xml" enthält folgendes Element:

```
<Object ID="com.comergent.bean.productMgr.OrderBean">
<ClassName>com.comergent.bean.matrix.MatrixOrderBean</ClassName>
</Object>
```

Hier ist die Klasse "MatrixOrderBean" eine Unterklasse der Klasse "OrderBean". Angenommen, die Klasse "MatrixOrderBean" hat einen Konstruktor im Format "MatrixOrderBean(CartBean cb)".

In diesem Fall wird mithilfe des folgenden Methodenaufrufs eine Instanz der Klasse "OrderBean" erstellt, wobei eine Instanz der Klasse "CartBean" als Parameter dient:

```
Cart temp_CartBean = (CartBean)
OMWrapper.getObject("com.comergent.bean.partnerMkt.CartBean");
/*
Code that processes the cart bean object
*/
OrderBean temp_OrderBean = (OrderBean)
OMWrapper.getObjectArg("com.comergent.bean.productMgr.OrderBean",
temp_CartBean);
```

Objektpool verwenden

Wenn Sie davon ausgehen, dass einige Objektklassen erstellt und häufig verwendet werden, können Sie mithilfe der Klassen "ObjectManager" und "OMWrapper" einen Pool mit Objekten erstellen. Durch das übergeordnete Objekt (gekennzeichnet durch das Attribut "ID") muss die poolfähige Schnittstelle implementiert werden. Diese Schnittstelle ist Teil des Pakets "com.comergent.dcm.objmgr". Darin wird eine zu implementierende Methode "reset()" deklariert.

Wenn Sie ein bestimmtes poolfähiges Objekt nicht mehr benötigen, können Sie es an den Objektpool zurückgeben. Verwenden Sie dazu wie folgt die Methode *return()*:

1. Definieren Sie in dem Eintrag in der Datei **ObjectMap.xml** für eine im Pool zu verwendende Klasse für das Attribut "MaxPoolSize" die Anzahl der Objekte, die im Pool erstellt werden sollen:

```
<Object ID="ObjectName" MaxPoolSize="n">
<ClassName>ObjectClass</ClassName>
</Object>
```

2. Erstellen Sie (wie oben beschrieben) mithilfe der Klassen "ObjectManager" und "OMWrapper" Instanzen der Objektklasse.
3. Wenn Sie das betreffende Objekt nicht mehr benötigen, geben Sie die Instanz wie folgt an den Pool zurück:

   ```
   OMWrapper.return(temp_ObjectClass);
   ```

4. oder

   ```
   temp_ObjectManager.return(temp_ObjectClass);
   ```

Beachten Sie, dass bei Erstellung eines Objekts durch Übergeben von Parametern entsprechend der Beschreibung unter „Parameter übergeben ” auf Seite 25 anstelle einer erneuten Verwendung eines Objekts aus dem Pool ein neues Objekt erstellt wird.

Klasse "AppExecutionEnv"

Sie können die Klasse "AppExecutionEnv" zum Ausführen von Geschäftslogikklassen verwenden. Allerdings ist die Verwendung von Geschäftslogikklassen veraltet, daher ist diese Klasse nur zur Unterstützung von traditionellen Anwendungen zu verwenden. Mithilfe der statischen Methode *runAppObj()* können Sie die Erstellung einer Geschäftslogikklasse und die Ausführung des entsprechenden Prologs und der Servicemethoden initiieren.

Das gängigste zu verwendende Format lautet:

```
AppExecutionEnv.runAppObj(String messageType, BizObjTable bizObjects)
```

Durch die Klasse "AppExecutionEnv" wird die Geschäftslogikklasse aufgerufen, die über die Zeichenfolge "messageType" festgelegt wird und die den Vektor "BizObjTable" für Geschäftsobjekte als Eingabegeschäftsobjekte verwendet.

Klasse "AppsLookupHelper"

Es gibt zahlreiche Situationen in Visual Modeler, in denen der Status eines Datenobjekts mithilfe eines Suchcodes verwaltet wird. Beispielsweise kann sich der Status eines Auftrags im Verlaufe der Auftragserteilung mehrmals ändern. Außerdem gibt es zahlreiche Beispiele für Anzeigefelder wie den "Titel" eines Benutzers, für die verschiedene sinnvoll definierte Werte verwendet werden können und die hinsichtlich unterschiedlicher Ländereinstellungen entsprechend gesteuert werden müssen. Diese Daten sind in der Tabelle "CMGT_LOOKUPS" des Wissensdatenbankschemas gespeichert.

Für jeden Suchtyp kann es einen oder mehrere Suchcodes geben und jeder dieser Suchcodes verfügt über eine entsprechende beschreibende Zeichenfolge. Beispiel:

Suchtyp	Suchcode	Beschreibung
AddressType	10	Rechnungsstellung
AddressType	20	Versand

Über die Klasse "AppsLookupHelper" können Sie einer Beschreibung einen Suchcode zuordnen. Durch Aufrufen der passenden Methode der Klasse "AppsLookupHelper" übergeben Sie den Suchcode als Parameter, woraufhin das

entsprechende Element "String" zurückgegeben wird. Je nachdem, an welchem Suchtyp Sie interessiert sind, müssen Sie die entsprechende Methode für diesen Suchtyp auswählen. Über die verwendete Methode wird festgelegt, welcher Suchtyp zum Abrufen des Suchcodes aus der Tabelle "CMGT_LOOKUPS" zu verwenden ist. Möchten Sie beispielsweise die Codezeichenfolge für den Status eines Auftrags abrufen, können Sie wie folgt vorgehen:

```
String orderStatusString =
AppsLookupHelper.getOrderStatusForCode(orderStatusCode);
```

Umgekehrt können Sie den Suchcode auch wie folgt abrufen:

```
int orderStatusCode =
AppsLookupHelper.getCodeForOrderStatus(orderStatusString);
```

Für die meisten (aber nicht alle) Suchtypen wurden Unterstützungsmethoden definiert. Details zur Klasse "AppsLookupHelper" finden Sie in der Java-Dokumentation. Weitere Informationen finden Sie unter „Unterstützung für Suchcodes” auf Seite 32.

Klasse "ComergentAppEnv"

Verwenden Sie die Klasse "ComergentAppEnv", um Ihren Code mit Informationen zur Umgebung zu versehen, die sich auf Ihre Anwendung beziehen. Über diese Klasse werden die folgenden nützlichen Methoden bereitgestellt:

- *adjustFileName()*: Diese Methode ist in die Klasse "LegacyFileUtils" verlegt worden. Siehe „Klasse "LegacyFileUtils"” auf Seite 29.
- *constructExternalURL()*: Verwenden Sie diese Methode zum Erstellen eines URL, mit dessen Hilfe ein Client zurück zum Server geleitet werden kann. In erster Linie wird diese Methode dazu verwendet, einen URL zum Umleiten zu generieren, mit dessen Hilfe der Server Sitzungsdaten wiederherstellen kann.
- *getEnv()*: Mit dieser Methode wird das Umgebungsobjekt zurückgegeben.
- *getContext()*: Mit dieser Methode wird der Anwendungskontext zurückgegeben.

Klasse "Global"

Die Verwendung dieser Klasse wird nicht weiter unterstützt. Seine Protokollierungsfunktion wurde durch die log4j-API ersetzt. Weitere Informationen hierzu finden Sie unter „In Visual Modeler protokollieren: Eine Übersicht” auf Seite 69. Seine Unterstützung beim Abrufen der Werte von Eigenschaften wurde durch den Mechanismus "Preferences" ersetzt. Wenn Sie trotzdem weiterhin Code verwenden müssen, der mit der Klasse "Global" arbeitet, ersetzen Sie diese jeweils durch die Klasse "LegacyPreferences".

Schnittstelle "GlobalCache"

Verwenden Sie diese Schnittstelle, um einen Cache zu definieren, über den der Zugriff auf im Cache befindliche Objekte möglich ist, die von allen Visual Modeler-Anwendungen verwendet werden. Diese Schnittstelle kann dazu verwendet werden, eine Clusterumgebung zu unterstützen, in der Visual Modeler auf mehreren Maschinen ausgeführt wird.

Damit Sie eine Cacheklasse verwenden können, durch die die Klasse "GlobalCache"implementiert wird, müssen Sie die Methoden der Schnittstelle implementieren. Die Cacheklasse wird geladen, wenn die Methode *init()* der Klasse "InitServlet" aufgerufen wird. Sie müssen den Namen der Klasse als Element

"General.globalCacheImplClass" der Datei **Comergent.xml** bereitstellen. Folgende Standardimplementierung wird mit Visual Modeler bereitgestellt: "com.comergent.dcm.cache.impl.AppContextCache".

Sie können wie folgt auf die Implementierung der Schnittstelle "GlobalCache" zugreifen:

```
GlobalCache globalCache = GlobalCacheManager.getGlobalCache();
```

Die Schnittstelle bietet Unterstützung für folgende Methoden:

- *public String store(Serializable entry)*: Damit wird ein Objekt im globalen Cache gespeichert, wo es verbleibt, bis der Cache von der Anwendung bereinigt wird.
- *public boolean store(String id, Serializable entry)*: Damit wird ein Objekt im globalen Cache gespeichert, wo es verbleibt, bis der Cache von der Anwendung bereinigt wird.
- *public String cache(Serializable entry)*: Damit wird ein Objekt im globalen Cache gespeichert. Das Objekt bleibt solange verfügbar, wie es von der Anwendung verwendet wird, wird dann aber vom Cachesystem automatisch bereinigt.
- public String cache(Serializable entry, long lease)
- public boolean cache(String id, Serializable entry)
- public boolean cache(String id, Serializable entry, long lease)
- *public boolean contains(String id)*: Damit wird geprüft, ob der Cache das spezielle Objekt enthält.
- *public Object get(String id)*: Damit wird das im Cache ablegbare Objekt abgerufen.
- *public Object remove(String id)*: Damit wird ein im Cache ablegbares Objekt entfernt.
- *public boolean gc()*: Diese Methode dient der Verwendung durch einen Cron-Job, in dessen Rahmen der Cache von unbenutzten Einträgen befreit werden kann.

Klasse "LegacyFileUtils"

Die Klasse "LegacyFileUtils" bietet Unterstützungsmethoden für den Umgang mit Dateien. Die Verwendung dieser Klasse ist zwar veraltet, doch bietet sie Unterstützung für Methoden, die bisher von der Klasse "ComergentAppEnv" bereitgestellt wurden:

- *adjustFileName()*: Damit wird der echte Pfadname einer Datei ausgegeben. Verwenden Sie diese Methode, um im Lese- oder Schreibmodus auf Dateien zuzugreifen. Verzichten Sie auf die Verwendung der Methode *getRealPath()*, da es dabei zur Ausgabe des Werts null kommen kann. In einer Clusterumgebung wird durch die Methode *adjustFileName()* sichergestellt, dass von allen Mitgliedern des Clusters auf dieselbe Datei zugegriffen wird. Sie müssen diese Methode mit vier Parametern verwenden:

  ```
  adjustFileName(String fileName, boolean share, boolean xPublic,
  boolean xLoadable);
  ```

  Die Verwendung des Formats dieser Methode mit nur einem Parameter wird nicht weiter unterstützt. Die booleschen Parameter werden dazu verwendet, unter Verwendung der Konfigurationsparameter im Element "WritableDirectory" der Datei **web.xml** die Position der Datei festzulegen.

Klasse "OutOfBandHelper"

Mit der Klasse "OutOfBandHelper" ist es möglich, unter Verwendung einer JSP-Seite als Schablone einen Ausgabedatenstrom zu generieren. Ein Beispiel für die Verwendung dieser Klasse sehen Sie hier:

```
ComergentRequest request = ComergentAppEnv.getRequest();
ComergentResponse response = ComergentAppEnv.getResponse();
ByteArrayOutputStream stream = new ByteArrayOutputStream();
OutOfBandHelper outOfBandHelper = new OutOfBandHelper(request,
response, stream);
outOfBandHelper.getRequest().setAttribute(
ComergentRequest.COMERGENT_SESSION_ATTR,
request.getComergentSession());
outOfBandHelper.callJSP(messageType);
/*
* Initialize SendSMTP and use the stream to to set the body of the
* message
*/
String mimeType = "text/html";
String smtpHost = Global.getString(
"C3_Commerce_Manager.SMTP.SMTPHost");
SendSMTP smtp = new SendSMTP(smtpHost);
StringBuffer sb = new StringBuffer(subject);
String message = null;
String enc = ComergentI18N.getComergentEncoding();
   message = stream.toString(enc);
//Send the mail
smtp.send( from, to, cc, subject, message, mimeType);
```

In diesem Beispiel können Sie erkennen, wie die Klasse "OutOfBandHelper" unter Verwendung der vorhandenen Anforderungs- und Antwortobjekte und eines Ausgabedatenstroms initialisiert wird. Durch die dazugehörige Methode *callJSP()* wird der Ausgabedatenstrom generiert, indem die Anforderungs- und Antwortobjekte an die (durch den Nachrichtentypparameter bestimmte) JSP-Seite weitergeleitet werden. Der Ausgabedatenstrom kann dann von der Anwendung zum Abrufen des Inhalts verwendet werden.

Die Klasse "OutOfBandHelper" arbeitet beim Zuordnen eines Nachrichtentyps zu einer JSP-Seite mit entsprechenden Sitzungs- und Kontextinformationen. Dementsprechend können Sie unterschiedliche JSP-Seiten für unterschiedliche Ländereinstellungen verwenden und dabei genauso wie beim Verarbeiten von Browseranforderungen verfahren. Von der Klasse "OutOfBandHelper" wird dann ermittelt, welche JSP-Seite welcher Ländereinstellung zu verwenden ist. Dabei wird die entsprechende Ausfallsicherungslogik verwendet.

Klasse "Preferences"

Im Modul "Preferences" wird der Mechanismus für den Zugriff auf die Visual Modeler-Eigenschaften bereitgestellt. Dabei handelt es sich um eines der Plattformmodule. Weitere Informationen dazu finden Sie unter „Preferences Service” auf Seite 41. Die grundlegende Verwendung der "Preferences"-API läuft wie folgt ab:

```
private static Preferences temp_Preferences =
Preferences.getPreferences();
```

```
String temp_MyPropertyString =
temp_Preferences.getString("MyProperty");
```

Die wichtigsten von der API unterstützten Methoden zum Abrufen von Eigenschaften lauten:

- public String getString(String key, String def)
- public boolean getBoolean(String key, boolean def)
- public double getDouble(String key, double def)
- public float getFloat(String key, float def)
- public int getInt(String key, int def)
- public long getLong(String key, long def)

Für jede *get*Type*()*-Methode gibt es entsprechende *put*Type*()*-Methoden. Beispiel:

- public void putString(String key, String value)

Wenn Sie die Methode *getPreferences()* ohne einen Parameter aufrufen, rufen Sie das von Visual Modeler unterstützte Singleton-"Preferences"-Objekt ab. Wenn Sie den Namen einer Klasse wie "getPreferences(MyClass.class)" weitergeben, wird das abgerufene Objekt bereichsorientiert behandelt. Das bedeutet: Bei den Namen der Eigenschaften, deren Werte Sie mithilfe des Objekts "Preferences" abrufen, wird der Paketpfad der Klasse dem von Ihnen angegebenen Eigenschaftsnamen als Präfix vorangestellt.

Nehmen Sie beispielsweise an, "MyClass" ist im Paket "com.comergent.myApplication" enthalten. Dann sind die folgenden Codefragmente funktional gleich:

```
private static Preferences temp_Preferences =
Preferences.getPreferences();

String temp_MyPropertyString =
temp_Preferences.getString("com.comergent.myApplication.MyProperty");
```

und

```
private static Preferences temp_Preferences =
Preferences.getPreferences(com.comergent.myApplication.MyClass.class);

String temp_MyPropertyString =
temp_Preferences.getString("MyProperty");
```

Transaktionen

Visual Modeler bietet Unterstützung für Transaktionen, d. h. für Datenbankaktionen, die eine oder mehrere atomare Operationen umfassen. Gewöhnlich können Sie mithilfe der Klasse "Transaction" Situationen steuern, in deren Rahmen mehrere Datenobjekte als persistent definiert sind, was bedeutet, dass bei Ausfall eines Objekts auch die anderen Objekte ausfallen.

Unterstützung für Suchcodes

Visual Modeler verwendet Suchcodes, mit deren Hilfe ein Mechanismus bereitgestellt wird, über den den unterschiedlichen Ländereinstellungen entsprechende Zeichenfolgen gesteuert und den Benutzern angezeigt werden können. Für jeden Suchtyp können Sie einen oder mehrere Suchcodes definieren und für jeden Suchcode können Sie für jede unterstützte Ländereinstellung eine entsprechende Zeichenfolge definieren.

Welche Unterstützung wird von Visual Modeler für die Suche bereitgestellt?

Visual Modeler ist in der Lage, automatisch Vorgänge für die Suche zwischen Codewerten und den entsprechenden Zeichenfolgen sowie zwischen Suchcodezeichenfolgen und Codewerten bereitzustellen.

Wenn der “Code” für "DsElement" definiert wird, wird die entsprechende “Zeichenfolge” automatisch aus dem Suchcache aufgefüllt. Wenn der Wert für “Zeichenfolge” definiert wird, wird mithilfe des Zeichenfolgewerts nach dem “Code” gesucht.

Werden Zeichenfolgewerte lokalisiert?

Ja. Für eine Suche von Code nach Zeichenfolge bedient sich der Mechanismus der Ländereinstellung des Benutzers, um den zu verwendenden Zeichenfolgewert zu ermitteln. Für eine Suche von Zeichenfolge nach Code bedient sich der Mechanismus der Ländereinstellung des Benutzers, um über einen Zeichenfolgewert nach dem entsprechenden Code zu suchen.

Wie wird eine Zuordnung von Code nach Zeichenfolge definiert?

Beziehungen von Code nach Zeichenfolge werden in der Schemadatei **DsDataElement.xml** definiert. Wenn dann sowohl der “Code” als auch die “Zeichenfolge” für "DsDataElements" in einem Datenobjekt verwendet wird, wird die Zuordnung von Code nach Zeichenfolge automatisch abgewickelt.

Nachfolgend sehen Sie ein Beispiel eines "DataElement"-Code-Zeichenfolge-Paars.

```
<DataElement Name="OrderStatus" Description="Order Status"
DataType="LONG" MaxLength="20" LookupType="OrderStatus"
LookupString="OrderStatusString"/>
<DataElement Name="OrderStatusString" Description="Order Status"
DataType="STRING" MaxLength="260" LookupType="OrderStatus"
LookupCode="OrderStatus"/>
```

Werden Suchvorgänge für XML-Nachrichten durchgeführt?

Ja. Wenn ein für den Nachrichtenaustausch verwendetes Datenobjekt ein Code-Zeichenfolge-Paar enthält, wird der Zeichenfolgewert automatisch für die Suche nach dem Code verwendet.

Wie wird der Suchcache geladen?

Der Suchcache wird beim Systemstart geladen.

Kapitel 7. Plattformmodularität

Einführung in die Plattformmodularität von Visual Modeler

Mit der modularen Architektur von Visual Modeler sollen Implementierungen einfacher angepasst und aktualisiert werden können. Mit der Plattformarchitektur von Visual Modeler kann die gewünschte Plattform modularer aufgebaut werden, sodass Änderungen und Upgrades für die Plattform schneller und einfacher durchgeführt und die Module zur Unterstützung der verschiedenen Produkte, die auf deren Basis aufgebaut sind, wiederverwendet werden können.

Welche Vorteile sich ergeben, wenn Plattformfunktionalität in Plattformmodulen bereitgestellt wird und Module die Aufrufe anderer Module ausschließlich über die entsprechenden externen Schnittstellenbereiche vornehmen müssen, können Sie der folgenden Auflistung entnehmen:

- Es ist einfacher, die Funktionalität von Anwendungen aufzugliedern.
- Es ist einfacher, die Abhängigkeiten zwischen den verschiedenen Teilen von Visual Modeler zu verstehen und zu steuern.
- Es ist einfacher, Anpassungen an einzelnen Modulen vorzunehmen und dabei die Auswirkungen der Änderungen in einem bestimmten Modul auf das komplette System einzuschätzen.
- Für die Module können unabhängig voneinander leichter Upgrades durchgeführt werden, wodurch sich die Effekte, die von einem Upgrade ausgehen können, minimieren lassen.
- Upgrades für Module, die nicht angepasst wurden, haben keinerlei Auswirkungen auf die Anpassungen in anderen Modulen.
- Neue Funktionen können in Form eines Moduls bereitgestellt werden, das dann in die aktuelle Implementierung von Visual Modeler übernommen wird.

Plattformmodule

Die Plattform von Visual Modeler wurde als Gruppe voneinander abhängiger Module entwickelt, die einer gemeinsamen organisatorischen Struktur entsprechen. Gewöhnlich entspricht jedes Plattformmodul einer funktionalen Komponente von Visual Modeler (z. B. einem Service) oder einer Komponente der Visual Modeler-Plattform. In den Plattformmodulen wird eine Java-API zu anderen Modulen bereitgestellt. In einigen Modulen wird außerdem eine Anzahl von “Hilfeprogramm”-Klassen bereitgestellt, die auch von anderen Modulen verwendet werden.

Gewöhnlich weist jedes Plattformmodul die folgende Struktur auf:

- Java-Klassen: Sind in den folgenden Baumstrukturen organisiert. Zur Erstellungszeit werden die Verzeichnisse für das betreffende Modul in einer einzelnen JAR-Datei zusammengefasst.
  - com.comergent.api.*modul*: Externe API-Schnittstellen: Werden von anderen Modulen für den Zugriff auf die in dem Modul bereitgestellten Funktionen verwendet. Wenn von einem Modul die Klasse eines anderen Moduls aufgerufen wird, muss dieser Aufruf gewöhnlich über die externe API des anderen Moduls erfolgen. Dafür steht das Paket "com.comergent.api" für das Modul zur Verfügung.

– com.comergent.*module*: Implementierungsklassen: Dabei geht es um die Implementierung der externen API-Schnittstellen. Wenn die externe API des Moduls von einem anderen Modul aufgerufen wird, fungieren die aktuell verwendeten Klassen als Implementierungsklassen der Modulschnittstelle. In den Implementierungspaketen können auch interne Klassen enthalten sein. Diese werden zwar von den Implementierungsklassen verwendet, sie werden aber nicht extern verfügbar gemacht und sind nicht Bestandteil des unterstützten Javadocs.

- Konfigurationsdateien (wie Eigenschaftendateien) speziell für das Modul. Diese sind für die Bereitstellung in der Klassenhierarchie vorgesehen, sodass sie über Aufrufe des Typs *getResource()* referenziert werden können.

Plattformmodularität: Modulschnittstellen

Jedes Plattformmodul muss eine externe Schnittstelle bereitstellen, sodass alle Aufrufe von Java-Klassen und -Schnittstellen innerhalb des Moduls über diese Schnittstelle erfolgen. Über diese externe Schnittstelle wird eine umfassende Anzahl Javadoc-Seiten bereitgestellt, sodass die Autoren anderer Module die externe Schnittstelle zuverlässig und ohne großen Aufwand verwenden können.

Die externen Schnittstellen sind in dem Paket "com.comergent.api" organisiert. Dieses Paket enthält alle von den Modulen unterstützten externen APIs. Die Anordnung erfolgt nach Modul: "com.comergent.api.converter", "com.comergent.api.logging" etc.

Schnittstellen aufrufen

Sie können eine Schnittstelle über eine Java-Klasse aufrufen, indem Sie ein Objekt oder eine untergeordnete Schnittstelle für die Schnittstelle umwandeln und dann eine von der Schnittstelle deklarierte Methode aufrufen. Jedes Modul arbeitet entweder mit dem einen oder mit dem anderen Verfahren, aber nicht mit beiden. Sorgen Sie beim Umgang mit einem vorhandenen Modul oder beim Erstellen eines neuen Moduls hinsichtlich des Aufrufs der Schnittstellen für Konsistenz. Damit erleichtern Sie Ihren Kollegen die Arbeit am selben Modul.

Grundsätzlich sollten Sie immer versuchen, mit den in dem Paket "com.comergent.api" enthaltenen Schnittstellen zu arbeiten. Dies sind die Schnittstellen, die auch dann von Release zu Release von den Plattformmodulen unterstützt werden, wenn sich die jeweiligen Implementierungen der Schnittstellen ändern.

Plattformmodulbeschreibungen

In diesem Abschnitt finden Sie kurze Beschreibungen zum Zweck der verschiedenen Plattformmodule sowie Beispiele zu deren Verwendung.

Access Policy

Über dieses Modul wird der Service zum Überprüfen der Zugriffsrichtlinien bereitgestellt.

Authentication

Über dieses Modul werden die APIs zum Authentifizieren von Berechtigungsnachweisen und Benutzern bereitgestellt.

Base64

Über dieses Modul werden die Klassen bereitgestellt, die zum Konvertieren von Daten in die bzw. aus der Base64-Schreibweise verwendet werden.

Classpath Appender

Über dieses Modul werden Klassen bereitgestellt, über die Pfade zum Klassenpfad hinzugefügt werden können.

Cryptography Service

Über dieses Modul werden Services zum Ver- und Entschlüsseln von Daten bereitgestellt.

Data Services

Über dieses Modul wird eine neue Paketierung sowie eine Bereinigung der vorhandenen Datenservicefunktion bereitgestellt. Die dazugehörige API wurde in ein separates Paket "com.comergent.api.dataservices" verlegt. "Data services" arbeitet zum Steuern der Eigenschaften ab sofort mit demselben Einstellungsmechanismus wie der Rest von Visual Modeler. Die Verwendung von Verbindungspools wurde in einem Pool vereinheitlicht, außerdem können Optimierungen vorgenommen werden. Die Seitennummerierung wurde aktualisiert und ist nicht länger davon abhängig, dass entsprechende Seitennummerierungsdateien in das Dateisystem geschrieben werden.

Dispatch Authorization

Über dieses Modul wird die Zugriffsprüfung gesteuert. Anhand dieser Prüfung können Sie dafür sorgen, dass jedem Benutzer nur die Teile der Anwendung angezeigt werden, für die er über die entsprechende Berechtigung verfügt.

Dispatch Framework

Über dieses Modul wird die Rahmendefinition für die Zuteilung der Visual Modeler-Klassen gesteuert, durch die die Servletanforderungs-, Antwort-, Kontext- und Sitzungsklassen mit den vom Zuteilungsmechanismus verwendeten Basiscontrollerklassen in Einklang gebracht werden.

Email Service

Über dieses Modul werden die Basis-APIs für das Versenden von E-Mail über Visual Modeler bereitgestellt.

Event Service

Über dieses Modul werden die von "EventBus" und "Events" verwendeten Klassen bereitgestellt.

Exception Service

Über dieses Modul werden die von Visual Modeler verwendete/n Basisrahmendefinition und -klassen für Ausnahmebedingungen bereitgestellt.

Global Cache Service

Über dieses Modul werden die für den Zugriff auf den Cache zu verwendenden APIs bereitgestellt.

Help

Über dieses Modul wird die Klasse "ComergentHelpBroker" bereitgestellt. Dabei handelt es sich um eine einfache Wrapperklasse zur Klasse "ServletHelpBroker" der JavaHelp 2.0-Implementierung.

Initialization Service

Über das Initialisierungsmodul wird der Initialisierungsservice bereitgestellt. Dabei handelt es sich um ein Paket, mit dem Sie Visual Modeler mithilfe einer konsistenten Rahmendefinition mit Klassen und Methoden initialisieren können.

Der Initialization Manager bildet einen zentralen Punkt zum:

- Definieren von Initialisierungstasks
- Umsetzen von Richtlinien in Bezug auf fehlgeschlagene Initialisierungen
- Zusammenfassen von Konfigurationsfragmenten

Die Hauptzuständigkeit des Initialization Manager besteht in der klar strukturierten und vorhersehbaren Steuerung einer Liste von Initialisierungstasks. Das impliziert den Umgang mit einer sortierten Liste, die

- programmatisch definiert
- oder als Datei im XML-Format angegeben werden kann.

Der folgende Codeauszug zeigt ein typisches Beispiel für die Verwendung der Klasse "InitManager".

```
InitManager initManager = InitManager.getInitManager();
try
{
String resourceName = args[0];
initManager.init(resourceName);
// or programatically created
//List modules = initModules();
//ResourceLocator resourceLocator = createNewResourceLocator();
//initManager.init(modules, resourceLocator);
}
catch (InitManagerException ime)
{
log.error(ime, ime);
System.exit(1);
}
// Initialization completed. OK to go on //
...
```

Sie können den Initialisierungsprozess über eine entsprechende Konfigurationsdatei angeben. Hier sehen Sie ein Beispiel für eine solche Datei:

```
<?xml version="1.0" encoding="UTF-8"?>
<initializationManager>
<resourceLocator>
```

```
<path>/a/b/c</path>
<path>.</path>
<path>CLASSPATH</path>
</resourceLocator>
<module name="ObjectManager"
initClass="com.comergent.objectManager.InitHelper>
<config name="Preferences">
/com/comergent/objectManager/preferences.xml
</config>
<init-param name="param0">param0Value</init-param>
</module>
<module name="module1" initClass="com.comergent.module1.InitHelper>
<config name="ObjectManager">
/com/comergent/module1/objectMap.xml
</config>
<config name="MessageTypes">
/com/comergent/module1/messageTypes.xml</config>
<config name="Preferences">
/com/comergent/modules1/preferences.xml
</config>
<init-param name="param1">param1Value</init-param>
</module>
<module name="module2" initClass="com.comergent.module2.InitHelper>
<config name="ObjectManager">
/com/comergent/module2/objectMap.xml
</config>
<config name="MessageTypes">
/com/comergent/module2/messageTypes.xml
</config>
<config name="Preferences">
/com/comergent/modules2/preferences.xml
</config>
<init-param name="param2">param2Value</init-param>
</module>
<!-- it is allowable to have no initClass -->
<module name="custom1" >
<config name="ObjectManager">
/com/comergent/module1/overlay/objectMap.xml
</config>
</module>
</initializationManager>
```

In diesem Beispiel werden bei Aufruf der folgenden Methode durch den Initialization Manager

```
com.comergent.objmgr.ObjManagerInitHelper.init(initParams,
configFragments, resourceLocator)
```

die folgenden Informationen verfügbar gemacht:

- "initParams" verfügt über eine Liste mit Schlüssel/Wert-Paaren: param0-param0Value
- "configFragments" verfügt über eine Liste mit:
  - `/com/comergent/module1/objectMap.xml`
  - `/com/comergent/module12/objectMap.xml`

- "resourceLocator" kann die Ressource entlang dem Pfad /a/b/c, current und entlang dem aktuellen Klassenpfad finden.

Internationalization

Über dieses Modul wird die Basisunterstützung für die in Visual Modeler verfügbaren Internationalisierungsfunktionen bereitgestellt.

Logging

Über dieses Modul wird der Zugriff auf den Protokollierungsservice bereitgestellt, mit dessen Hilfe die Aktivitäten in Visual Modeler erfasst werden. Mit der entsprechenden Eigenschaftendatei (**log4j.properties**) wird das Verhalten des Protokollierungsservice konfiguriert. Das Modul basiert auf dem log4j-Open-Source-Projekt und verwendet wie folgt dieselbe Syntax für die Konfiguration:

Log4j verfügt über die folgenden Hauptkomponenten: *Logger*, *Appender* und *Layouts*. Diese drei Komponententypen arbeiten zusammen, um so die Entwickler in die Lage zu versetzen, Nachrichten entsprechend Nachrichtentyp und -ebene zu erfassen und zur Laufzeit zu steuern, wie diese Nachrichten formatiert und wo sie dokumentiert werden.

Modul "Logging" konfigurieren

Sie können das Plattformmodul "Logging" mithilfe der "log4j.properties"-Konfigurationsdatei konfigurieren und dazu die Eigenschaften der entsprechenden Logger, Appender und Layouts angeben. Beispielsweise wird der folgende Ausschnitt zum Konfigurieren von Root-Logger und CMGT-Appender verwendet:

```
# Set root category priority
#log4j.rootCategory=info, CMGT
log4j.rootCategory=info, STDOUT
#log4j.rootCategory=info, CMGT, RTS
### START - CMGT
# CMGT appender
log4j.appender.CMGT=com.comergent.logging.ComergentRollingFileAppender
#log4j.appender.CMGT=com.comergent.logging.ComergentDailyRollingFileAppender

 #log4j.appender.CMGT.layout=org.apache.log4j.PatternLayout
log4j.appender.CMGT.layout=com.comergent.logging.ConversionPattern

 # The log format defaults to the "classic" format. This format is
# recommended for actual deployment to allow a log analyzer to
# work correctly.
log4j.appender.CMGT.layout.ConversionPattern=%d{yyyy.MM.dd HH:mm:ss:SSS} Env/%t:%p:%c{1} %m%n
```

Logger

Logger sind benannte Entitäten. Bei Loggernamen muss die Groß-/Kleinschreibung beachtet werden. Außerdem folgen sie einer hierarchischen Benennungsregel: Ein Logger gilt als Vorgänger eines anderen Loggers, wenn sein Name in Kombination mit einem nachgestellten Punkt als Präfix des untergeordneten Loggernamens fungiert. Ein Logger gilt als übergeordneter Logger eines untergeordneten Loggers, wenn zwischen diesem und dem untergeordneten Logger keine weiteren Vorgänger existieren.

Beispielsweise handelt es sich bei dem Logger “com.foo” um eine übergeordnete Entität zum Logger “com.foo.Bar”. Dementsprechend ist “java” eine übergeordnete Entität von “java.util” und ein Vorgänger von “java.util.Vector”. Dieses Benennungsschema sollte den meisten Entwicklern vertraut sein.

Der Root-Logger bildet die Spitze der Loggerhierarchie. Dieser Logger ist auf zwei Arten außergewöhnlich:

- Er ist immer vorhanden.
- Er kann nicht über den Namen aufgerufen werden.

Abgerufen wird er mit der "class/static"-Methode *Logger.getRootLogger()*. Alle anderen Logger werden instanziiert und über die "class/static"-Methode *Logger.getLogger(String name)* abgerufen.

Bei dieser Methode wird der Name des gewünschten Loggers als Parameter verwendet.

Den Loggern können verschiedene Ebenen zugeordnet werden. Mögliche Ebenen sind "DEBUG", "INFO", "WARN", "ERROR" und "FATAL". Diese werden in der Klasse "org.apache.log4j.Level" definiert. Wird einem Logger keine Ebene zugeordnet, wird diesem die Ebene seines nächsten Vorgängers mit zugeordneter Ebene vererbt. Formal ausgedrückt:

Ebenenvererbung: Die geerbte Ebene für einen bestimmten Logger ist gleich der ersten Ebene in der Loggerhierarchie mit einem anderen Wert als null. (Gezählt wird dabei beginnend mit dem Logger selbst und dann in der Hierarchie nach oben bis zum Root-Logger.)

Damit sichergestellt ist, dass alle Logger bei Bedarf eine Ebene übernehmen können, verfügt der Root-Logger grundsätzlich über eine zugeordnete Ebene.

Appender

Die Fähigkeit, selektiv Protokollierungsanforderungen auf der Basis der entsprechenden Logger zu aktivieren bzw. zu inaktivieren, ist nur ein Teil der Wahrheit. Es können mehrere Appender an einen Logger angehängt werden.

Mit der Methode "addAppender" wird ein Appender zu einem bestimmten Logger hinzugefügt. Jede aktivierte Protokollierungsanforderung für einen bestimmten Logger wird an alle Appender in diesem Logger sowie an alle in der Hierarchie höher angeordneten Appender weitergeleitet. Mit anderen Worten: Appender werden additiv aus der Loggerhierarchie übernommen. Wenn beispielsweise ein Konsolen-Appender zum Root-Logger hinzugefügt wird, werden alle aktivierten Protokollierungsanforderungen mindestens über die Konsole ausgegeben. Wenn dann zusätzlich ein Datei-Appender zu dem Logger hinzugefügt wird, werden die für den Logger und dessen untergeordnete Entitäten aktivierten Protokollierungsanforderungen über eine Datei und über die Konsole ausgegeben. Es ist möglich, dieses Standardverhalten außer Kraft zu setzen, sodass die Appendersummierung nicht länger additiv ausgeführt wird. Dazu müssen Sie beim Additivitätskennzeichen "false" angeben.

Die bei der Appendersummierung angewendeten Regeln werden nachfolgend zusammengefasst:

- Die Ausgabe in Bezug auf eine Protokollierungsanweisung für Logger C gilt für alle Appender in C und die entsprechenden Vorgänger. Das ist unter dem Begriff "Appenderadditivität" zu verstehen.

- Wenn allerdings für den Vorgänger eines Loggers das Additivitätskennzeichen "false" angegeben ist, wird die Ausgabe dieses Loggers zu allen seinen Appendern und Vorgängern geleitet (und zwar bis einschließlich deM Vorgänger, nicht aber bis zu den Appendern in den Vorgängern).
- Standardmäßig ist das Additivitätskennzeichen für Logger auf "true" eingestellt.

Layouts

Manchmal empfiehlt es sich, nicht nur das Ausgabeziel, sondern auch das Ausgabeformat anzupassen. Dazu können Sie einem Appender ein entsprechendes Layout zuordnen. Über das Layout können Sie eine Protokollierungsanforderung entsprechend Ihren Wünschen formatieren, wobei über einen Appender dafür gesorgt wird, dass die formatierte Ausgabe dann an ihr Ziel übertragen wird. Über "PatternLayout" (Teil der standardmäßigen log4j-Verteilung) können Sie (auf der Basis von Umrechnungsmustern entsprechend der Funktion "printf" in der Programmiersprache C) das Ausgabeformat angeben.

Beispielsweise wird über "PatternLayout" mit dem Umrechnungsmuster:

```
%r [%t] %-5p %c - %m%
```

in etwa folgende Ausgabe generiert:

```
176 [main] INFO Translator - got current date: 10/22/2005.
```

Beim ersten Feld handelt es sich um die Anzahl der seit dem Start des Programms verstrichenen Millisekunden. Das zweite Feld ist der Thread, durch den die Protokollierungsanforderung abgesetzt wird. Beim dritten Feld handelt es sich um die Ebene der Protokollierungsanweisung. Das vierte Feld enthält den Namen des der Protokollierungsanforderung zugeordneten Loggers. Beim Text hinter dem Zeichen “-” handelt es sich um die Anweisungsnachricht.

Memory Monitor

Über dieses Modul werden Klassen zum Überwachen und Protokollieren der Speicherbelegung bereitgestellt.

Message Type Entitlement

Über dieses Modul wird der Service bereitgestellt, mit dem die Berechtigung von Benutzern zum Aufrufen von Nachrichtentypen überprüft wird.

Die Schnittstellen sind in dem Paket "com.comergent.api.dispatchAuthorization" definiert. Dieses Paket enthält Factory-Klassen, Schnittstellen und Ausnahmebedingungen für den Service. Die Implementierungsklassen sind in dem Paket "com.comergent.dispatchAuthorization" enthalten.

Haupteinstiegspunkt in dieses Modul ist die Klasse "EntitlementRepository". Eine Instanz dieser Klasse stammt aus der Klasse "EntitlementFactory". Von den Anwendungen können benannte Instanzen der Klasse "EntitlementRepository" erstellt werden. Mithilfe benannter Instanzen wird das Testen von Komponenten vereinfacht. Außerdem können diese Instanzen in alternativen Implementierungsumgebungen von Nutzen sein.

Für eine Anwendung, für die Zuteilungsregeln oder andere Objekte in Bezug auf die Berechtigungen für Nachrichtentypen erforderlich sind, wird eine Logik ähnlich der folgenden ausgeführt:

```
import com.comergent.api.dispatchAuthorization.EntitlementRepository;
import com.comergent.api.dispatchAuthorization.EntitlementFactory;
import javax.xml.dom.Document;
...
Document document = ...;
...
EntitlementRepository repository =
EntitlementFactory.getEntitlementRepository();
repository.setRules(document);
```

Object Manager

Über dieses Modul werden die Klassen bereitgestellt, die zum Instanziieren von Objekten verwendet werden. Weitere Informationen hierzu finden Sie unter „Klassen "ObjectManager" und "OMWrapper"" auf Seite 24.

Out Of Band Response

Über dieses Modul werden Ausgabedaten in andere Ausgabedatenströme als die standardmäßigen JSP-Seiten übertragen.

Preferences Service

Über dieses Modul werden von Visual Modeler verwendete Konfigurationseigenschaften abgerufen und definiert. Sie können Eigenschaften über die folgenden Zeilen abrufen:

```
private static final Preferences prefs =
Preferences.getPreferences(MyClass.class);
// implict scope of "com.comergent.apps.module.MyClass"
int max = prefs.getInt("PromotionManager.maxValue", 100);
int min = prefs.getInt("PromotionManager.minValue", 1);
```

Über den zweiten Parameter in Aufrufen des Typs *getInt()* wird der Wert angegeben, der zurückgegeben werden muss, wenn keine Eigenschaft mit dem betreffenden Namen gefunden werden kann. Hinsichtlich der Konfigurationsdatei, in der die Eigenschaft definiert wird, wird angenommen, dass diese sich im Klassenpfad befindet (z. B. in der Datei **com.comergent.apps.module.Preferences.xml**). Wird die XML-Eigenschaftendatei mithilfe von "Preferences Service" eingelesen, müssen Sie sicherstellen, dass die XML-Datei das Comergent-Stammelement verwendet. Beispiel:

```
<Comergent>
<PromotionManager>
<maxValue>50</maxValue>
<minValue>20</minValue>
</PromotionManager>
</Comergent>
```

Sie können sicherstellen, dass "Preferences Service" zum Initialisieren der Eigenschaften verwendet wird, indem Sie die Konfigurationsdatei "WEB-INF/properties/init.xml" durch Hinzufügen eines Elements zu den folgenden Zeilen entsprechend anpassen:

```
<module name="PromotionMgr">
<config name="Preferences">
```

```
com/comergent/reference/apps/mktMgr/controller/Init.xml
</config>
</module>
```

Die Klasse "Preferences" enthält Methoden zum Abrufen und Definieren von Eigenschaftswerten. Beispiel:

```
prefs.putInt("PromotionManager.maxValue", 25);
prefs.putObject("currentShoppingCart", cartBean);
```

Bei Verwendung der Methode *putObject()* muss das Objekt den Anforderungen der XMLEncoder-API genügen. In erster Linie müssen über die Felder des Objekts Getter- und Setter-Methoden bereitgestellt werden.

Tagbibliotheken

Die über Visual Modeler bereitgestellten Tagbibliotheken stehen in Form eines Plattformmoduls zur Verfügung.

Thread Management

Über dieses Modul steht eine zentrale Funktion zur Handhabung von Threads (Erstellung, Abrufen des Status und erneute Verwendung) zur Verfügung. Dieses Modul wird über die Bibliothek "backport⌂util⌂concurrent.jar" bereitgestellt. Anwendungsentwickler müssen daher nicht länger den folgenden Aufruf tätigen:

```
Thread t = new Thread(new MyRunnable());
```

Stattdessen können Sie über eine zentrale Funktion:

- Threads (bei Bedarf) in einem Pool zusammenfassen und wiederverwenden
- Alle aktiven Threads überwachen, um so eine bessere CPU- und Ressourcennutzung zu erzielen
- Eine einfache Statusberichterstattung ermöglichen (Scoreboardstrategie: gemeinsam genutzte zentrale Position, über die der jeweils aktive Thread seinen Status ausgeben kann)
- Einfache Abbruch- und Unterbrechungssignale durch Aufrufen von *Thread.interrupt()* bereitstellen, damit lange aktive (aber in einer Schleife befindliche) Threads schneller abgebrochen werden können

Das Modul verfügt über die folgenden Funktionalitäten:

1. Threads transparent in einem Pool zusammenfassen und wiederverwenden.
2. Zu administrativen Zwecken Maßnahmen bereitstellen, um alle vom Thread Manager erfassten aktiven Threads abzufragen.
3. Für die Benutzer von Threads Maßnahmen bereitstellen, um den aktuellen Threadstatus in einem gemeinsamen Scoreboard zu dokumentieren.
4. Anleitungen zum Erkennen einfacher Schleifen oder zum Überprüfen des Protokolls mit dem Unterbrechungsstatus bereitstellen, um so lange aktive oder in einer Schleife befindliche Threads schneller abbrechen zu können.
5. Ein Zeitgebertool bereitstellen, um einen aktiven Thread benachrichtigen zu können, wenn ein Zeitgeberwert abgelaufen ist. Dieses Tool kann zum Implementieren einer einfachen Time-Out- oder Time-Sharing-Strategie verwendet werden.

Die API richtet sich weiterhin nach dem "Runnable()"-Muster: Die Anwendung erhält ein Thread-ähnliches Objekt und bringt dieses zur Ausführung.

```
Excutor executor = ExecutorFactory.getPooledExecutor();
executor.execute(new MyComergentRunnable());
```

XML Message Converter

Über dieses Modul steht eine Funktion zum Umwandeln von XML-Dokumenten von einer Nachrichtenkategorie (Familie und Version) in eine andere zur Verfügung. Der Paketname für die API lautet "com.comergent.api.converter", der Paketname für die Implementierungsklassen lautet "com.comergent.converter".

Zum API-Paket gehören:

- ConverterFactory: Dies ist die Factory-Klasse zum Erstellen von Convertern.
- Converter: Dies ist die Klasse, über die ein Dokument aus einer Nachrichtenkategorie in eine andere konvertiert wird. Dabei können Dokumente oder Streams als Quelle und Ziele für die Konvertierung verwendet werden.

XML Message Service

Über dieses Modul werden abgehende Nachrichten als XML-Dokumente erstellt und übergeben. Zur API gehören die Schnittstelle "MsgContext", die Schnittstelle "MsgService", die Klasse "MsgServiceFactory" und die Klasse "MsgServiceException" im Paket "com.comergent.api.msgService" sowie die Implementierungsklassen im Paket "com.comergent.msgService".

Die Schnittstelle "MsgService" verfügt über eine generische Methode *service()*, über die eine Data-Bean und ein XML-Dokument entsprechend der Angabe im Nachrichtenkontext übergeben werden können.

Das allgemeine Verwendungsmuster sieht wie folgt aus:

1. Unter Verwendung von "MsgContextFactory" eine "MsgContext"-Instanz erstellen.
2. Passende Attribute für das Kontextobjekt definieren.
3. Eine "MsgService"-Instanz für die Zielnachrichtenfamilie erstellen.
4. Eine Nachricht durch Aufrufen der Methode "service" mit einer Data-Bean und Nachrichtenkontext übergeben.

Beispiel:

```
MsgContext ctx = new MsgContext();
ctx.setMessageType("ERPOrderCreateRequest");
ctx.setURL("http://www.server.com");
ctx.setMessageCategory("ERPOrderCreateRequest");
ctx.setContentType("text/xml");
ctx.setRemoteUser(username);
ctx.setRemotePassword(password);
MsgService msgService =
MsgServiceFactory.getMsgService(ctx.getMessageCategory());
resultBean = msgService.service(requestBean, ctx);
```

XML Services

Über dieses Modul wird Funktionalität für XML-Parsing, XSL Transformation, DOM-Wrapper und Dienstprogrammklassen gekapselt.

Kapitel 8. Einführung zu Data-Beans und Geschäftsobjekten in Visual Modeler

Data-Beans in Visual Modeler

Bei einer Data-Bean handelt es sich um eine von Datenquellen unabhängige Darstellung einer realistischen Entität in Visual Modeler. Visual Modeler bedient sich eines (als Gruppe von XML-Dateien definierten) externen Schemas, wenn es darum geht, die Struktur für jeden einzelnen Data-Bean-Typ zu definieren. Beispielsweise werden Data-Beans als Datenstrukturen für Benutzer, Produktabfragelisten, Partner, Produkte und Arbeitsbereiche verwendet.

- Verwenden Sie die Klassen "OMWrapper" und "ObjectManager", um Instanzen der Data-Bean-Klassen zu erstellen. Weitere Informationen hierzu finden Sie unter „Klassen "ObjectManager" und "OMWrapper"" auf Seite 24.
- Sie können eine Data-Bean auch über "DataManager" erstellen. Rufen Sie also die "DataManager"-Methode *getDataBean(String beanName)* auf, um eine Data-Bean des genannten Typs zu erstellen. Bei dieser Methode wird, wenn keine derartige Data-Bean-Klasse existiert, eine Ausnahmebedingung des Typs "InvalidBizobjException" ausgelöst.

  Anmerkung: Die Verwendung dieser Methode wird nicht weiter unterstützt, da es keine Unterstützung für Erweiterungen des Datenobjekts gibt.

Lebenszyklus einer Data-Bean

Gewöhnlich gestaltet sich der grundsätzliche Arbeitsablauf mit einem Datenobjekt wie folgt:

1. Instanziieren Sie mithilfe der Klasse "OMWrapper" ein Data-Bean-Objekt.
2. Fügen Sie Daten zu der Bean hinzu. Verwenden Sie dazu die Set-Methoden, über die Sie direkt Werte in die Datenfelder eingeben können.
3. Definieren Sie die Data-Bean als persistent, um das neue Datenobjekt zum ersten Mal an seiner Datenquelle zu speichern.
4. Dementsprechend können Sie dasselbe Datenobjekt abrufen, indem Sie die Werte für die Schlüsselfelder definieren und anschließend einen *restore()*-Vorgang für die Data-Bean ausführen, um so die aktuellen Datenfeldwerte aus der jeweiligen Datenquelle auszulesen.
5. Führen Sie die erforderliche Geschäftslogik für die Data-Bean aus. Dadurch kann es zu Änderungen der speicherinternen Werte von Feldern kommen, die in der Datenquelle der Data-Bean gespeicherten Werte bleiben dagegen unberührt.
6. Speichern Sie die Änderungen an der Data-Bean. Definieren Sie dazu die Data-Bean für ihre Datenquelle als persistent.
7. Später empfiehlt es sich, das Datenobjekt, wenn es nicht mehr verwendet wird, zu löschen.
8. Schließlich empfiehlt es sich, die Daten komplett aus der Datenquelle zu entfernen und zu diesem Zweck das Datenobjekt zu löschen.

Für den Fall von Datenobjekten mit einer Datenbank als Datenquelle werden die Java-Methodenaufrufe und die entsprechenden SQL-Methoden in der folgenden Tabelle zusammengefasst:

Schritt	Java-Methode	SQL-Methode
Datenobjekt instanziieren	*OMWrapper.getObject()*	
Datenfelder belegen	*setDataField()*	
Datenfelder belegen	*setDataField()*	
Zum ersten Mal als persistent definieren	*persist()*	INSERT
Datenobjekt abrufen	*restore()*	SELECT
Geschäftslogik zum Aktualisieren von Feldwerten anwenden	*getDataField()* *setDataField()*	
Änderungen speichern	*persist()*	UPDATE
Datenobjekt löschen	*delete()*	UPDATE **Anmerkung:** Bei der Löschoperation wird die Spalte "ACTIVE_FLAG" der Tabellenzeile in der unterlegten Datenbank aktualisiert. Der betreffende Datensatz wird nicht aus der Tabelle entfernt.
Datenobjekt löschen	*erase()*	DELETE

Data-Bean definieren

Data-Beans werden mithilfe eines XML Schemas definiert. Von Data-Beans werden Zugriffsmethoden bereitgestellt, über die Werte bestimmter Datenfelder definiert und abgerufen werden können. Sie sollten grundsätzlich Data-Beans verwenden, wenn Sie Visual Modeler-Anwendungen anpassen.

Struktur eines Datenobjekts definieren

Jedes Datenobjekt muss über eine definierte Struktur verfügen, damit Visual Modeler eine Instanz des Datenobjekts erstellen kann. Die Struktur eines Datenobjekts wird in dessen XML Schema-Datei definiert. Darin wird angegeben, über welche Felder das Datenobjekt verfügt und ob untergeordnete Objekte vorhanden sind.

Jedes Datenobjekt entspricht einer Java-Klasse, durch die die Klasse "DataBean" erweitert wird. Diese Klassen werden als Data-Bean-Klassen bezeichnet. Die Data-Bean-Klassen werden automatisch als Teil des SDK-Zusammenführungsprozesses generiert. Wenn Sie die entsprechende Data-Bean-Klasse generieren, werden Methoden bereitgestellt, über die auf die Felder und untergeordneten Data-Beans zugegriffen wird, die in der XML-Datei des Datenobjekts deklariert sind.

Sie können die Definition des XML Schemas und damit der Datenobjekte und der entsprechenden Data-Bean-Klassen ändern, indem Sie die XML Schema-Dateien bearbeiten.

Die Konfigurationsdatei **DsRecipes.xml** wird zum Verknüpfen der verschiedenen Datenobjekte und ihrer Datenquellen verwendet. Über diese Datei wird auch angegeben, ob die Ordinalität des Datenobjekts “1” oder “n” lautet. Die Datenobjektdatei wird dazu verwendet, die präzise Struktur des Datenobjekts

anzugeben. In der Konfigurationsdatei **DsDataElements.xml** dagegen werden die Datentypen (LIST, LONG, STRING etc.) der einzelnenen Elemente angegeben.

Datenobjekte erweitern

Wenn Sie ein Datenobjekt mit einer XML Schema-Datei definieren, können Sie über das Attribut "Extends" deklarieren, das dieses Datenobjekt ein anderes Datenobjekt erweitert. Diese Funktion kann auf zwei Arten verwendet werden:

- Sie können ein bestimmtes Datenobjekt als übergeordnetes Element mehrerer unterschiedlicher Erweiterungsdatenobjekte verwenden, durch die eine einheitliche Gruppe von Datenfeldern gemeinsam genutzt wird. Beispielsweise wird das Datenobjekt "C3PrimaryRW" von vielen Datenobjekten in Visual Modeler erweitert. In diesem Datenobjekt werden die Basisdatenfelder "OwnedBy" und "AccessKey" zur Zugriffssteuerung bereitgestellt.
- Sie können ein Datenobjekt anpassen, indem Sie ein Datenobjekt erstellen, durch das das erste Datenobjekt erweitert wird. Durch Hinzufügen von Datenfeldern zum erweiternden Datenobjekt können Sie Attribute hinzufügen, die Sie als Teil Ihrer Anpassung einsetzen müssen. Über die Klasse "ObjectManager" können Sie sicherstellen, dass das erweiternde Datenobjekt erstellt wird, sobald das System zum Erstellen eines Datenobjekts des Typs "Erweitert" aufgerufen wird. Wenn vom vorhandenen Code die Klasse "ObjectManager" dazu verwendet wird, Instanzen des erweiterten Datenobjekts zu erstellen, werden beim Aufrufen dieses Codes zwar Instanzen des erweiternden Datenobjekts erstellt, doch bieten diese weiterhin Unterstützung für die Schnittstellen der erweiterten Datenobjekte, weshalb der vorhandene Code auch weiterhin funktioniert.

Die Methode "DataManager" verwendet eine *Anleitung* und ein *Datenobjekt*, um die Elementstruktur der Data-Bean oder des Geschäftsobjekts sowie die Position der Datenquelle, über die die Elementwerte bereitgestellt werden, zu bestimmen. Wenn Sie die Definition von Datenobjekten ändern oder neue Datenobjekte erstellen, müssen Sie die SDK-Zielelemente "generateDTD" und "generateBean" erneut ausführen, um so die "DataBean"-Klassen zu erstellen und zu kompilieren. Details hierzu finden Sie unter „Software-Development-Kit zum Anpassen der Visual Modeler-Implementierung verwenden” auf Seite 85. Alternative Methoden zum Erweitern von Datenobjekten finden Sie unter „Datenobjekte erweitern” auf Seite 58.

Data-Beans und Geschäftsobjekte erstellen

In den Klassen "ObjectManager" und "OMWrapper" in Visual Modeler werden Data-Beans und Geschäftslogikklassen erstellt, die von Controllern verarbeitet werden. Weitere Informationen hierzu finden Sie unter „Klassen "ObjectManager" und "OMWrapper"” auf Seite 24.

Die Geschäftslogikklassen werden über Controller aufgerufen. Dabei ist jeder Controller selbst dafür zuständig, zu ermitteln, welche Geschäftslogikklasse (falls überhaupt) als Reaktion auf eine Nachricht und den entsprechenden Nachrichtentyp erstellt werden muss.

Die Verwendung von Geschäftsobjekten und der Klasse "BusinessObject" wird nicht weiter unterstützt. Wenn möglich, sollten Sie Data-Bean-Klassen einsetzen und Geschäftsobjekte nur noch in Code aus früheren Versionen verwenden.

Klasse "DataContext"

Mit der Methode *restore()* wird eine Instanz der Klasse "DataContext" als Parameter verwendet. Mit der Klasse "DataContext" werden Information zu dem Kontext angegeben, in dem die Operation *restore()* ausgeführt wird. Diese Klasse kann dazu verwendet werden, die maximale Anzahl zurückzumeldender Ergebnisse und die Anzahl der auf einer Seite auszugebenden Ergebnisse (Paginierung) festzulegen. Sie kann auch verwendet werden, um anzugeben, ob für die Ergebnisse der Operation *restore()* eine Zugriffsprüfung erfolgen soll. Standardmäßig wird eine Zugriffsprüfung ausgeführt.

Im folgenden Codefragment wird beispielswiese eine "DataContext"-Klasse erstellt, es werden bestimmte Kontextwerte definiert und anschließend werden der Kontext sowie eine Abfrage zum Wiederherstellen einer Data-Bean eingesetzt:

```
DataContext temp_DataContext = new DataContext();
temp_DataContext.setMaxResults(DsConstants.NO_LIMIT);
temp_DataContext.setNumPerPage(-1);
skuMappingListBean.restore(temp_DataContext, query);
```

Wenn ein "DataContext"-Objekt initialisiert wird, werden aus den Konfigurationsdateien Werte der Elemente "DataServices.General.MaxResults" und "DataServices.General.NumPerCachePage" abgerufen, um diese Parameter für die Wiederherstellungsoperation zu definieren. Standardmäßig wird für keines der beiden Elemente ein Grenzwert definiert. Falls Sie das Verhalten eines "DataContext"-Objekts ändern müssen, stehen zu diesem Zweck entsprechende Zugriffsmethoden zur Verfügung. Weitere Informationen hierzu finden Sie im "DataContext"-Javadoc.

Im Rahmen der Klasse "DataContext" steht die Methode *setCacheId(String cacheId)* zur Unterstützung der Paginierung zur Verfügung. Mit dieser Klasse wird der verwendete Cache angegeben.

Was ist die "DataContext"-Klasse?

Mit der "DataContext"-Klasse wird das Verhalten von Operationen zum "Wiederherstellen" und "Als persistent definieren" gesteuert.

Welches Verhalten kann gesteuert werden?

Von einer "DataContext"-Instanz können folgende Aspekte gesteuert werden:

- Die Anzahl der auf einer Seite auszugebenden Abfrageergebnisse
- Die maximale Anzahl der zu verarbeitenden Abfrageergebnisse
- Die Verwendung mehrerer Seitengruppen pro Data-Bean-Typ und Sitzung

Wozu dienen die Cache-ID-Methoden?

Mit den Cache-ID-Methoden kann von einer Anwendung eine eindeutige Kennung für die Paginierung von Ergebnislisten angegeben werden. Mit dieser neuen Funktionalität ist eine Anwendung in der Lage, mehrere separate Ergebnislisten für eine bestimmte Data-Bean und Sitzung zu führen.

Wird von einer Anwendung keine Cache-ID angegeben, wird zum Kennzeichnen des Cache eine Kombination aus Beanname und Sitzungs-ID verwendet. In diesem

Fall werden bei allen weiteren Versuchen, dieselbe Data-Bean innerhalb derselben Sitzung wiederherzustellen, alle eventuell vorhandenen Ergebnisse überschrieben.

Über die Klasse "DataContext" werden die folgenden Methoden bereitgestellt, mit deren Hilfe die Cache-ID in Anforderungen zum Wiederherstellen von Data-Beans gesteuert werden kann:

- void setCacheId(String cacheId): Definiert eine neue Cache-ID. Diese Zeichenfolge wird in Kombination mit dem Beannamen und der Sitzungs-ID zum Generieren einer eindeutigen Kennung verwendet.
- String getCacheId( ): Gibt die aktuelle Cache-ID (oder, falls nicht definiert, den Wert null) aus.

Wie funktionieren "Max Results" und "Num Per Page"?

Über die Einstellung "Max Results" wird die maximale Anzahl an Datensätzen festgelegt, die im Rahmen eines Vorgangs zum Wiederherstellen abgerufen werden können. Wenn die angegebene Anzahl erreicht ist, wird die Anforderung freigegeben.

Über die Einstellung "Num Per Page" wird festgelegt, wie viele Datensätze pro Ergebniscacheseite gespeichert werden sollen. Liegt die ermittelte Anzahl unter dem Wert von "Num Per Page", wird kein Ergebniscache erstellt.

Beachten Sie, dass es der Anwendung mit dieser Kombination von Attributen möglich ist, eine Gruppe paginierter Ergebnisse abzurufen, während weiterhin eine maximale Anzahl abzurufender Datensätze festgelegt ist.

Über die Klasse "DataContext" werden für Anforderungen zum "Wiederherstellen" und "Als persistent definieren" für Data-Beans die folgenden Methoden zu "Max Results" und "Num Per Page" bereitgestellt:

- void setMaxResults(int maxResults): Definiert die maximale Anzahl an Ergebnissen, die ohne Paginierung ausgegeben werden.
- int getMaxResults( ): Ruft die maximale Anzahl an Ergebnissen ab, die ohne Paginierung ausgegeben werden.
- void setMaxPaginatedResults(int maxResults): Definiert die maximale Anzahl an Ergebnissen, die mit Paginierung ausgegeben werden.
- int getMaxPaginatedResults( ): Ruft die maximale Anzahl an Ergebnissen ab, die mit Paginierung ausgegeben werden.
- void setNumPerPage(int numPerPage)
- int getNumPerPage( )

Wenn von einer Anwendung die standardmäßigen Grenzwerte für Datenservices verwendet werden sollen, muss für die entsprechende Eigenschaft der Klasse "DataContext" der Wert **DsConstants.USE_DEFAULT** festgelegt werden. Bei den folgenden Werte handelt es sich um die Standardwerte:

- maxResults: 125
- maxPaginatedResults: 125
- numPerPage: 25

Wenn in der Anwendung kein Wert für "numPerPage" angegeben wird, wird der in der Datei "prefs.xml" enthaltene Wert verwendet. Wenn ein Wert weder über die Anwendung noch über die Datei "prefs.xml" definiert wird, wird der Wert "-1" verwendet, was bedeutet, dass die Anforderung nicht paginiert wird.

Außerdem werden über die folgenden Methoden Grenzwerte für Ergebnislisten bereitgestellt, die als Bestandteil der SQL-Abfrage direkt an die Datenbank weitergeleitet werden. Da von Visual Modeler aufgrund seiner Zugriffsrichtlinien (Beispiel: Dürfen dem Benutzer diese Daten angezeigt werden?) möglicherweise Ergebnisse gelöscht werden, können Sie mithilfe dieser Methoden höhere Grenzwerte für die Ergebnislisten angeben.

- public void setDBResultLimit(int limit)
- public int getDBResultLimit( )

Sie können auch die Vorgabe "DataServices.General.LimitDBResults" definieren. Wenn Sie für "LimitDBResults" den Wert "true" angeben, werden die Ergebnisse automatisch auf die Anzahl begrenzt, die für "MaxResults" (oder für "MaxPaginatedResults" im Falle paginierter Ergebnisse) zulässig sind. Damit dieser Mechanismus verwendet werden kann, müssen die Zugriffsrichtlinien in SQL geschrieben werden. Bei Oracle-Datenbanken darf für die Vorgabe "LimitDBResults" nicht der Wert "true" angegeben werden.

Die Zugriffsrichtlinien werden immer nach einer von zwei möglichen Vorgehensweisen gehandhabt. Viele von ihnen werden in WHERE-Klauseln in SQL konvertiert, die auf die Abfrage angewendet werden. Auf diese Weise kann die Datenbank die jeweiligen Zugriffsrichtlinie handhaben. Wenn die Richtlinie zu komplex (und beispielsweise auf einer Hierarchie von Partnern aufgebaut) ist, kann die Zugriffsrichtlinie nur beim Verarbeiten der Ergebnisse aus der Datenbank angewendet werden. Solche Richtlinien können nicht in SQL konvertiert werden.

Bei Oracle gibt es einige Fälle, in denen es für die SQL-Generierung erforderlich ist, dass im XML Schema Spaltenaliasnamen definiert werden. Dies ist aber nur erforderlich, wenn in der Abfrage mehrere Tabellen verknüpft werden, die ein und denselben Spaltennamen verwenden. Für SQL Server oder DB2 stellt das kein Problem dar.

Wie wird eine "DataContext"-Instanz erstellt?

Eine neue "DataContext"-Instanz wird momentan mithilfe des standardmäßigen Mechanismus “new” instanziiert:

```
DataContext dc = new DataContext();
```

Wie lauten die Standardeinstellungen für eine neue "DataContext"-Instanz?

Wenn “new DataContext( )” aufgerufen wird, werden den Attributen die folgenden Standardwerte zugeordnet:

Attribut
: **Standardwert**

doAccessCheck
: true

maxResults
: Eigenschaft "DataServices.xml maxResults"

numPerPage
: Eigenschaft "DataServices.xml numPerPage"

CacheId
: Null

doAccessCheck
true

List Data Beans

Eine spezielle Klasse von Geschäftsobjekten wird *list data beans* und *list business objects* genannt. Mit diesen Klassen können Sie eine Liste von Datenobjekten desselben Typs steuern. Wenn ein Datenobjektelement in einem Element "Recipe" mit der Ordinalität “n” deklariert wird, wird eine "List Data Bean" erstellt. Zugriffsberechtigungen werden weiterhin auf der Ebene des einzelnen Geschäftsobjekts gesteuert.

Anmerkung: Bei älteren Versionen von Datenobjekten wurde die Ordinalität in der zum Datenobjekt gehörenden Definitionsdatei definiert. Ab sofort wird die Ordinalität eines Datenobjekts über die entsprechende Anleitungsdatei gesteuert. In Datenobjekten der Version 6.0 wird das Ordinalitätsattribut weiterhin dazu verwendet, untergeordnete Datenobjekte, Referenzdatenobjekte und einbezogene Datenobjekte zu deklarieren.

Gewöhnlich müssen Sie keine "DataBeans" für Listendatenobjekte erstellen: Diese werden automatisch erstellt. Weitere Informationen hierzu finden Sie unter „Data-Bean-Klassen” auf Seite 23. Sie bieten Unterstützung für automatisch generierte Methoden, über die eine Liste der Datenobjekte ausgegeben wird. Beispielsweise können Sie dem folgenden Codefragment entnehmen, wie eine Liste mit Benutzern wiederherzustellen ist. Über ein durch “context” gekennzeichnetes "DataContext"-Objekt und ein als “query” gekennzeichnetes "DsQuery"-Objekt werden die anhand eines *restore()*-Aufrufs ausgegebenen Benutzer entsprechend eingeschränkt:

```
UserListBean userList = (UserListBean)
OMWrapper.getObject("com.comergent.bean.simple.UserListBean");
// Restore the list.
userList.restore(context, query);
// Return immediately if no results found.
if (userList.getUserCount() == 0)
{
return;
}
// At least one user in list, so walk through the list of users
ListIterator userIterator = userList.getUserIterator();
while (userIterator.hasNext())
{
UserBean user = (UserBean) userIterator.next();
// Perform any business logic on each user.
}
```

Anmerkung: Verwendung der Parameter "DataContext" und "DsQuery" in der Methode *restore()*: Diese Parameter werden verwendet, um zu steuern, wie die Abfrage mithilfe der Wissensdatenbank ausgeführt werden soll.

Application-, Entity- und Presentation-Beans

Es gibt verschiedene Hauptarten von in Visual Modeler verwendeten Data-Beans: Data-Beans, Application-Beans, Entity-Beans und Presentation-Beans. Im vorliegenden Abschnitt werden die wichtigsten Unterschiede beschrieben.

- Bei Data-Beans handelt es sich um die Java-Klassen, die automatisch aus der XML Schema-Beschreibung der Geschäftsobjekte erstellt werden. Bei Ausführung des SDK-Zielelements "generateBean" wird der Quellcode für die einzelnen Data-Beans erstellt. Diese Beans entsprechen dem Paket "com.comergent.bean.simple".

  Wenn möglich, sollten Sie den Befehl *instanceof* verwenden, um (statt des Geschäftsobjekttyps) die Data-Bean-Klasse zu bestimmen.
- Application-Beans sind Java-Klassen, die erstellt wurden, um Funktionalität hinzuzufügen, die von einfachen Beans nicht unterstützt wird. Beispielsweise können in einer Application-Bean zusätzliche Methoden bereitgestellt werden, die nicht automatisch generiert werden können, oder es können zwei oder mehr einfache Beans miteinander kombiniert werden, um Daten auf eine JSP-Seite übertragen zu können. Die Application-Beans sind nach Anwendungen organisiert und jede Anwendung verfügt über ein eigenes Paket für seine Application-Beans ("com.comergent.apps.<*anwendungsname*>.bean).

  Application-Beans können Unterklassen von einfachen Beans sein. Meist sind es aber Java-Klassen, die eine oder mehrere einfache Beans als Elementvariablen enthalten.

  Beispielsweise handelt es sich bei der Application-Bean-Klasse "com.comergent.appservices.productService.productMgr.BizProductBean" um eine Java-Klasse, die über eine Elementvariable verfügt, über die die Schnittstelle "com.comergent.bean.simple.IDataProduct" implementiert wird. Mit der Application-Bean-Klasse "BizProductBean" werden Methoden wie *getProductID()* an die Elementvariable "com.comergent.bean.simple.IDataProduct" delegiert, aber zusätzlich werden noch Methoden zum Abrufen der Funktionen eines Produkts, seiner Ablösungskette sowie der Preise bereitgestellt. Beachten Sie die Verwendung der Schnittstelle "IDataProduct" anstelle von "ProductDataBean": Dies ist ein Beispiel für die Verwendung einer generierten Schnittstelle anstelle der betreffenden Klasse. Weitere Informationen zur Generierung und Verwendung dieser Schnittstellen finden Sie unter Kapitel 12, „Generierte Schnittstellen”, auf Seite 79.

  Entsprechend der Konvention gilt: Wenn Sie eine Application-Bean erstellen, um eine Data-Bean einzuschließen, müssen Sie eine Methode namens *getDataBean()* bereitstellen, über die die Data-Bean abgerufen wird.
- Presentation-Beans werden auch dazu verwendet, Daten auf JSP-Seiten zu übertragen. Diese Beans unterscheiden sich dadurch von Application-Beans, dass sie keine Geschäftslogik bereitstellen. Zwecks Benutzerfreundlichkeit können hier mehrere Data-Beans in einer einzigen Klasse zusammengefasst werden oder es werden Formatierungsangaben bereitgestellt. Wie bei den Application-Beans muss in den Presentation-Beans eine Methode bereitgestellt werden, über die der Zugriff auf die unterlegte Data-Bean möglich ist. Beispielsweise wird über die Schnittstelle "IPresProduct" die Methode *getIRdProduct()* bereitgestellt: Mit dieser Methode wird die Schnittstelle "IRdProduct" bereitgestellt und Sie können (bei Bedarf) einen Downcast auf die unterlegte Data-Bean oder auf die erweiterte Data-Bean vornehmen.
- Entity-Beans kamen in früheren Releases von Visual Modeler zum Einsatz. Sie führten dieselbe Rolle wie Application-Beans aus. Ihre Verwendung wird allerdings nicht weiter unterstützt.

Gespeicherte Vorgänge verwenden

Beim Wiederherstellen von Datenobjekten können Sie gespeicherte Vorgänge verwenden. Der Name des gespeicherten Vorgangs wird im Element "ExternalName" des Datenobjekts deklariert.

Wenn Sie Datenobjekte definieren, müssen Sie unbedingt das Attribut "SourceType" angeben. Dieses Attribut kann die folgenden Werte annehmen:

- “1”: Die zugrunde liegende Datenquelle verwendet eine Tabelle. Dies ist der Standardwert.
- “2”: Die zugrunde liegende Datenquelle verwendet einen gespeicherten Vorgang.

Ist kein Attribut "SourceType" definiert, gilt standardmäßig, dass es sich bei dem für das Datenobjekt zugrunde liegenden Quellentyp um eine Tabelle handelt.

Data-Bean-Methoden

Im Allgemeinen sollten Sie von den generierten Schnittstellen Gebrauch machen, die von den Data-Beans bereitgestellt werden: über diese werden die Zugriffs- und Datenmethoden organisiert, mit deren Hilfe Sie den Zugriff auf die Datenobjekte während deren Lebenszyklus steuern können. Weitere Informationen hierzu finden Sie unter Kapitel 12, „Generierte Schnittstellen”, auf Seite 79.

Verwenden Sie für die Zugriffssteuerung den entsprechenden Sicherheitsmechanismus anhand der Zugriffsrichtlinien.

"IData"-Methoden

Die "IData"-Schnittstelle verfügt über die folgenden wichtigen Methoden:

- *copyBean()*: Über diese Methode können Sie die Werte von Datenfeldern von einer Bean in die andere kopieren. Dazu ist ein Argument erforderlich: Dies muss eine Bean sein, bei der es sich entweder um eine Instanz derselben Klasse oder um eine Unterklasse der Bean handeln muss, durch die diese Methode aufgerufen wird.
- *delete()*: Über diese Methode wird das entsprechende Datenobjekt als gelöscht markiert: Die Spalte "ACTIVE_FLAG" in der diesem Datenobjekt entsprechenden Datenbanktabelle wird auf “N” gesetzt, wenn das Objekt als persistent definiert wird. Beachten Sie, dass Sie nach dem Aufruf der Methode *delete()* die Methode *persist()* aufrufen müssen: andernfalls ist der Löschvorgang nicht gültig.
- *erase()*: Über diese Methode wird der dem Geschäftsobjekt entsprechende Datenbanksatz entfernt. Beachten Sie, dass es beim Entfernen von Datensätzen aus Datenbanktabellen zu Datenintegritätsproblemen kommen kann, wenn durch andere Tabellen Bezug auf Schlüssel genommen wird, die gelöscht wurden. Daher sollten Sie diese Methode nur dann verwenden, wenn Sie alle Verwendungszwecke des betreffenden Datensatzes und seiner Schlüssel einschätzen und die relevanten Datensätze aus den anderen Tabellen gelöscht werden können.
- *generateKeys()*: Über diese Methode werden die Schlüsselfelder der Data-Bean belegt. Sie können diese Methode aufrufen, ohne die Methode *persist()* aufrufen zu müssen. Bei Aufruf dieser Methode können Sie die generierten Schlüssel dazu verwenden, andere Objekte zu erstellen, für die die Schlüssel erforderlich sind.
- *setDataContext()*: Über diese Methode wird der Datenkontext definiert, damit bei den Aufrufen der Methoden *restore()* und *persist()* die passenden Werte für Parameter (z. B. die Anzahl der Ergebnisse pro Seite bei paginierten Datenlisten) verwendet werden. Weitere Informationen zur Klasse "DataContext" finden Sie unter „Klasse "DataContext"” auf Seite 48.
- *persist()*: Über diese Methode werden die Daten in der Data-Bean an ihrer Datenquelle gespeichert.

- *prune()*: Über diese Methode wird die Bean im Speicher zum Löschen markiert. Wenn Sie nach der Methode *prune()* die Methode *restore()* aufrufen, hat das keinerlei Einfluss auf die der Bean zugrunde liegende Datenquelle.
- *restore()*: Über diese Methode werden die Daten für die Data-Bean aus ihrer Datenquelle abgerufen. Weitere Informationen zum Verwenden der Klasse "DataContext" in der Methode *restore()* finden Sie unter „Klasse "DataContext"" auf Seite 48.
- *update()*: Über diese Methode wird der diesem Geschäftsobjekt entsprechende Datenbanksatz aktualisiert.

Beachten Sie, dass nach dem Aufruf von Methoden, durch die sich Änderungen im Status ergeben, grundsätzlich die Methode *persist()* aufgerufen werden muss, damit die jeweilige Änderung auf den Datenbanksatz angewendet wird.

Über die Schnittstelle "IData" werden auch die Methoden *isRestorable()* und *isPersistable()* bereitgestellt, über die geprüft wird, ob ein Datenobjekt wiederhergestellt bzw. als persistent definiert werden kann.

"IRd"- und "IAcc"-Schnittstellenmethoden

Über die Schnittstelle "IRd" werden die schreibgeschützten Zugriffsmethoden für die Datenobjektfelder bereitgestellt. Über die Schnittstelle "IAcc" wird die Schnittstelle "IRd" insofern erweitert, als für jedes Datenfeld die entsprechenden "set"-Zugriffsmethoden hinzugefügt werden. Durch die Abgrenzung dieser beiden Schnittstellen haben Sie die Möglichkeit, ein schreibgeschütztes Objekt an eine Clientanwendung oder eine JSP-Seite zu übergeben.

Nehmen Sie beispielsweise an, dass in der Datenobjektdatei mit den Bedingungen (**Condition.xml**) ein "DataField"-Element wie folgt angegeben ist:

```
<DataField Name="ControlType"
Writable="y" Mandatory="y"
ExternalFieldName="CONTROL_TYPE"/>
```

In der dann automatisch generierten Schnittstelle "IRdCondition" gibt es den folgenden Methodenaufruf:

public Long getControlType()

In der automatisch generierten Schnittstelle "IAccCondition" dagegen gibt es den folgenden Methodenaufruf:

```
public void setControlType(Long value) throws ICCException
```

Die Signaturen dieser beiden Zugriffsmethoden richten sich nach der entsprechenden "DataElement"-Definition in der Datei **DsDataElements.xml**:

```
<DataElement Name="ControlType" DataType="LONG"
Description="Condition Control Type" MaxLength="20" />
```

Anmerkung: Wenn Sie für das Attribut "Writable" eines Datenfelds den Wert “n” angeben, wird die entsprechende "*set*DataField*()*"-Methode nicht generiert.

Daten wiederherstellen und als persistent definieren

Die folgenden wichtigen Operationen können für ein Datenobjekt ausgeführt werden: *delete*(), *persist()* und *restore()*.

- Wenn Sie die Methode *delete()* für ein Datenobjekt aufrufen, kennzeichnen Sie das betreffende Objekt damit als gelöscht, was dazu führt, dass dieses Datenobjekt anschließend von keiner anderen Anwendung mehr aufgerufen wird. Dazu wird als Wert für die Spalte "ACTIVE_FLAG" in der unterlegten Datenbank "N" definiert. Beachten Sie, dass die Datenobjektdaten nicht aus der Datenquelle gelöscht werden. Wenn die dem Datenobjekt unterlegte Datenbank nicht über eine Spalte "ACTIVE_FLAG" verfügt, darf die Methode *delete()* nicht verwendet werden. Allerdings können Sie noch die Methode *erase()* verwenden, um solche Datenobjekte aus der Wissensdatenbank zu entfernen.
- Wenn Sie eine Data-Bean über die Methode *persist()* als persistent definieren, werden die in der "DsElement"-Baumstruktur des Datenelements enthaltenen Daten von Visual Modeler in der/den entsprechenden externen Datenquelle(n) gespeichert. Beachten Sie, dass sich Visual Modeler mit der Methode *persist()* sowohl mit der Aktualisierung bereits vorhandener Datenobjekte als auch mit der Erstellung neuer Datenobjekte befasst.
- Wenn Sie eine Data-Bean oder ein Geschäftsobjekt über die Methode *restore()* wiederherstellen, werden durch Visual Modeler die entsprechenden Daten aus der/den entsprechenden externen Datenquelle(n) abgerufen. Ist in der Methode *restore()* kein Abfrageobjekt angegeben, werden alle Datenobjekte wiederhergestellt, deren Werte in den Schlüsselfeldern denen in der Data-Bean entsprechen.
  - Beachten Sie Folgendes: Wenn Sie die Methode *restore()* für eine nicht in einer Liste erfasste Data-Bean aufrufen, können Sie davon ausgehen, dass deren Daten ausschließlich über die Werte abgerufen werden können, die in den entsprechenden Schlüsselfeldern definiert sind. Wenn der Aufruf *restore()* abgesetzt wird, wird nicht überprüft, ob nur ein Datensatz abgerufen wurde. Daher wird der jeweils erste abgerufene Datensatz zum Auffüllen der Data-Bean verwendet. Wird kein Datensatz abgerufen, wird nach Aufruf der Methode *restore()* eine Ausnahmebedingung des Typs "ICCException" ausgegeben.
  - Wenn Sie die Methode *restore()* für eine in einer Liste erfasste Data-Bean aufrufen, müssen Sie gewöhnlich die Klasse "DsQuery" angeben. Wenn Sie keine Klasse "DsQuery" angeben, sind in der wiederhergestellten aufgelisteten Data-Bean die Daten aller Data-Beans dieses Typs enthalten.

Data-Bean-Methode "restore()"

In diesem Abschnitt finden Sie eine Beschreibung der wichtigsten Formate der Data-Bean-Methode *restore()*.

```
public void restore(DataContext dataContext, DsQuery dsQuery)
```

Dies ist das wichtigste Format der Methode *restore()*. Verwenden Sie den Parameter "dsQuery", um die Abfrage anzugeben, die über die Wiederherstellungsoperation ausgeführt werden soll. Über den Parameter "dataContext" wird die maximale Anzahl zurückzuliefernder Objekte und für die Paginierung die Anzahl der Ergebnisse pro Seite bestimmt. Verwenden Sie den Parameter "dataContext", um anzugeben, ob überprüft werden soll, ob der aktuelle Benutzer über die zur Ausführung dieser Operation erforderlichen Berechtigungen verfügt. Standardmäßig wird eine Zugriffsprüfung ausgeführt. Möchten Sie darauf verzichten, müssen Sie diese mithilfe der Methode *disableAccessCheck()* inaktivieren.

public void restore(DataContext dataContext)

Dies ist äquivalent zum Aufruf *restore(dataContext, null)*.

Es folgt ein Beispiel für die Verwendung der beiden Klassen "DataContext" und "DsQuery" zum Steuern des Aufrufs der Methode *restore()*:

```
try
{
DataContext dataContext = new DataContext();
if (doAccessCheck == true)
{
dataContext.enableAccessCheck();
}
else
{
dataContext.disableAccessCheck();
}
dataContext.setNumPerPage(pageSize);
DsQuery dsQuery = QueryHelper.newWhereClause("PartnerKey",
DsConstants.EQUALS, partnerKey);
LightWeightPartnerBean partnerBean =
(com.comergent.bean.simple.LightWeightPartnerBean)
com.comergent.dcm.util.OMWrapper.getObject(
"com.comergent.bean.simple.LightWeightPartnerBean");
partnerBean.restore(dataContext, dsQuery);
QueryHelper.freeQuery(dsQuery);
return partnerBean;
}
catch (ICCException e)
{
throw (new ProfileMgrException(e));
}
```

Data-Bean-Methode "persist()"

In diesem Abschnitt finden Sie eine Beschreibung der wichtigsten Formate der Data-Bean-Methode *persist()*.

```
public void persist(DataContext dataContext)
```

Wenn über "dataContext" angegeben wird, dass eine Zugriffsprüfung erfolgen soll, wird bei diesem Format der Methode *persist()* vor Durchführung der Operation eine Zugriffsprüfung vorgenommen. Verfügt der Benutzer nicht über die erforderlichen Berechtigungen, wird die Operation nicht ausgeführt.

Verschiedene Methoden

Methode "getBizObj()"

Wenn Sie eine Geschäftsobjektdarstellung des Datenobjekts und seiner Daten abrufen möchten, können Sie die Methode *getBizObj()* aufrufen. Dies ist dann nützlich, wenn Sie die internen Strukturen des Objekts anzeigen möchten. Beispiel:

```
BusinessObject bo = bean.getBizobj();
ComergentDocument doc = bo.serializeToXml();
doc.prettyPrint();
```

Beachten Sie, dass es sich hierbei um eine nicht weiter unterstützte Methode handelt.

Methode "writeExternal()"

Verwenden Sie diese Methode zur Ausgabe einer XML-Darstellung der Data-Bean und ihrer Daten.

Untergeordnete Datenobjekte

Von vielen Datenobjekten werden mithilfe des Elements "ChildDataObject" untergeordnete Datenobjekte deklariert. So wird beispielsweise im Rahmen des Datenobjekts "ShoppingCart" wie folgt das untergeordnete Datenobjekt "LineItem" deklariert:

```
<DataObject Name="ShoppingCart" Extends="C3PrimaryRW"
ExternalName="CMGT_CARTS" ObjectType="JDBC" Version="6.0">
...
<ChildDataObject Access="RWID" Name="LineItem">
<Relationship CascadeDelete="y" CascadeErase="n"
ChangeUpdatesParent="y">
<JoinKeys>
<JoinKey DstJoinField="ShoppingCartKey"
SrcJoinField="ShoppingCartKey"/>
</JoinKeys>
</Relationship>
</ChildDataObject>
...
</DataObject>
```

Dessen Element "Relationship" verfügt über Attribute, über die angegeben wird, wie sich untergeordnete Elemente verhalten sollen, wenn das jeweilige übergeordnete Element aktualisiert wird, und ob ein übergeordnetes Element aktualisiert werden soll, wenn ein untergeordnetes Element geändert wird. Über "JoinKey"-Elemente wird festgelegt, wie die untergeordneten Datenobjekte wiederhergestellt werden. Gewöhnlich wird dazu angegeben, wie Werte aus dem übergeordneten Datenobjekt dazu verwendet werden sollen, Werte im untergeordneten Datenobjekt zu definieren.

Bei der Generierung der übergeordneten Data-Bean wird eine Methode mit Namen *get*ChildDataObject*Iterator()* generiert, über die ein "ListIterator"-Objekt mit den untergeordneten Data-Beans zurückgegeben wird. Wenn Sie durch die Objekte navigieren, können Sie die einzelnen untergeordneten Data-Beans nacheinander untersuchen und mithilfe der standardmäßigen Zugriffsmethoden auf deren Felder zugreifen.

Beispielsweise bietet die Klasse "ShoppingCartBean" Unterstützung für die Methode "getLineItemIterator()". Im folgenden Code wird gezeigt, wie das Feld einer Position abgerufen wird:

```
/*
shoppingCartBean is a ShoppingCartBean object that has already been
restored
*/
ListIterator lineItemIterator =
shoppingCartBean.getLineItemIterator();
```

```
LineItemBean lineItemBean =
(LineItemBean) lineItemIterator.getLineItemBean(0);
Long quantity = lineItemBean.getQuantity();
```

Wenn ein übergeordnetes Datenobjekt wiederhergestellt wird, werden die untergeordneten Datenobjekte nicht wiederhergestellt. Diese werden nur dann wiederhergestellt, wenn die Anwendung in der oben beschriebenen Weise auf die untergeordneten Elemente zugreift.

Datenobjekte erweitern

Informationen zu diesem Vorgang

Bei allen Implementierungen von Visual Modeler ist es gängige Praxis, dass Datenfelder zu Datenobjekten hinzugefügt oder zur Erweiterung der bereits vorhandenen Datenobjekte neue Datenobjekte erstellt werden müssen.

Es wird empfohlen, die Zusatzdaten in einer neuen Datenbanktabelle zu speichern. Für den Zugriff auf diese neue Tabelle muss dann ein neues Element "DataObject" definiert werden. Anschließend wird durch Hinzufügen eines neuen Elements "IncludeDataObject" ein weiteres "DataObject"-Element definiert, durch das das ursprüngliche "DataObject"-Element erweitert wird.

Nehmen Sie beispielsweise an, dass Sie ein neues Datenfeld zum Datenobjekt "Order" hinzufügen müssen, um “gehostete” Aufträge überwachen zu können: Dies sind Aufträge, die bei Partnern mit Onlineschaufenster platziert werden. Das zusätzliche Datenfeld ist der Partnerschlüssel des Onlineschaufensterpartners. Es wird folgende Vorgehensweise empfohlen:

Vorgehensweise

1. Erstellen Sie unter dem Namen "HostedPartner" ein neues Datenobjekt mit genau zwei Feldern: einem Feld "OrderKey" und einem Feld "PartnerKey". Definieren Sie das neue Datenobjekt so, dass auf eine zweispaltige Tabelle verwiesen wird: auf die Tabelle "CMGT_ORDER_X_PARTNER" mit den beiden Spalten "ORDER_KEY" und "PARTNER_KEY".

   ```
   <?xml version="1.0"?>
   <DataObject Name="HostedPartner"
   ExternalName="CMGT_ORDER_X_PARTNER" ObjectType="JDBC"
   Version="6.0">
   <KeyFields>
   <KeyField Name="OrderKey" ExternalName="ORDER_KEY"/>
   <KeyField Name="PartnerKey" ExternalName="PARTNER_KEY"/>
   </KeyFields>
   <DataFieldList>
   <DataField Name="OrderKey" ExternalFieldName="ORDER_KEY"
   Mandatory="n" Writable="y"/>
   <DataField Name="PartnerKey"
   ExternalFieldName="PARTNER_KEY"
   Mandatory="n" Writable="y"/>
   </DataFieldList>
   </DataObject>
   ```

2. Erstellen Sie unter dem Namen "HostedOrder" ein neues Datenobjekt, durch das das Datenobjekt "Order" erweitert wird. Die Datei **HostedOrder.xml** hat folgendes Aussehen:

```
<?xml version="1.0"?>
<DataObject Name="HostedOrder" Extends="Order" ObjectType="JDBC"
Version="6.0">
<IncludedDataObject Access="RWID" Name="HostedPartner"
Ordinality="1">
<Relationship CascadeDelete="y" CascadeErase="n"
ChangeUpdatesParent="y">
<JoinKeys>
<JoinKey DstJoinField="OrderKey"
SrcJoinField="OrderKey"/>
</JoinKeys>
</Relationship>
</IncludedDataObject>
</DataObject>
```

Es stehen drei verschiedene Basisvorgehensweisen zur Verfügung:

3. Sie können Erweiterungen verwenden, um einfach zusätzliche Datenfelder hinzuzufügen und den Tabellennamen zu überschreiben. Auf diese Weise können Sie alle Daten in einer neuen Tabelle erfassen. Diese Vorgehensweise ist dann sinnvoll, wenn Sie zwar dieselben Daten, aber eine bestimmte Kopie davon benötigen. (Erstellen Sie ggf. einen Snapshot, um so zu dokumentieren, wie ein Datenobjekt "Order" ausgesehen hat, bevor es zu einem Datenobjekt "HostedOrder" wurde.)
4. Sie können das Datenobjekt "Order" dahingehend erweitern, dass ein Element "IncludedDataObject" für das Datenobjekt "HostedOrder" hinzugefügt wird, wobei über "HostedOrder" nur zusätzliche Daten zur Speicherung in einer anderen Tabelle definiert werden. Das bedeutet, dass Änderungen an den Originaldatenfeldern des Datenobjekts "Order" weiterhin in der Tabelle "Order", die zusätzlichen Daten für das Datenobjekt "HostedOrder" aber in einer anderen Tabelle als persistent definiert werden. Dies ist die empfohlene Vorgehensweise.
5. Sie können das Datenobjekt "HostedOrder" definieren, indem Sie angeben, dass es sich beim Datenobjekt "Order" um ein integriertes Datenobjekt ("IncludedDataObject") handelt. Damit erzielen Sie den gleichen Effekt wie bei der zweiten Alternative. Das Problem bei dieser Vorgehensweise besteht darin, dass durch ein Datenobjekt "HostedOrder" keine Erweiterung des Datenobjekts "Order" erfolgt und dass dieses vom Anwendungscode nicht mehr als Datenobjekt des Typs "Order" behandelt werden kann.

   Anmerkung: Bei der Verwendung von zwei Tabellen kommt es zwar zu leichten Leistungseinbußen, aber die Abfrageausführung ist nie ein Problem gewesen. Durch die Verwendung von zwei Tabellen kann (je nach Anforderungen) die Datenredundanz reduziert werden.

Ergebnisse

Wenn Sie nur gelegentlich Bezug auf die Kundenerweiterung nehmen, empfiehlt sich die Verwendung eines Datenobjekts des Typs "ChildDataObject", um so in den Genuss der Vorteile einer verzögerten Verbindung zu kommen.

Data-Bean-Beispiel

In diesem Abschnitt wird beschrieben, wie ein Datenobjekt definiert und verwendet wird. Stellen Sie sich vor, dass Sie ein Datenobjekt zur Darstellung einer einfachen Anfrage seitens eines Kunden verwenden möchten. Dazu zählen:

- Eine E-Mail-Adresse für den Kunden

- Das Datum der Anfrage
- Das Datum der Anfragebeantwortung (optional)
- Der Inhalt der Anfrage
- Der Inhalt der Antwort (optional)
- Die Produkt-ID des Produkts, zu dem die Anfrage gestellt wurde (optional)

Datenobjektdefinition erstellen

Informationen zu diesem Vorgang

So erstellen Sie eine Datenobjektdefinition:

Vorgehensweise

1. Erstellen Sie das Geschäftsobjektelement "Enquiry" und fügen Sie es zu der Datei **DsBusinessObjects.xml** hinzu.

   ```
   <BusinessObject Name="Enquiry" Version="6.0"
   Description="Customer enquiry"/>
   ```

   Verwenden Sie das Attribut "Version", um verschiedene Versionen von Geschäftsobjekten zu steuern, die möglicherweise gleichzeitig verwendet werden. Beachten Sie, dass das Attribut "Version" auch verwendet wird, um festzulegen, ob Zugriffsprüfungen automatisch (Version 5.0 oder höher) ausgeführt werden sollen oder nicht.

2. Erstellen Sie die Anleitung für dieses Geschäftsobjekt und fügen Sie sie zur Datei **DsRecipes.xml** hinzu.

   ```
   <Recipe Name="Enquiry" Version="6.0" Ordinality="n"
   Description="Customer enquiry">
   <DataObjectList>
   <DataObject Name="Enquiry"
   DataSourceName="ENTERPRISE" />
   </DataObjectList>
   </Recipe>
   ```

   Das Attribut "Name" in der Anleitung muss exakt dem Namen des Geschäftsobjekts entsprechen (Groß-/Kleinschreibung muss beachtet werden). In Release 9.1 können zwar zu jeder Anleitung mehrere in der Datenobjektliste definierte Datenobjekte gehören, aber immer nur ein Datenobjekt ist ein *modifizierbares*. In den Datenobjekten werden die Datenquellennamen als Attribut der einzelnen Datenobjektelemente definiert. Über diese Einträge werden die Quellen bestimmt, aus denen das Geschäftsobjekt seine Daten bezieht, und es wird die Quelle festgelegt, in der das Geschäftsobjekt als persistent definiert werden kann.

3. Erstellen Sie zum Definieren des Datenobjekts eine Datei mit Namen **Enquiry.xml**. Der Name des Datenobjektelements muss exakt dem Attribut "Name" entsprechen (Groß-/Kleinschreibung muss beachtet werden), das im Eintrag "DataObject" der Anleitung definiert ist.

   Im vorliegenden Beispiel befinden sich die Daten für diese Datenobjekte in einer Datenbanktabelle mit Namen "CMGT_ENQUIRY". Über das Attribut "ExternalFieldName" der einzelnen "DataField"-Elemente wird angegeben, welche Spalte für den Abruf des "DataField"-Werts zu verwenden ist. Beispielsweise sind in der Spalte "EMAIL_ADDRESS" der Tabelle "CMGT_ENQUIRY" die zu einer bestimmten Abfrage gehörenden E-Mail-Adressdaten enthalten.

   ```
   <?xml version="1.0"?>
   <DataObject Name="Enquiry" Extends="C3PrimaryRW" Version="6.0"
   ```

```
ExternalName="CMGT_ENQUIRY"
Access="R" ObjectType="JDBC">
<KeyFields>
<KeyField Name="Key" ExternalName="ENQUIRY_KEY"/>
</KeyFields>
<DataFieldList>
<DataField Name="EnquiryKey"
Writable="n" Mandatory="y"
ExternalFieldName="ENQUIRY_KEY"/>
<DataField Name="EmailAddress"
Writable="n" Mandatory="y"
ExternalFieldName="EMAIL_ADDRESS"/>
<DataField Name="EnquiryDate"
Writable="n" Mandatory="y"
ExternalFieldName="ENQUIRY_DATE"/>
<DataField Name="ResponseDate"
Writable="n" Mandatory="n"
ExternalFieldName="RESPONSE_DATE"/>
<DataField Name="TimeToRespond"
Writable="n" Mandatory="n"/>
<DataField Name="EnquiryContent"
Writable="n" Mandatory="y"
ExternalFieldName="ENQUIRY_CONTENT"/>
<DataField Name="ResponseContent"
Writable="y" Mandatory="n"
ExternalFieldName="RESPONSE_CONTENT"/>
<DataField Name="SKU"
Writable="n" Mandatory="n"
ExternalFieldName="SKU"/>
</DataFieldList>
</DataObject>
```

Beachten Sie die Definition des Datenfelds "TimeToRespond": Dabei handelt es sich nicht um ein "ExternalFieldName"-Attribut, da es nicht einer Datenbankspalte entspricht. Die Werte für dieses Feld werden zur Laufzeit berechnet und in der Data-Bean "EnquiryBean" definiert, sodass der Wert angezeigt werden kann.

4. Definieren Sie in der Datei **DsDataElements.xml** die Datenelemente ("DataElements") "Enquiry" und "EnquiryList":

```
<DataElement Name="Enquiry" Description="Enquiry"
DataType="HEADER"/>
<DataElement Name="EnquiryList" Description="Enquiry list"
DataType="LIST"/>
```

5. Definieren Sie für jedes Datenfeld ("DataField") in der Datei **DsDataElements.xml** ein Datenelement ("DataElement"). Über Datenelemente werden Datentypinformationen bereitgestellt, die vom Datenmanager verwendet werden, wenn Daten für diesen Geschäftsobjekttyp abgerufen oder gespeichert werden. Beispiel:

```
<DataElement Name="EnquiryKey" LongName="Enquiry Key"
DataType="LONG" MaxLength="20" />
<DataElement Name="EnquiryDate" LongName="Enquiry Date"
DataType="DATE" />
<DataElement Name="ResponseDate" LongName="Response Date"
DataType="DATE" />
<DataElement Name="EnquiryContent" LongName="Enquiry content"
```

```
DataType="STRING" MaxLength="256" />
<DataElement Name="ResponseContent" LongName="Response content"
DataType="STRING" MaxLength="256" />
```

Beachten Sie, dass kein Datenelement ("DataElement") für E-Mail-Adresse und SKU angegeben wurde. Die Datenelemente für diese Datenfelder ("DataFields") sind bereits definiert und Sie können Datenelemente beliebig häufig wiederverwenden (vorausgesetzt, es handelt sich immer um denselben Datentyp).

6. Erstellen Sie für diese Data-Bean Einträge in der Datei **ObjectMap.xml**. Beispiel:

```
<Object ID="com.comergent.bean.simple.EnquiryBean">
<ClassName>com.comergent.bean.simple.EnquiryBean</ClassName>
</Object>
<Object ID="com.comergent.bean.simple.IRdEnquiry">
<ClassName>com.comergent.bean.simple.EnquiryBean</ClassName>
</Object>
<Object ID="com.comergent.bean.simple.IAccEnquiry">
<ClassName>com.comergent.bean.simple.EnquiryBean</ClassName>
</Object>
<Object ID="com.comergent.bean.simple.IDataEnquiry">
<ClassName>com.comergent.bean.simple.EnquiryBean</ClassName>
</Object>
```

7. Definieren Sie schließlich ein Datenquellenelement, das dem Attribut "DataSourceName" entspricht, das im Element "DataObject" definiert wurde. Diese Datenquelle wird in der Datei "DsDataSources.xml" als Teil des Schemas definiert. In den meisten Fällen ist diese Datenquelle bereits definiert. Sie müssen nur dann eine neue definieren, wenn Sie eine andere Datenbank oder eine andere Datenquelle als der Rest der Wissensdatenbank verwenden. Beispiel:

```
<DataSource Name="ENTERPRISE" Version="2.0">
<Primary Type="SQL" DataService="JdbcService"
SubType="ORACLE"
ConnectionString="jdbc:<driver>:<server>:<port>:<sid>"
UserId="userid" Password="password" />
<Alternate Type="SQL" DataService="JdbcService"
SubType="MSSQL"
ConnectionString="jdbc:<driver>:<server>:<port>:<sid>"
UserId="userid" Password="password" />
</DataSource>
```

Über das Attribut "DataService" der Elemente "Primary" und "Alternate" wird festgelegt, welche Klasse dazu verwendet wird, die Methoden *restore()* und *persist()* der Data-Bean "EnquiryBean" zu verarbeiten. Über die verbleibenden Attribute wird genau festgelegt, wie auf die externe Quelle zugegriffen wird.

8. Führen Sie das SDK-Zielelement "generateBean" aus, um den Quellcode für die neuen Data-Beans "EnquiryBean" und "EnquiryListBean" sowie die dazugehörigen Schnittstellen zu erstellen. Weitere Informationen zu diesen Schnittstellen finden Sie unter Kapitel 12, „Generierte Schnittstellen”, auf Seite 79.

Ergebnisse

Sie können nun "Enquiry"-Data-Beans und deren Schnittstellen in Geschäftslogikklassen verwenden. Möchten Sie eine Instanz einer "Enquiry"-Data-Bean erstellen, rufen Sie eine Methode im folgenden Format auf:

OMWrapper().getObject("com.comergent.bean.simple.EnquiryBean")

Dadurch werden eine "EnquiryBean"-Data-Bean und deren Struktur entsprechend der Angabe im Datenobjekt "Enquiry" zurückgegeben. Sobald Ihnen eine Instanz der "Enquiry"-Data-Bean vorliegt, können Sie deren Schlüsselfelder belegen und die Bean zum Abrufen der Daten wiederherstellen:

```
int queryIndex = 0;
try
{
String queryKey = request.getParameter("querykey");
queryIndex = Integer.parseInt(queryKey);
}
catch (Exception e)
{
//Throw exception if parameter not valid
}
QueryBean queryBean = (QueryBean)
OMWrapper().getObject("com.comergent.bean.simple.EnquiryBean");
queryBean.setKey(queryIndex);
queryBean.restore();
```

So rufen Sie eine Liste mit Abfragen ab:

```
// Use default settings for DataContext parameters
DataContext context = new DataContext();
// Retrieve enquiries relating to a particular product ID, MXWS-7000
DsQuery query =
QueryHelper.newWhereClause("SKU", DsQueryOperators.EQUALS,
"MXWS-7000");
EnquiryListBean enquiryList = (EnquiryListBean)
OMWrapper.getObject("com.comergent.bean.simple.EnquiryListBean");
// Restore the list.
enquiryList.restore(context, query);
// Walk through the list...
ListIterator enquiryIterator = enquiryList.getEnquiryIterator();
while (enquiryIterator.hasNext())
{
boolean isModified = false;
EnquiryBean enquiry = (EnquiryBean) enquiryIterator.next();
// Process each enquiry
}
```

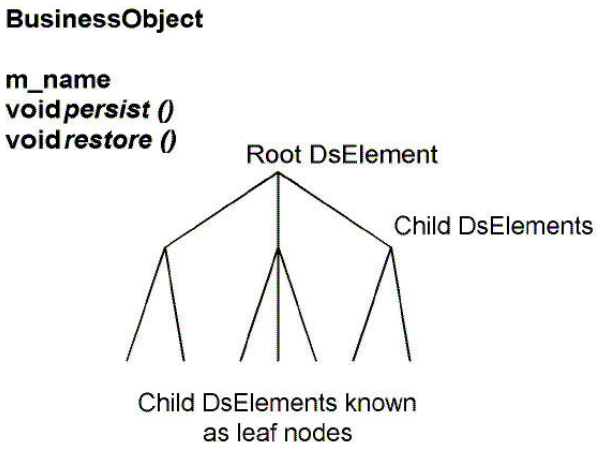
Generell sollten Sie immer versuchen sicherzustellen, dass Anwendungen, die mit der "EnquiryBean"-Data-Bean arbeiten, nicht die Data-Bean selbst, sondern eine der generierten Schnittstellen verwenden. So ist die Anwendung in der Lage, zwischen der Implementierung des Datenobjekts und seiner Schnittstelle zu unterscheiden, und Sie können steuern, welchen Zugriff die Anwendung auf die Daten des Objekts hat. Möchten Sie eine Instanz einer Klasse abrufen, über die die Klasse "IAccEnquiry" implementiert wird, gehen Sie wie folgt vor:

```
IAccEnquiry temp_IAccEnquiry = (IAccEnquiry)
OMWrapper.getObject("com.comergent.bean.simple.IAccEnquiry");
```

"DsElement"-Baumstruktur

Im vorliegenden Abschnitt werden Methoden zum Abrufen von Metadaten zu Data-Beans beschrieben. Außerdem wird die "DsElement"-Baumstruktur beschrieben, die zum Speichern von Daten in den Datenobjekt- und Geschäftsobjektklassen verwendet wird. Diese Beschreibung erfolgt nur im Hinblick auf die Unterstützung traditioneller Anwendungen: Alle neuen Anwendungen, die sich der Data-Bean-Klassen bedienen, sind davon nicht betroffen.

Datenobjekte werden als Objekte von Data-Bean-Klassen erstellt. Die Inhalte der Datenobjekte sind in Form einer Baumstruktur aus Komponenten mit Namen "DsElement" aufgebaut (siehe „"DsElement"-Komponenten"). Die Inhalte werden mithilfe des XML Schemas und der in dem XML Schema definierten Anleitungen und Datenquellen aus externen Systemen abgerufen. In der folgenden Abbildung wird ein Geschäftsobjekt dargestellt:

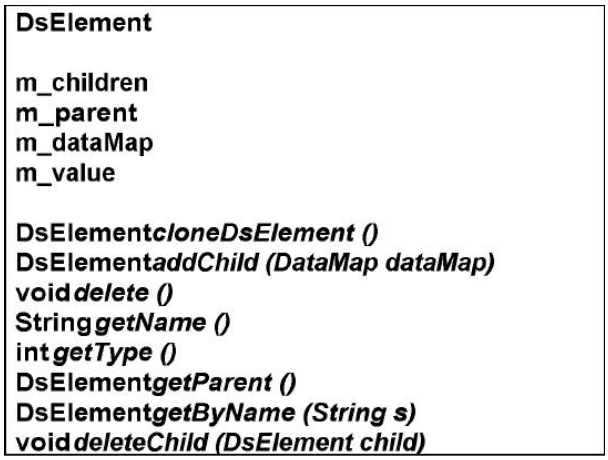


.

Wenn vom Datenmanager eine Data-Bean oder ein Geschäftsobjekt erstellt wird, wird das XML Schema verwendet, um die Struktur der entsprechenden "DsElement"-Baumstruktur zu bestimmen. Bei der "DsElement"-Baumstruktur handelt es sich um die Java-Darstellung der Struktur des Geschäftsobjekts. Über das Schema werden auch die Datentypen festgelegt, die an Endpunktknoten eingefügt werden können. Außerdem wird hier angegeben, ob für die Werte des Knotens bestimmte Einschränkungen gelten. Sie können auf die "DsElement"-Baumstruktur zugreifen, indem Sie die Methode *getRootElement()* aufrufen.

"DsElement"-Komponenten

Jede "DsElement"-Komponente enthält Daten sowie eine Datenzuordnung, über die definiert wird, wie die Daten der jeweiligen Datenquelle zugeordnet sind. Eine "DsElement"-Komponente kann untergeordnete Komponente einer anderen "DsElement"-Komponente (nämlich deren *übergeordneter Komponente*) sein. Bei einer "DsElement"-Baumstruktur handelt es sich um eine Gruppe von "DsElement"-Komponenten, die alle (bis auf eine) in der Baumstruktur über eine übergeordnete Komponente verfügen. Per Definition handelt es sich bei der "DsElement"-Komponente mit einer übergeordneten Komponente mit dem Wert null um die "DsElement"-*Stammkomponente*. In der folgenden Abbildung werden die Methoden für die "DsElement"-Komponenten dargestellt:



Über die Klasse "DsElement" werden verschiedene zusätzliche Methoden bereitgestellt, mit deren Hilfe das Navigieren durch eine "DsElement"-Baumstruktur unterstützt wird. Dabei geht es in erster Linie um die Methode *children()*, über die ein Iterator der untergeordneten "DsElement"-Komponenten einer bestimmten "DsElement"-Komponente ausgegeben wird. Neben der Methode *getRootElement()* wird über die Geschäftsobjektklasse auch die Methode *getElementByName()* bereitgestellt, mit der direkt auf eine benannte "DsElement"-Komponente in deren Baumstruktur zugegriffen werden kann.

Alle "DsElement"-Komponenten mit demselben Namen (z. B. "child_name"), die untergeordnete Komponenten einer "DsElement"-Komponente sind, müssen über eine übergeordnete Komponente mit Namen "<child_name>List" verfügen. Vom XML Schema werden solche Komponenten angegeben, indem deren Ordinalität mit “n” im Gegensatz zu “1” definiert wird. Eine "DsElement"-Komponente verwaltet ihre untergeordneten Komponenten in einem Vektor namens "m_children".

Die Klasse "DsElement" verfügt über die folgenden wichtigen Methoden:

- *addChild()*: Über diese Methode wird (entsprechend der Datenzuordnung für diese "DsElement"-Komponente) eine neue "DsElement"-Komponente hinzugefügt.
- *cloneDsElement()*: Über diese Methode wird eine Kopie dieser "DsElement"-Komponente ausgegeben.
- *delete()*: Über diese Methode wird der Status "DsElemState" in "DsElemState.DELETED" geändert.
- *deleteChild()*: Über diese Methode wird eine untergeordnete Komponente aus dem Vektor "m_children" entfernt, indem sie als "DsElement"-Komponente angegeben wird.
- *getName()*: Über diese Methode wird der Name der Komponente entsprechend der Definition in den Metadaten ausgegeben.
- *getParent()*: Über diese Methode wird die übergeordnete Komponente dieser "DsElement"-Komponente ausgegeben.
- *getType()*: Über diese Methode wird der Typ der Komponente entsprechend der Definition in der Datenzuordnung ausgegeben.

Metadaten der "DsElement"-Komponenten

Manchmal ist es sinnvoll, Informationen zu einem Datenfeld sowie dessen unterlegter "DsElement"-Komponente abzurufen. Dazu können Sie die "IData"-Schnittstellenmethode *getMetaData(String elementName)* verwenden. Über diese Methode wird ein Objekt ausgegeben, über das die Schnittstelle "IMetaData" implementiert wird. Mit dieser Schnittstelle werden die folgenden Methoden unterstützt:

- public int getDataType(): Über diese Methode werden Werte entsprechend der Definition in "DsDataTypes" ausgegeben.
- public long getMaxLength(): Über diese Methode wird die maximale Länge in Byte ausgegeben.
- public long getMaxCharLength(Locale locale): Über diese Methode wird die maximale Länge in Zeichen ausgegeben.
- public Object getMinValue(): Über diese Methode wird der zulässige Minimalwert (oder der Wert null, wenn kein Minimalwert vorhanden ist) ausgegeben.
- public Object getMaxValue(): Über diese Methode wird der zulässige Maximalwert (oder der Wert null, wenn kein Maximalwert vorhanden ist) ausgegeben.
- public int getCountAllowedValues()
- public ListIterator getAllowedValueIterator()
- public Object getDefaultValue()

Anmerkung: Von jeder generierten Data-Bean-Klasse wird die "IData"-Schnittstelle implementiert. Damit stehen diese Methoden allen generierten Data-Beans zur Verfügung.

Geschäftsobjektmethoden

Die Verwendung von Geschäftsobjekten wird nicht weiter unterstützt. Im vorliegenden Abschnitt finden Sie zwar Informationen zu einigen Geschäftsobjektmethoden, doch dienen diese nur zu Referenzzwecken.

Geschäftsobjektmethode "restore()"

In diesem Abschnitt finden Sie eine Beschreibung der wichtigsten Formate der Geschäftsobjektmethode *restore()*.

```
public void restore(BusinessObject queryObj, int maxResults,
boolean accessCheck)
```

Dies ist das wichtigste Format der Methode *restore()*. Verwenden Sie den Parameter "queryObj", um die Abfrage anzugeben, die über die Wiederherstellungsoperation ausgeführt werden soll. Über den Parameter "maxResults" wird die maximale Anzahl zurückzuliefernder Objekte bestimmt. Verwenden Sie den Parameter "accessCheck", um anzugeben, ob überprüft werden soll, ob der aktuelle Benutzer über die zur Ausführung dieser Operation erforderlichen Berechtigungen verfügt. Sobald die Zugriffsprüfung ausgeführt ist, wird *restore(BusinessObject queryObj, int maxResults)* aufgerufen.

public void restore(BusinessObject queryObj, int maxResults)

Über diese Methode wird die *restore()*-Methode *restore(this, queryObj, maxResults, false)* des unterlegten Datenobjekts aufgerufen.

public void restore(BusinessObject queryObj)

Dies ist äquivalent zum Aufruf *restore(queryObj, 0)*.

```
public void restore()
```

Über dieses Format der Methode wird die Methode *restore(null, 0)* aufgerufen.

Geschäftsobjektmethode "persist()"

In diesem Abschnitt finden Sie Beschreibungen der wichtigsten Formate der Geschäftsobjektmethode *persist()*.

```
public void persist(boolean synch, boolean commit,
boolean accessCheck)
```

Über die booleschen Parameter wird definiert, ob die Operation zum "Als persistent definieren" synchronisiert und in der zugrunde liegende Datenquelle festgeschrieben werden soll und ob vor Ausführung der Operation eine Zugriffsprüfung ausgeführt werden soll.

```
public void persist(boolean synch, boolean commit)
```

Dieses Format der Methode ist äquivalent zu *persist(synch, commit, false)* und bezieht sich auf Geschäftsobjekte, deren Attribut "Version" den Wert "4.0" oder niedriger führt. Es ist äquivalent zu *persist(synch, commit, true)* und bezieht sich auf Geschäftsobjekte, deren Attribut "Version" den Wert "5.0" oder höher führt.

```
public void persist()
```

Über dieses Format der Methode wird *persist(false, true)* aufgerufen.

Die Klasse "BusinessObject" verfügt auch über die folgenden Methoden:

- *delete()*: Über diese Methode wird das Geschäftsobjekt durch Löschen seiner "DsElement"-Baumstruktur geleert.
- *getRootElement()*: Über diese Methode wird die "DsElement"-Stammkomponente der "DsElement"-Baumstruktur ausgegeben.
- *getType()*: Über diese Methode wird der Name der "DsElement"-Stammkomponente der "DsElement"-Baumstruktur ausgegeben. Dabei handelt es sich um den Typ des Geschäftsobjekts.
- *setRootElement()*: Über diese Methode wird die Stammkomponente dieses Geschäftsobjekts definiert.

Kapitel 9. In Visual Modeler protokollieren

In Visual Modeler protokollieren: Eine Übersicht

Über den Protokollierungsmechanismus von Visual Modeler können Anwendungsprogrammierer die Aktivitäten in Visual Modeler erfassen. Zum Konfigurieren des Protokollierungsverhaltens kommen dabei die "log4j"-API sowie entsprechende **log4j.properties**-Konfigurationsdateien zum Einsatz. Die Protokollierungsfunktion bietet außerdem Unterstützung für die Überwachung der an Datenobjekten vorgenommenen Änderungen. Weitere Informationen hierzu finden Sie unter „Änderungen an Datenobjekten protokollieren” auf Seite 70.

Über die "log4j"-API wird ein flexibles und erweiterbares Protokollierungsframework zum Steuern des Protokollierungsverhaltens von Visual Modeler bereitgestellt. Im vorliegenden Abschnitt wird der Umgang mit diesem Framework beim Anpassen und Erweitern von Visual Modeler beschrieben.

Beachten Sie, dass durch dieses Framework das bisher von Visual Modeler verwendete Framework ersetzt wird. In dem bisher verwendeten Framework kamen die Klasse "Global" und deren *log*Level*()*-Methoden zum Einsatz. Deren Verwendung wird allerdings nicht weiter unterstützt.

Für die Verwendung der "log4j"-API sollten Sie in jeder Klassendatei in den folgenden Zeilen eine "Logger"-Klasse einrichten:

```
private static final org.apache.log4j.Logger log =
org.apache.log4j.Logger.getLogger(NameOfClass.class);
```

Wenn Sie dann einen Protokolleintrag aufrufen möchten, gehen Sie wie folgt vor:

```
log.info("This is a log entry");
```

Welche Methode Sie aufrufen, hängt von der Protokollierungsstufe ab, auf der Sie die Nachricht erfassen möchten. Die folgenden Methoden stehen zur Auswahl:

- *debug()*
- *error()*
- *fatal()*
- *info()*
- *warning()*

Sie können auch die Methode *log(priority, message)* verwenden, aber gewöhnlich sollten die aufgeführten Methoden vollkommen ausreichen.

Systemeigenschaft "log4j.debug"

Wenn Sie für die Systemeigenschaft "log4j.debug" den Wert "true" angeben, können Sie sich die folgenden Protokolleinstellungen zurückmelden lassen. Beispiel: Nehmen Sie im Servlet-Container-Startscript folgende Eingabe vor:

```
-Dlog4j.debug=true
Daraufhin sollte die Protokollausgabe nach dem Start folgendes Format aufweisen: log4j: Trying to
sun.misc.Launcher$AppClassLoader@136228.
log4j: Trying to find [log4j.xml] using sun.misc.Launcher$AppClassLoader@136228 class loader.
log4j: Trying to find [log4j.xml] using ClassLoader.getSystemResource().
```

```
log4j: Trying to find [log4j.properties] using context classloader
sun.misc.Launcher$AppClassLoader@136228.
log4j: Using URL [jar:file:/home/hle/ws/32-cmgt-modules/modules.cryptography-
tool/target/cmgt-cryptography-tool-2.0.0-SNAPSHOT-app.jar!/log4j.properties]
for automatic log4j configuration.
log4j: Reading configuration from URL jar:file:/home/hle/ws/32-cmgt-modules/modules.
cryptography-tool/target/cmgt-cryptography-tool-2.0.0-SNAPSHOT-app.jar!/log4j.properties
log4j: Parsing for [root] with value=[WARN, A1].
log4j: Level token is [WARN].
log4j: Category root set to WARN
log4j: Parsing appender named "A1".
log4j: Parsing layout options for "A1".
log4j: Setting property [conversionPattern] to [%-4r [%t] %-5p %c %x - %m%n].
log4j: End of parsing for "A1".
log4j: Parsed "A1" options.
log4j: Finished configuring.
```

Änderungen an Datenobjekten protokollieren

In vielen Implementierungen empfiehlt es sich, eine Protokollaufzeichnung einzurichten, in deren Rahmen Änderungen an Daten in Visual Modeler protokolliert werden. Dazu können Sie alle Änderungen, die an Datenobjekten vorgenommen werden, erfassen lassen. Wenn Sie die Protokollierungsstufe in einer beliebigen Data-Bean-Klasse auf "INFO" oder höher einstellen, wird immer dann, wenn für eine Instanz dieser Klasse die Methode *persist()* aufgerufen wird, für die Klasse eine Protokollnachricht in den Logger geschrieben. Bei den folgenden Angaben handelt es sich um ein Beispiel für Zeilen, die ausgegeben werden, wenn für einen Partner eine Änderung vorgenommen wird:

```
2006.01.18 13:41:05:546 Env/http-8080-Processor23:INFO:PartnerBean Updating:
com.comergent.bean.simple.PartnerBean KeyFields - PartnerKey: 301 Changes -PartnerKey ->
old: 301 new: 301PartnerName -> old: Scalar2 new: Scalar2 LegalName ->
old: null new: null ParentCompany -> old: null new: nullStatus ->
old: A new: A DunBradID -> old: null new: nullBusinessID ->
old: Scalar2-001 new: Scalar2-001PartnerTypeCode -> old: 10 new: 10PartnerLevelCode ->
old: 20 new: 20XMLMessageVersion -> old: dXML 4.0 new: dXML 4.0BusinessTransaction ->
old: SELL new: SELL NetWorth -> old: null new: null NumEmployees ->
old: null new: null PotRevCurrFy -> old: null new: null PotRevNextFy ->
old: null new: null ReferenceUseFlag -> old: null new: null CotermDayMonth ->
old: null new: nullURL -> old: http:///www.scalar.com new: http:///www.scalar2.com LogoURL ->
old: null new: null DistiAccess -> old: null new: null YearEstd -> old: null new:
null AnalysisFy -> old: null new: null FyEndMonthCode -> old: null new: null AccountManagerKey ->
old: null new: null MessageURL -> old: null new: null EmailAddress ->
old: null new: nullCommerceCategory -> old: 2 new: 2 PartnerRefNum ->
old: null new: null ParentKey -> old: null new: null RootPartnerKey ->
old: null new: null ParentCode -> old: null new: null CustomField1 ->
old: null new: null CustomField2 -> old: null new: null CustomField3 ->
old: null new: null CustomField4 -> old: null new: null CustomField5 ->
old: null new: null PartnerCom -> old: null new: null Storefront ->
old: null new: null URLName -> old: null new: null ContentType ->
old: null new: nullPartnerStatusCode -> old: 10 new: 10OrganizationType ->
old: DirectPartner new: DirectPartner InheritedPartnerStatusCode ->
old: null new: nullCreditLimit -> old: 0.0000 new: 0.00AvailableCredit ->
old: 0.0000 new: 0.0000CreditCurrencyCode -> old: 23 new: 23 MaxAssignableReps ->
old: null new: null RemotePrices -> old: null new: null RemotePriceExpiryInterval ->
old: null new: nullCoopPercentage -> old: 0.000000 new: 0.000CoopAccountMax ->
old: 0.000000 new: 0.00 PartnerID -> old: null new: nullOwnedBy ->
old: 0 new: 0AccessKey -> old: 5601 new: 5601UpdateDate ->
old: 2006-01-18 13:39:33.0 new: 2006-01-18 13:41:05.484UpdatedBy ->
old: 0 new: 0CreateDate -> old: 2006-01-04 13:19:38.0 new: 2006-01-04 13:19:38.0CreatedBy ->
old: 0 new: 0
```

Sie können die Protokollierungsstufe für jede beliebige Klasse in Visual Modeler dynamisch über die Verwaltungsbenutzerschnittstelle ändern. In diesem Falle ist

die Änderung der Protokollierungsstufe dann allerdings nicht persistent und geht nach dem Neustart des Servlet-Containers wieder verloren. Außerdem wird die Protokollierung in den standardmäßigen Appender geschrieben, der nicht notwendigerweise sicher sein muss.

Sie sollten die Durchführung von Protokollaufzeichnungen durch Anpassen der Konfigurationsdatei **log4j.properties** angeben. So können Sie sicherstellen, dass die Protokollierung auch dann weiterhin ausgeführt wird, wenn der Servlet-Container neu gestartet wurde. Außerdem können Sie einen angepassten Appender angeben, über den die Protokolldaten verarbeitet werden können. Beispielsweise können Sie angeben, dass die Protokollnachricht vom Appender auf einen fernen Web-Server übertragen werden soll, der unabhängig von Visual Modeler gesichert werden kann.

Zum Beispiel wird über die folgenden Einträge in der Konfigurationsdatei **log4j.properties** sichergestellt, dass alle Änderungen am Datenobjekt "UserContact" protokolliert werden:

```
log4j.logger.com.comergent.bean.simple.UserContactBean=info
log4j.appender.com.comergent.bean.simple.
UserContactBean=com.comergent.logging.ComergentRollingFileAppender
log4j.appender.com.comergent.bean.simple.
UserContactBean.layout = org.apache.log4j.PatternLayout
```

Wenn Sie angeben möchten, dass ein ferner Protokollserver als Client angeschlossen werden kann, um Protokolldaten aus Visual Modeler zu speichern, nehmen Sie die folgenden Eingaben vor:

```
log4j.appender.com.comergent.bean.simple.UserContactBean=org.apache.log4j.net.
SocketHubAppender
log4j.appender.com.comergent.bean.simple.UserContactBean.port=4321
```

Kapitel 10. Modularität und generierte Schnittstellen

Visual Modeler verfügt über die folgenden Features, die dafür vorgesehen sind, Implementierungen leichter anpassen und aktualisieren zu können:

- Module
- Generierte Schnittstellen

Diese Features hängen insofern miteinander zusammen, als die Schnittstellen nach Modulen organisiert sind und Änderungen an Schnittstellen bereits in den Änderungen an einzelnen Modulen enthalten sein können.

Dank der Bereitstellung von Funktionalität in Modulen sowie der Festlegung, dass Module andere Module nur über die entsprechenden externen Schnittstellen aufrufen können, kommen Sie in den Genuss folgender Vorteile:

- Es ist einfacher, die Funktionalität von Anwendungen aufzugliedern.
- Es ist einfacher, die Abhängigkeiten zwischen den verschiedenen Teilen von Visual Modeler zu verstehen und zu steuern.
- Es ist einfacher, Anpassungen an einzelnen Modulen vorzunehmen und dabei die Auswirkungen der Änderungen in einem bestimmten Modul auf das komplette System einzuschätzen.
- Für die Module können unabhängig voneinander leichter Upgrades durchgeführt werden, wodurch sich die Effekte, die von einem Upgrade ausgehen können, minimieren lassen.
- Upgrades für Module, die nicht angepasst wurden, haben keinerlei Auswirkungen auf die Anpassungen in anderen Modulen.
- Neue Funktionen können in Form eines Moduls bereitgestellt werden, das dann in die aktuelle Implementierung von Visual Modeler übernommen wird.

Kapitel 11. Module in Visual Modeler

Visual Modeler-Module: Eine Übersicht

Visual Modeler wurde als Gruppe voneinander abhängiger Module entwickelt, die einer gemeinsamen organisatorischen Struktur entsprechen. Gewöhnlich entspricht jedes Modul einer funktionalen Komponente von Visual Modeler (z. B. einer Anwendung) oder einer Komponente der Visual Modeler-Plattform. Einige Module bieten möglicherweise Unterstützung sowohl für eine Java-API als auch für eine Benutzerschnittstelle, während von anderen nur eine Java-API zu anderen Modulen unterstützt wird. In einigen Modulen wird außerdem eine Anzahl von “Hilfeprogramm”-Klassen, JSP-Seiten und anderen Dateien (wie JavaScript-Dateien und Bildern) bereitgestellt, die auch von anderen Modulen verwendet werden.

Gewöhnlich weist jedes Modul die folgende Struktur auf:

- Java-Klassen: Sind in drei Baumstrukturen organisiert. Zur Erstellungszeit werden die Verzeichnisse für alle Module in einer einzigen Baumstruktur unter dem Paket "com.comergent" zusammengefasst.
  - Externe API-Schnittstellen: Werden von anderen Modulen für den Zugriff auf die in dem Modul bereitgestellten Funktionen verwendet. Wenn von einem Modul die Klasse eines anderen Moduls aufgerufen wird, muss dieser Aufruf gewöhnlich über die externe API des anderen Moduls erfolgen. Dafür steht das Paket "com.comergent.api" für das Modul zur Verfügung. Außerdem wird "com.comergent.appservices.appServiceUtils.OFApiHelper" dazu verwendet, die Sterling Selling and Fulfillment Foundation X-APIs aufzurufen.
  - Implementierungsklassen: Dabei geht es um die Implementierung der externen API-Schnittstellen. Wenn die externe API des Moduls von einem anderen Modul aufgerufen wird, fungieren die aktuell verwendeten Klassen als Implementierungsklassen der Modulschnittstelle. In den Implementierungspaketen können auch interne Klassen enthalten sein. Diese werden zwar von den Implementierungsklassen verwendet, sie werden aber nicht extern verfügbar gemacht und sind nicht Bestandteil des unterstützten Javadocs. Dafür steht das Paket "com.comergent.apps" oder "com.comergent.appservices" für das Modul zur Verfügung.
  - Referenzkomponenten: Controllerklassen und JSP-Seiten sind grundsätzlich Teile der Referenzimplementierung und deren Quelle wird zusammen mit Visual Modeler bereitgestellt. Ressourcenpakete werden ebenfalls als Teil der Referenz bereitgestellt. Dafür steht das Paket "com.comergent.reference" für das Modul zur Verfügung.
- JSP-Seiten: Sind je nach Organisation des Moduls möglicherweise in Verzeichnissen organisiert. Sollten grundsätzlich über die von den anderen Modulen bereitgestellten externen APIs auf die Klassen der anderen Module zugreifen. Auf diese Weise wird sichergestellt, dass JSP-Seiten releaseübergreifend verwendet werden können (Voraussetzung: die externen APIs werden unterstützt).
- Ressourcenpakete, Javascripts und statische Dateien (wie Bilder und HTML-Fragmente).
- Konfigurationsdateien speziell für das Modul (wie **MessageTypes.xml**-Dateien) sowie Geschäftsregeln.

Modulschnittstellen

Jedes Modul muss über eine externe Schnittstelle verfügen, sodass alle Aufrufe von Java-Klassen und -Schnittstellen innerhalb des Moduls über diese Schnittstelle erfolgen können. Über diese externe Schnittstelle wird eine umfassende Anzahl Javadoc-Seiten bereitgestellt, sodass die Autoren anderer Module die externe Schnittstelle zuverlässig und ohne großen Aufwand verwenden können.

Bei den externen Schnittstellen der einzelnen Module handelt es sich typischerweise um eine Kombination aus manuell erstellten und automatisch generierten Schnittstellen. In den meisten Modulen werden manuell erstellte Schnittstellen für Presentation-Beans bereitgestellt, über die die Presentation-Beans in die Lage versetzt werden, Daten über die einfachen Zugriffsmethoden der generierten Data-Bean-Schnittstellen hinaus zu bearbeiten. In den Presentation-Beans ist gewöhnlich eine Data-Bean eingeschlossen und es werden dieselben Schnittstellen implementiert, aber zusätzlich werden auch noch Unterstützungsmethoden und etwas Geschäftslogik implementiert.

Die externen Schnittstellen sind in den folgenden Hauptpaketen organisiert:

- com.comergent.api: Dieses Paket enthält alle externen APIs des Moduls. Dabei wird unterschieden zwischen:
  - apps: Dies sind die manuell erstellten APIs der Anwendung. Gewöhnlich handelt es sich dabei um Presentation-Bean-Schnittstellen, Dienstprogrammschnittstellen und Factory-Klassen.
  - appservices: Dies sind die Pakete, die über die von anderen Anwendungen verwendeten Servicemodule bereitgestellt werden.
  - dcm: Dies sind die externen APIs, die über die Visual Modeler-Plattform bereitgestellt werden.
- com.comergent.bean.simple: Dieses Paket enthält alle automatisch generierten Bean-Schnittstellen sowie die Data-Bean-Klassen selbst.

Die generierten Schnittstellen werden für alle in den XML Schemadateien deklarierten Datenobjekte bereitgestellt. Sie sind so organisiert, dass entsprechende Zugriffsstufen für die Datenfelder der unterlegten Data-Beans bereitgestellt werden. So kann sichergestellt werden, dass sauber zwischen Darstellung und Geschäftslogik in Visual Modeler getrennt werden kann. Weitere Informationen zu den generierten Schnittstellen finden Sie unter Kapitel 12, „Generierte Schnittstellen”, auf Seite 79.

Schnittstellen aufrufen

Sie können eine Schnittstelle über eine Java-Klasse aufrufen, indem Sie ein beliebiges Objekt oder eine beliebige untergeordnete Schnittstelle für die Schnittstelle umwandeln und dann eine von der Schnittstelle deklarierte Methode aufrufen. In Visual Modeler können Sie dazu eine der folgenden Verfahren verwenden:

- „Klasse "ObjectManager" verwenden” auf Seite 77
- „Factory-Klassen verwenden” auf Seite 77

Jedes Modul arbeitet entweder mit dem einen oder mit dem anderen Verfahren, aber nicht mit beiden. Sorgen Sie beim Umgang mit einem vorhandenen Modul oder beim Erstellen eines neuen Moduls hinsichtlich des Aufrufs der Schnittstellen für Konsistenz. Damit erleichtern Sie Ihren Kollegen die Arbeit am selben Modul.

Grundsätzlich sollten Sie immer versuchen, mit den in dem Paket "com.comergent.api" enthaltenen Schnittstellen zu arbeiten. Dies sind die Schnittstellen, die auch dann von Release zu Release von den Modulen unterstützt werden, wenn sich die jeweiligen Implementierungen der Schnittstellen ändern.

Klasse "ObjectManager" verwenden

Mithilfe der Klasse "ObjectManager" können Sie wie folgt eine passende Schnittstelle ausgeben lassen. Angenommen, Sie möchten die Schnittstelle "IAccProduct" abrufen, um die Datenfelder für ein Produkt zu definieren. Führen Sie dazu einen Aufruf entsprechend den folgenden Zeilen aus:

```
IAccProduct temp_IAccProduct =

(com.comergent.bean.simple.IAccProduct)
com.comergent.dcm.util.OMWrapper.getObject(
  "com.comergent.bean.simple.IAccProduct");
```

Vorausgesetzt, es gibt einen Eintrag in der Datei **ObjectMap.xml**, in dem das auszugebende Objekt angegeben ist und vorausgesetzt, dieses Objekt implementiert die Schnittstelle "IAccProduct", dann ist dieser Aufruf erfolgreich und die Methoden zur Schnittstelle können aufgerufen werden. Beispiel: Die Datei **ObjectMap.xml** enthält den folgenden Eintrag:

```
<Object ID="com.comergent.bean.simple.IAccProduct">
<ClassName>com.comergent.bean.simple.ProductBean</ClassName>
```

In diesem Fall wird über die Klasse "ObjectManager" ein "com.comergent.bean.simple.ProductBean"-Objekt ausgegeben, das über die "IAccProduct"-Schnittstelle umgewandelt werden kann, da von der "com.comergent.bean.simple.ProductBean"-Klasse die Schnittstelle "com.comergent.bean.simple.IAccProduct" implementiert wird.

Factory-Klassen verwenden

Das Aufrufen einer Schnittstelle kann über Factory-Klassen erfolgen, über die eine Instanz der Schnittstelle ausgegeben wird. Beispielsweise wird über das Paket "com.comergent.api.apps.commerce" eine allgemein zugängliche Schnittstelle "IInquiryListFactory" bereitgestellt. Wenn von einem anderen Modul eine Instanz dieser Factory-Schnittstelle benötigt wird, wird die Methode *getFactory(int i)* der Klasse "CommerceAPI" aufgerufen. Über den Parameter "int" wird festgelegt, welcher Typ Factory-Klasse ausgegeben werden soll. Entsprechend kann das aufrufende Modul nun Methoden zur Schnittstelle "IInquiryListFactory" aufrufen, um Anfragelistenschnittstellen des passenden Typs ausgeben zu lassen. Beispiel: Über *getInquiryList(Long listKey, boolean bFillPrices)* wird ein Objekt ausgegeben, durch das die Schnittstelle "IInquiryList" implementiert wird.

Kapitel 12. Generierte Schnittstellen

Wenn Sie auf Daten in einem bestimmten Datenobjekt zugreifen müssen, verwenden Sie zu diesem Zweck die von den verschiedenen Datenobjekten bereitgestellten generierten Schnittstellen. Diese generierten Schnittstellen werden erstellt und kompiliert, wenn das SDK-Zielelement "generateBean" als Teil der Implementierung von Visual Modeler ausgeführt wird.

Für jedes innerhalb der Datei **DsRecipes.xml** als "DataObject" deklarierte Datenobjekt sowie für alle Referenz-, übergeordneten oder untergeordneten Datenobjekte werden die folgenden Klassen und Schnittstellen im Paket "com.comergent.bean.simple" generiert und deklariert:

- <*name*>.java: Dies ist die Data-Bean-Klasse. Über sie werden die hier aufgeführten Schnittstellen implementiert. Wenn durch das jeweilige Datenobjekt ein anderes Datenobjekt erweitert wird, wird über die Bean außerdem die "<*übergeordnetes_element*>.java"-Bean erweitert.
- IAcc<*name*>.java: Über diese Schnittstelle wird die Schnittstelle "IRd<*name*>.java" dahingehend erweitert, dass für alle Datenfelder des Datenobjekts die Zugriffsmethoden zum Schreiben (Definieren) bereitgestellt werden. Wenn durch das jeweilige Datenobjekt ein anderes Datenobjekt erweitert wird, wird über die "IAcc"-Schnittstelle außerdem die "IAcc<*übergeordnetes_element*>.java"-Schnittstelle erweitert.
- IData<*name*>.java: Über diese Schnittstelle wird die Schnittstelle "IAcc<*name*>.java" dahingehend erweitert, dass die Methoden *restore()* und *persist()* für das Datenobjekt bereitgestellt werden. Wenn durch das jeweilige Datenobjekt ein anderes Datenobjekt erweitert wird, wird über die "IData"-Schnittstelle außerdem die "IData<*übergeordnetes_element*>.java"-Schnittstelle erweitert.
- IRd<*name*>.java: Über diese Schnittstelle werden für alle Datenfelder des Datenobjekts die schreibgeschützten (ausschließlich zum Abrufen zu verwendenden) Zugriffsmethoden bereitgestellt. Wenn durch das jeweilige Datenobjekt ein anderes Datenobjekt erweitert wird, wird über die "IRd"-Schnittstelle außerdem die "IRd<*übergeordnetes_element*>.java"-Schnittstelle erweitert.
- Zusätzlich wird die "IData<*name*>List.java"-Schnittstelle auch über Listenbeans implementiert. Über jede Listenschnittstelle wird sowohl die "IDataList.java"-Schnittstelle als auch die Listenschnittstelle eines beliebigen übergeordneten Objekts erweitert.

Im Allgemeinen sollten Sie die "IRd"-Schnittstelle für alle Objekte verwenden, die an JSP-Seiten übergeben werden sollen. Damit werden diese Objekte faktisch schreibgeschützt. Verwenden Sie Objekte, durch die die "IData"-Schnittstelle implementiert wird, nur dann, wenn klar ist, dass das betreffende Datenobjekt entweder wiederhergestellt oder als persistent definiert werden muss.

Beispiel für eine generierte Schnittstelle

Stellen Sie sich das Datenobjekt "ACL" in der Datei **ACL.xml** vor:

```
<?xml version="1.0"?>
<DataObject Name="ACL" Extends="C3PrimaryRW"
ExternalName="CMGT_ACLS"
```

```
Access="RWID" Ordinality="1"
ObjectType="JDBC" Version="5.0">
<KeyFields>
<KeyField Name="AccessKey" ExternalName="ACL_KEY"
KeyGenerator="ACLKey"/>
</KeyFields>
<DataFieldList>
<DataField Name="AccessKey"
Writable="n" Mandatory="n"
ExternalFieldName="ACL_KEY"/>
<DataField Name="ACLName"
Writable="y" Mandatory="n"
ExternalFieldName="NAME"/>
</DataFieldList>
<ChildDataObject Name="Access" />
</DataObject>
```

Folglich wird in der Klasse "IRdACL.java" wie folgt deklariert:

```
public interface IRdACL extends IRdC3PrimaryRW
```

Anschließend werden die folgenden Methoden verfügbar gemacht:

- public Long getAccessKey();
- public String getACLName();

In der Klasse "IAccACL.java" wird wie folgt deklariert:

```
public interface IAccACL extends IAccC3PrimaryRW, IRdACL
```

Anschließend werden die folgenden Methoden verfügbar gemacht:

- public void setACLName(String value) throws ICCException;
- public void addAccess(AccessBean bean) throws ICCException;

In der Klasse "IDataACL.java" wird wie folgt deklariert:

```
public interface IDataACL extends IAccACL,IDataC3PrimaryRW, IData
```

Gewöhnlich werden über diese Schnittstelle neben den über die Schnittstelle "IData" deklarierten Methoden keine weiteren deklariert, da alle zum Lesen und Schreiben von Daten aus externen Datenquellen erforderlichen Standardmethoden bereits in dieser Schnittstelle deklariert werden.

Kapitel 13. Logische Klassen in Visual Modeler

Logische Klassen implementieren

Im vorliegenden Abschnitt sowie in den beiden folgenden Abschnitten wird beschrieben, wie im Rahmen der Implementierung von Visual Modeler Geschäftslogikklassen (Business Logic Classes, BLCs) implementiert werden. Bevor Sie sich jedoch mit dem vorliegenden Abschnitt beschäftigen, sollten Sie sich die erforderlichen Kenntnisse hinsichtlich der grundlegenden Architektur von Visual Modeler und Java angeeignet haben.

Anmerkung: Die Verwendung von BLCs wird nicht weiter unterstützt. In neuen Anwendungen wird zum Implementieren der erforderlichen Geschäftslogik gewöhnlich mit Bizlets, Controllern und BizAPIs gearbeitet.

Schlüsselkonzepte bei logischen Klassen

Um wirklich verstehen zu können, wie Visual Modeler als Anwendung arbeitet, müssen Sie dessen Architektur begreifen.

Visual Modeler verarbeitet sowohl Anforderungen, die von den Browsern von Benutzern empfangen werden als auch Nachrichten von anderen Visual Modeler-Installationen und externen Systemen. Sie müssen Visual Modeler für die Verarbeitung der verschiedenen Typen von Anforderungen und Nachrichten entsprechend konfigurieren.

Zentraler Bestandteil von Visual Modeler ist der Manager. Aufgabe dieses leistungsfähigen und flexiblen Servers ist die nahtlose Integration eines Netzes aus Channel-Partnern und externen Systemen, aus denen sich dann die E-Commerce-Umgebung der betreffenden Partner zusammensetzt.

Jeder Visual Modeler-Server im Netz von Vertriebspartnern fungiert gleichzeitig als Server für die ankommenden Anforderungen von Browsern als auch als Client für den Abruf von Informationen von anderen Visual Modeler-Servern sowie externen Systemen.

Möchten Sie Visual Modeler an Ihre Umgebung anpassen, müssen Sie berücksichtigen, wie das System Daten von externen Systemen abruft. Gewöhnlich können Sie Schema und Serviceklassen verwenden, um durch Austauschen von Nachrichten Daten aus einer lokalen Datenbankquelle oder von einem anderen Visual Modeler-Server abzurufen. Allerdings müssen Sie benutzerdefinierte Geschäftslogikklassen (Business Logic Classes, BLCs) erstellen, wenn Informationen von einem anderen externen System als diesem abgerufen werden sollen.

Anwendungslogikklassen

Anwendungslogikklassen werden als "bizAPI"-, Geschäftslogik- oder Controller-Klassen implementiert.

- "bizAPI"-Klassen werden dazu verwendet, die Geschäftslogik von Geschäftsobjekten zu steuern. Konzeptionell entspricht jede "bizAPI"-Klasse einem Geschäftsobjekt und die dazugehörigen Methoden entsprechen den Aktionen, die für das Geschäftsobjekt ausgeführt werden können. So werden über die "bizAPI"-Klasse "OrderInquiryList" z. B. die folgenden Methoden

bereitgestellt: *duplicate()*, *copyLineItem()* und *changeOwner()*. Diese Methoden entsprechen Aktionen, die für eine Produktanfrageliste ausgeführt werden können. Über diese Klasse wird die Schnittstelle "com.comergent.api.apps.orderMgmt.oil.IOrderInquiryList" implementiert.

Die "bizAPI"-Klassen werden in den "com.comergent.apps.<*anwendung*>.bizAPI"-Paketen definiert. Gewöhnlich wird durch sie eine im entsprechenden Paket "com.comergent.api.apps.<*anwendung*>" deklarierte Schnittstelle implementiert.

Beispiel: Die "bizAPI"-Klasse "Order" ist im Paket "com.comergent.apps.orderMgmt.orders.bizAPI" enthalten. Durch sie wird die Klasse "OrderInquiryList" erweitert und die Schnittstelle "com.comergent.api.apps.orderMgmt.orders.IOrder" implementiert.

- Jede Geschäftslogikklasse (Business Logic Class, BLC) bildet eine Unterklasse der abstrakten Geschäftslogikklasse. Über diese Klasse wird die Schnittstelle "ApplicationObject" implementiert. Über BLCs wird die Geschäftslogik Ihrer Implementierung von Visual Modeler ausgeführt. Jede BLC enthält eine Tabelle mit Geschäftsobjekten wie z. B. Sitzung, Benutzer und Einkaufswagen. Durch Ausführen der Methode *service()* einer BLC werden die Methoden *persist()* und *restore()* für diese Geschäftsobjekte aufgerufen.

  Anmerkung: Im Allgemeinen wird die Verwendung von Geschäftslogikklassen nicht weiter unterstützt. Stattdessen sollten Sie entweder Controller- oder "bizAPI"-Klassen zur Steuerung der Geschäftslogik verwenden.
- In einigen Visual Modeler-Installationen werden zur Ausführung der Geschäftslogik Controllerklassen verwendet. Diese Klassen befinden sich in den "com.comergent.reference.apps.<*anwendung*>.controller"-Paketen der Anwendungen.

Visual Modeler wird mit einer Anzahl standardmäßiger "bizAPI"-Klassen, BLCs, Controller und JSP-Seiten ausgeliefert. Trotzdem müssen Sie möglicherweise neue logische Klassen erstellen oder bereits vorhandene ändern.

XML Schema

Führen Sie den Datenzugriff über das Schema und die Serviceklassen aus.

Namensservice

Für den Abruf von Parametern zur Laufzeit wird von Visual Modeler ein Namensservice bereitgestellt, über den auf eine unstrukturierte Datei oder auf eine Datenbank zugegriffen werden kann, um Parameter wiederherzustellen.

Über Anwendungslogikklassen kann ein Namensservice aufgerufen werden, indem die "NamingManager"-Methoden *getInstance()* und *getInstance(int i)* für statische Klassen aufgerufen werden. Bei beiden Methoden wird ein Objekt zurückgegeben, das die Schnittstelle "NamingService" implementiert.

- Wenn kein ganzzahliges Argument bereitgestellt wird, wird ein Objekt entsprechend dem Standardtyp erstellt (entweder ein Objekt des Typs "NamingServiceProperties" oder ein Objekt des Typs "NamingServiceDatabase").
- Wenn es sich bei dem ganzzahligen Argument um die Konstante "NamingManager.DATABASE" handelt, wird ein "NamingServiceDatabase"-Objekt erstellt.
- Wenn es sich bei dem ganzzahligen Argument um die Konstante "NamingManager.PROPERTIES" handelt, wird ein "NamingServiceProperties"-Objekt erstellt.

- Wenn es sich bei dem ganzzahligen Argument weder um die eine noch um die andere Konstante handelt, wird ein Objekt entsprechend dem Standardtyp erstellt.

In allen Fällen greift Visual Modeler auf die Datei **Comergent.xml** zu, um genau zu bestimmen, wie das "NamingService"-Objekt erstellt werden soll:

- Soll ein "NamingServiceDatabase"-Objekt erstellt werden, werden zur Angabe der Verbindung zur Datenbank die "NamingManager.database"-Einträge verwendet.
- Soll ein "NamingServiceProperties"-Objekt erstellt werden, wird über den Eintrag "NamingManager.properties" bestimmt, in welcher Eigenschaftendatei sich die Parameterwerte befinden.

Sobald das "NamingService"-Objekt erstellt ist, können Sie die unten aufgeführten Methoden verwenden, um die Parameter als "NamingResult"-Klasse abzurufen:

- public NamingResult get(int key)
- public NamingResult get(Long key)
- public NamingResult get(String key)

Mit dem "key"-Parameter haben Sie die Möglichkeit, nur jene Parameter abzurufen, deren Name mit der jeweiligen Schlüsselzeichenfolge beginnt.

Über die Klasse "NamingResult" wird die Methode *get(String parameter)* bereitgestellt, über die der Wert des Parameters zurückgegeben werden kann.

"NamingService" - Beispiel

Beispielsweise wird im folgenden Codefragment der Wert des Nachrichten-URL-Parameters für einen Distributor wiederhergestellt, auf den über den entsprechenden Partnerschlüssel verwiesen wird.

```
NamingService namingService = NamingManager.getInstance();
NamingResult namingResult = namingService.get(partnerKey);
String url = namingResult.get(NamingResult.MESSAGE_URL);
```

Anmerkung: Standardmäßig wird bei "NamingService" ein Objekt des Typs "NamingServiceDatabase" erstellt, da in der Datei **Comergent.xml** für das Element "defaultType" bei "NamingManager" der Wert "database" definiert ist.

Kapitel 14. Software-Development-Kit für Visual Modeler

Software-Development-Kit zum Anpassen der Visual Modeler-Implementierung verwenden

Über das Software-Development-Kit (SDK) von Visual Modeler können Sie Ihre Implementierung von Visual Modeler installieren und anpassen. In der mit jeder Version des SDK mitgelieferten HTML-Dokumentation finden Sie eine Übersicht dazu, wie das SDK funktioniert und wie es beim Verwalten von Projekten einzusetzen ist. Im vorliegenden Abschnitt wird die Basisstruktur eines Anpassungsprojekts beschrieben. Befolgen Sie diese Anweisungen zum Organisieren Ihres Projekts, damit dieses den Anpassungsrichtlinien entspricht.

Projektorganisation

Jedes mithilfe des Software-Development-Kit (SDK) aufgebaute Projekt wird auf der Basis eines Releases von Visual Modeler erstellt. Wenn Sie das Projekt mithilfe des Zielelements "newproject" erstellen, wird vom SDK eine Gruppe von Projektdateien passend zum jeweiligen Release erstellt. Alle im Projekt vorgenommen Änderungen erfolgen durch Hinzufügen von Dateien zum Projekt. Sie können Dateien wie folgt zum Projekt hinzufügen:

- Verwenden Sie das Zielelement "customize", um eine Datei aus dem Release in das Projekt zu kopieren. Wenn Sie das Zielelement "customize" verwenden, wird die Datei automatisch in das entsprechende Unterverzeichnis des Projekt kopiert.
- Erstellen Sie die Datei manuell im entsprechenden Unterverzeichnis des Projekts.

Informationen darüber, wo die Dateien positioniert sein müssen, finden Sie unter „Projektdatei- und Projektverzeichnispositionen”.

Projektdatei- und Projektverzeichnispositionen

Diesem Abschnitt liegt die Annahme zugrunde, dass Sie ein Projekt mit Namen *projekt* erstellt haben und dass Sie über ein Projektverzeichnis mit Namen ***sdk_ausgangsverzeichnis*/projects/***projekt*/ verfügen. Stellen Sie sicher, dass sich alle Projektdateien wie folgt an der entsprechenden Position unter dem Projektverzeichnis befinden:

- Java-Quellendateien: Müssen sich im Verzeichnis ***projekt*/src/** befinden und der Paketorganisation für Visual Modeler entsprechen.
- JSP-Seiten: Sind nach Modul und Ländereinstellung im Verzeichnis ***projekt*/WEB-INF/web/** organisiert.
- Schemadateien: Dazu gehören die Datenobjektdateien und die unterstützenden Datenservicesdateien. Alle Ihre Anpassungen sollten im Verzeichnis ***projekt*/WEB-INF/schema/custom/** geführt werden. Stellen Sie sicher, dass für das Element "schemaRepositoryExtn" der Wert “WEB-INF/schema/custom” definiert wird.

Java-Quellendateien

Halten Sie sich im Verzeichnis ***projekt*/src/** beim Organisieren der Anpassungen von Visual Modeler an die folgenden Richtlinien:

- Verwenden Sie die "com/comergent/api/"-Pakete, um Ihre Erweiterungen zur Visual Modeler-API hinzuzufügen. Gewöhnlich werden Sie neue Klassen

erstellen, durch die die vorhandene API erweitert wird. Überschreiben Sie dabei nicht die Release-API, da das Einfluss auf die Durchführung von Upgrades haben kann.

- Verwenden Sie die "com/comergent/apps/"- und "com/comergent/appservices/ "-Pakete, um Implementierungsklassen hinzuzufügen. Dabei kann es sich um komplett neue Klassen oder um neue Klassen handeln, durch die bereits vorhandene Implementierungsklassen erweitert werden.
- Verwenden Sie die "com/comergent/reference/"-Pakete für Controllerklassen. Sie können bereits vorhandene Controllerklassen anpassen oder neue Controllerklassen erstellen.

JSP-Seiten

Halten Sie sich im Verzeichnis ***projekt*/WEB-INF/web/** beim Organisieren der Anpassungen von Visual Modeler an die folgenden Richtlinien:

- Verwenden Sie zum Hinzufügen neuer JSP-Seiten oder zum Anpassen vorhandener JSP-Seiten (wenn möglich) die bereits vorhandene Organisation der JSP-Seiten.
- Wenn Sie ein neues Funktionsmodul hinzufügen, erstellen Sie für das Modul unter der/den entsprechenden Ländereinstellung(en) ein neues Verzeichnis, wobei Sie dieselben Namenskonventionen anwenden, die Sie bereits bei den für das Modul erstellten Java-Klassen angewendet haben.

Schemadateien

Halten Sie sich im Verzeichnis ***projekt*/WEB-INF/schema/custom/** beim Organisieren der Anpassungen von Visual Modeler an die folgenden Richtlinien:

- So fügen Sie neue Datenobjekte hinzu:
  - Platzieren Sie die XML-Definition des Datenobjekts im Verzeichnis ***projekt*/WEB-INF/schema/custom/**. Erstellen Sie z. B. die Datei ***projekt*/WEB-INF/schema/custom/CustComment.xml**.
  - Ändern Sie die Datei ***projekt*/WEB-INF/schema/custom/ DsBusinessObjects.xml**, indem Sie das neue Geschäftsobjekt hinzufügen. Beispiel:

    ```
    <?xml version="1.0"?>
    <Schema Name="project" Description="project Custom Schema"
    Version="6.0">
    <BusinessObject Name="CustComment" Version="6.0"
    Description="CustComment BusinessObject"/>
    </Schema>
    ```

  - Ändern Sie die Datei ***projekt*/WEB-INF/schema/custom/DsDataElements.xml**, indem Sie (zusammen mit beliebigen, durch das Datenobjekt deklarierten Feldern) die neuen Datenelemente für die Header- und Listendatenobjekte hinzufügen. Beispiel:

    ```
    <?xml version="1.0"?>
    <Schema Name="project" Description="project Custom Schema"
    Version="6.0">
    <DataElement Name="CustComment" Description="Customer Comment data
    object"
    DataType="HEADER"/>
    <DataElement Name="CustCommentList" Description="Customer Comment list
    data
    object" DataType="HEADER"/>
    ```

```
<DataElement Name="CustCommentKey" Description="Customer Comment Key"
DataType="LONG" MaxLength="20"/>
</Schema>
```

– Ändern Sie die Datei *projekt*/**WEB-INF/schema/custom/DsRecipes.xml**, indem Sie ein Anleitungselement hinzufügen. Beispiel:

```
<Schema Name="project" Description="project Custom Schema"
Version="6.0">
<Recipe Name="CustComment" BusinessObject="CustComment"
Description="Default Approvals List Recipe" Version="6.0">
        <DataObjectList>
<DataObject Name="CustComment" Access="RWID"
DataSourceName="ENTERPRISE" Ordinality="n"
Version="6.0"/>
</DataObjectList>
</Recipe>
</Schema>
```

– Ändern Sie die entsprechende Schlüsselgeneratordatei (z. B. projekt/WEB-INF/schema/custom/OracleKeyGenerators.xml), indem Sie die erforderlichen neuen Schlüssel hinzufügen:

```
<?xml version="1.0"?>
<Schema Description="project Custom Schema" Name="project"
Version="6.0">
<KeyGenerator Name="CustCommentKey" KeyProcedureName="CUSTCOMMENTKEY"
GeneratorType="PROCEDURE" />
</Schema>
```

Kapitel 15. Visual Modeler-Lokalisierung

Visual Modeler-Lokalisierung - Übersicht

Visual Modeler bietet integrale Unterstützung für:

- Mehrere Währungen
- Mehrere Sprachen
- Zahlen- und Datumsformate
- Zeichensätze

Sie können aber auch noch andere Aspekte der Lokalisierung für spezielle Märkte steuern, z. B. in Bezug auf:

- Nationale Gesetze und Regelungen
- Die Verarbeitung der Währung
- Informationen zu Versand und Export
- Steuern

Die Unterstützung für die Internationalisierung erfolgt über Ländereinstellungen. Jede Ländereinstellung entspricht einer Sprache und einem Land oder einer Region. Durch Angabe der für die Anzeige von Informationen für die Benutzer zu verwendenden Ländereinstellungen stellen Sie sicher, dass einem Benutzer die für sein Land spezifischen Angaben in der von ihm erwarteten Form dargeboten werden.

Wenn sich Benutzer bei Visual Modeler anmelden, wird dieser Sitzung eine Ländereinstellung zugeordnet. Dabei handelt es sich um die bevorzugte Ländereinstellung entsprechend der Festlegung im Benutzerprofil. Die Benutzer haben die Möglichkeit, die von ihnen bevorzugte Ländereinstellung in ihrem Benutzerprofil zu ändern, wobei diese Änderungen erst nach der nächsten Anmeldung bei Visual Modeler wirksam werden. Benutzeradministratoren können sowohl die bevorzugte Ländereinstellung eines Benutzers als auch andere Aspekte des Benutzerprofils ändern.

Die standardmäßige Systemländereinstellung wird in der Konfigurationsdatei **Internationalization.xml** im Element "defaultSystemLocale" festgelegt. Sie können für jede Sprache eine standardmäßige Ländereinstellung festlegen. Weitere Informationen hierzu finden Sie unter „Verhalten im Ausweichbetrieb” auf Seite 93.

Visual Modeler bietet vollständige Unicode-Unterstützung für die Dateneingabe und -anzeige.

Ein beträchtlicher Umfang an Lokalisierung kann mithilfe der Java-Ressourcenpakete ("ResourceBundles") erfolgen. Detaillierte Informationen hierzu finden Sie unter „Ressourcenpakete und Formate” auf Seite 100.

Ländereinstellungen unterstützen

Wenn Sie beabsichtigen, Visual Modeler so zu implementieren, dass neben der Ländereinstellung "en_US" weitere Ländereinstellungen unterstützt werden können, müssen Sie entsprechende Seiten mit den jeweiligen Landessprachen und

anderen länderspezifischen Informationen (z. B. den Bürostandorten) erstellen.

Darstellungs- und Sitzungsländereinstellungen

Wenn sich ein Benutzer bei Visual Modeler anmeldet, wird im Rahmen des Authentifizierungsprozesses die jeweils bevorzugte Ländereinstellung abgerufen. Diese ist in den entsprechenden Benutzerprofilen definiert. Das System bedient sich zweier logisch unterschiedlicher Ländereinstellungen:

- Sitzungsländereinstellung: In dieser Einstellung wird festgelegt, welche Daten für Datenobjekte aus der Wissensdatenbank abgerufen werden sollen.
- Darstellungsländereinstellung: In dieser Einstellung wird festgelegt, welche JSP-Seiten und Ressourcenpakete für die Ausgabe von HTML-Seiten an die Benutzer verwendet werden sollen.

In der Regel muss die Gruppe der von Ihnen als Darstellungsländereinstellungen unterstützten Ländereinstellungen eine Untergruppe der möglichen Sitzungsländereinstellungen bilden. Beispielsweise könnten Sie "fr_CA", "fr_CH" und "fr_FR" als Sitzungsländereinstellungen führen, aber lediglich "fr_FR" und "fr_CA" als Darstellungsländereinstellungen unterstützen.

Wenn sich ein Benutzer zum ersten Mal anmeldet, wird vom System wie folgt für diese Benutzersitzung eine Darstellungsländereinstellung ermittelt:

1. Wenn die bevorzugte Ländereinstellung des Benutzers in der Visual Modeler-Datei **web.xml** deklariert ist, übernehmen Sie diese Einstellung als Darstellungsländereinstellung.
2. Ist das nicht der Fall, informieren Sie sich in der Datei **Internationalization.xml**: Wenn hier der Wert für das Element "useCountryDefaulting" auf "true" lautet, geben Sie für die Sprache der bevorzugten Ländereinstellung des Benutzers die standardmäßige Ländereinstellung des betreffenden Landes bzw. der betreffenden Region an. Überprüfen Sie, ob die standardmäßige Ländereinstellung des Landes bzw. der Region in der Datei **web.xml** deklariert wird. Ist das der Fall, definieren Sie die Darstellungsländereinstellung entsprechend.
3. Wenn entweder der Wert für das Element "useCountryDefaulting" auf "false" lautet oder die standardmäßige Ländereinstellung des Landes bzw. der Region nicht in der Datei **web.xml** enthalten ist und wenn der Wert für das Element "useGeneralDefaulting" auf "true" lautet, dann definieren Sie für die Darstellungsländereinstellung des Benutzers die im Element "defaultSystemLocale" festgelegte standardmäßige Systemländereinstellung.
4. Wenn für die mit "Defaulting" gekennzeichneten Elemente der Wert "false" definiert ist oder keine in der Datei **web.xml** deklarierte Ländereinstellung gefunden werden kann, wird als Darstellungsländereinstellung die Sitzungsländereinstellung definiert.

Diese Darstellungsländereinstellung dient dem Umgang mit den Erfahrungen der Benutzer beim Navigieren durch Visual Modeler. Dabei wird gesteuert, welche JSP-Seiten und Eigenschaftendateien für die Ausgabe der den Benutzern anzuzeigenden Webseiten verwendet werden sollen. Gleichzeitig wird die bevorzugte Ländereinstellung als *Sitzungsländereinstellung* definiert: Diese Sitzungsländereinstellung wird dazu verwendet, festzulegen, welche Daten aus der Datenbank abgerufen werden sollen, wenn den Benutzern lokalisierte Datenobjekte angezeigt werden.

Anmerkung: Sie müssen grundsätzlich dafür sorgen, dass jede von Ihnen in der Datenbank definierte Ländereinstellung entweder über eine entsprechende Gruppe von Einträgen in der Datei **web.xml** verfügt oder dass die standardmäßige Ländereinstellung des Landes bzw. der Region über entsprechende Einträge in der Datei **web.xml** verfügt und Sie die Verwendung der für das Land bzw. die Region gültigen Standardwerte aktivieren. Wenn Sie dem nicht nachkommen, werden einige Benutzer möglicherweise nicht in der Lage sein, auf das System zuzugreifen.

JSP-Seiten und Eigenschaftendateien

1. Für jede JSP-Seite muss im entsprechenden Modulunterverzeichnis unter dem Systemverzeichnis mit der standardmäßigen Ländereinstellung mindestens eine JSP-Seite vorhanden sein. Wenn Sie Visual Modeler zum ersten Mal installieren, wird als standardmäßige Systemländereinstellung "en_US" definiert. Folglich wird unter ***debs_ausgangsverzeichnis*/SterlingWEB-INF/web/en/US/** eine umfangreiche Gruppe von JSP-Seiten bereitgestellt. Wenn Sie die standardmäßige Systemländereinstellung wechseln, müssen Sie darauf achten, dass die entsprechenden Verzeichnisse für die neue Ländereinstellung vollständig aufgefüllt werden.
2. Der auf den einzelnen Seiten sichtbare Text wird über den "text"-Tag der Comergent-Tagbibliothek oder die entsprechende *cmgtText()*-Methode deklariert. Beispiel:

   ```
   <cmgt:text
    id='cmgt_channelMgmt/channelCartDisplay/ChannelCartDisplayData_7'
   bundle='channelMgmt.channelCartDisplay.ChannelCartDisplayDataResources'
   >Build Product List </cmgt:text>
   ```

   oder

   ```
   String title = cmgtText("cmgt_commerce/search/AdvancedSearchBody_2",
   "Inquiry Lists Search");
   ```

   Das Attribut "bundle" muss einer Datei in dem Paket "com.comergent.reference.jsp" in der Klassenbaumstruktur entsprechen. Für das oben gezeigte Beispiel muss es im Verzeichnis ***debs_ausgangsverzeichnis*/Sterling/WEB-INF/classes/com/comergent/reference/jsp/channelMgmt/channelCartDisplay/** eine Datei mit Namen **ChannelCartDisplayDataResource.properties** geben. Das Attribut "id" muss innerhalb der Eigenschaftendatei eindeutig sein. In Bezug auf das oben gezeigte Beispiel muss eine Zeile wie die folgende existieren:

   ```
   cmgt_channelMgmt/channelCartDisplay/ChannelCartDisplayData_7=Build Product List
   ```

3. Für jede zusätzlich unterstützte Ländereinstellung (z. B. *la_CO*) müssen Sie die folgenden Verzeichnisse aus **debs_ausgangsverzeichnis/Sterling/WEB⌂INF/web/en/US/** nach ***debs_ausgangsverzeichnis*/Sterling/WEB⌂INF/web/*la*/*CO*/** kopieren:
   - **cic/**
   - **common/**
   - **home/**
4. Für jede zusätzlich unterstützte Ländereinstellung (z. B. *la_CO*) und für jede JSP-Seite müssen Sie:
   a. Entweder eine neue JSP-Seite für die Ländereinstellung erstellen und diese an der entsprechenden Verzeichnisposition in der Webanwendung (nämlich einem Verzeichnis unter ***debs_ausgangsverzeichnis*/Sterling/WEB⌂INF/web/*la*/*CO*/**) platzieren. Wenn ein und dieselbe Seite in einer Sprache für mehrere Ländereinstellungen (z. B für "fr_FR" und "fr_CA") verwendet werden kann, müssen Sie sicherstellen, dass sich diese in der standardmäßigen Ländereinstellung für die betreffende Sprache befindet.

Weitere Informationen zu standardmäßigen Ländereinstellungen für Sprachen finden Sie unter „Verhalten im Ausweichbetrieb” auf Seite 93.

b. Oder eine Eigenschaftendatei mit entsprechendem Text für jede ID vorbereiten. Diese Eigenschaftendateien werden so organisiert, dass pro JSP-Seite und JSP-Fragment eine Datei zur Verfügung steht.

   HTML- und Javascript-Zeichen wie "<", ">", " ' " etc. dürfen in den Eigenschaftswerten nicht verwendet werden. Diese Zeichen müssen über die entsprechenden HTML- oder Javascript-Mechanismen für Escapezeichen ersetzt werden. Beispiel: Verwenden Sie in HTML "<" für "<" und in Javascript " \' " für " ' ".

   Die Eigenschaftendateien müssen dem von den Ressourcenpaketen verwendeten Java-Standard für Eigenschaftendateien entsprechen. Insbesondere muss diese Namenskonvention befolgt werden: ***<name_der_jsp-seite>*Resources_*la_CO*.properties**. Bei den Dateien muss es sich um Textdateien handeln, in denen jede Zeile folgendes Format aufweisen muss:

   ```
   cmgt_module/package/JSPname_n=Display text for this locale
   ```

   Beispiel:

   ```
   cmgt_channelMgmt/channelCartDisplay/
   ChannelCartDisplayData_7=Build Product List
   ```

Alle Eigenschaftendateien befinden sich im Verzeichnis ***debs_ausgangsverzeichnis*/Sterling/WEB-INF/classes/com/comergent/reference/jsp/** und sind innerhalb dieses Verzeichnisses nach Modulen organisiert. Diese Struktur entspricht der Anordnung der Modul-JSP-Seiten innerhalb eines Moduls. Beachten Sie, dass Sie bei Änderung der Verzeichnisposition für diese Ressourcenpakete auch den "text"-Tag entsprechend anpassen müssen, damit die Ressourcenpakete anschließend von ihrer neuen Position abgerufen werden können.

Wenn Sie Text zu einer JSP-Seite hinzufügen, denken Sie daran, die entsprechenden JSP-Seiten oder Eigenschaftendateien der Ländereinstellung zu aktualisieren (und zwar entweder durch ergänzenden Text für eine bereits vorhandene Tag-ID oder durch Hinzufügen einer neuen ID).

Beachten Sie Folgendes:

- Die Länge des Texts nach seiner Umsetzung kann signifikant abweichen. Dies kann Einfluss auf das Layout einer Webseite haben.
- Dropdown-Listen und Javascript-Funktionen können Text enthalten, der nach seiner Umsetzung Einfluss auf die Logik von Visual Modeler haben kann. Weitere Informationen hierzu finden Sie unter „Javascript” auf Seite 97 und „JSP-Seiten” auf Seite 86.
- Lokale Regelungen können Einfluss auf die Anzeige bestimmter Informationen haben (Beispiel: die Anzeige von Preisen in Euro und einer Landeswährung).
- Seien Sie besonders vorsichtig, wenn sich der logische Ablauf von Seiten ändern muss, um so bestimmten lokalen Verfahren (z. B. in Bezug auf die Darstellung von Exportpapieren oder steuerlichen Informationen) zu entsprechen.

Mithilfe des Elements "debugJSPResouceBundle" in der Konfigurationsdatei **Internationalization.xml** können Sie fehlende Zeichenfolgen ermitteln. Geben Sie hierzu für dieses Element den Wert "true" an. Wenn dann in dem referenzierten Ressourcenpaket eine Zeichenfolge fehlt, wird auf der Browserseite eine entsprechende Fehlernachricht ausgegeben. In Ihren Produktionssystemen sollten Sie hierzu den Wert "false" angeben.

Verhalten im Ausweichbetrieb

Im vorliegenden Abschnitt wird beschrieben, was geschieht, wenn für die aktuelle Darstellungsländereinstellung eines Benutzers bestimmte Ressourcen (JSP-Seiten oder Eigenschaften) nicht definiert sind. Beachten Sie, dass sich die Verhaltensweisen im Ausweichbetrieb bei JSP-Seiten und Ressourcenpaketen leicht voneinander unterscheiden:

- Bei JSP-Seiten kann der Ausweichbetrieb ausgehend von einer bestimmten Ländereinstellung über die standardmäßige Angabe von Land oder Region für die Spracheinstellung hin zur standardmäßigen Systemländereinstellung erfolgen. Beispiel: Von "fr_CA" über "fr_FR" nach "en_US".
- Bei Ressourcenpaketen erfolgt der Ausweichbetrieb entsprechend der folgenden Java-Spezifikation: Von ***_fr_CA.properties** über ***_fr.properties** nach ***.properties**.

Zum Steuern des Verhaltens im Ausweichbetrieb für JSP-Seiten werden die folgenden beiden Eigenschaften in der Konfigurationsdatei **Internationalization.xml** verwendet:

- useCountryDefaulting: Wenn hier der Wert "true" angegeben ist und für die Darstellungsländereinstellung keine Ressource verfügbar ist, wird standardmäßig das Land bzw. die Region verwendet, das bzw. die in dem entsprechenden Sprachelement angegeben ist.
- useGeneralDefaulting: Wenn hier der Wert "true" angegeben ist und für die Darstellungsländereinstellung keine Ressource verfügbar ist, wird standardmäßig die Systemländereinstellung verwendet.

Verhalten im Ausweichbetrieb: Ressourcenpakete

Sie müssen nicht alle Textzeichenfolgen entsprechend jeder Ländereinstellung übersetzen lassen. Wenn eine bestimmte Textzeichenfolge für eine bestimmte ID in der Eigenschaftendatei für das betreffende Ressourcenpaket nicht vorhanden ist, wird der standardmäßige Java-Ausweichbetrieb ausgeführt. Wenn z. B. in der Datei **ChannelCartDisplayDataResource_fr_CA.properties** nicht die Zeichenfolge "cmgt_channelMgmt/channelCartDisplay/ChannelCartDisplayData_7" definiert ist, wird (falls vorhanden) die Datei **ChannelCartDisplayDataResource_fr.properties** zu Rate gezogen. Wenn diese Datei nicht existiert oder keinen Eintrag für diese ID enthält, wird in der Datei **ChannelCartDisplayDataResource.properties** nachgeschaut.

Verhalten im Ausweichbetrieb: JSP-Seiten

Nicht alle JSP-Seiten müssen auch für alle unterstützten Ländereinstellungen verfügbar sein. Beispielsweise ist es vorstellbar, dass Sie für fast alle Benutzer Seiten mit der Einstellung "en_US" verwenden, während eine kleine Anzahl von Seiten mit der Einstellung "en_CA" nur für bestimmte Benutzer angezeigt werden. Im vorliegenden Abschnitt wird beschrieben, was bei der Verarbeitung eines Nachrichtentyps geschieht:

Die Anforderung wird an die im Element "JSPMapping" des Nachrichtentyps in der entsprechenden Datei **MessageTypes.xml** angegebene JSP-Seite weitergeleitet.

1. Wenn die JSP-Seite für die aktuelle Ländereinstellung verfügbar ist, wird diese Seite zum Generieren der Webseite verwendet.
2. Wenn die JSP-Seite für die aktuelle Ländereinstellung nicht verfügbar ist, wird im Rahmen des Mechanismus bei Ausweichbetrieb für die Sprache der aktuellen Ländereinstellung die standardmäßige Ländereinstellung angegeben.

Dies erfolgt durch Deklarieren des Elements "defaultCountry" für die Sprache in der Konfigurationsdatei **Internationalization.xml**.

3. Wenn eine JSP-Seite in der Ländereinstellung mit dem Sprachenstandard existiert, wird diese Seite zum Generieren der Webseite verwendet. So wird beispielsweise über das folgende Element in der Datei **Internationalization.xml** angegeben, dass es sich bei der Angabe "US" um das standardmäßige Land bzw. die standardmäßige Region für die "en"-Spracheinstellungen handelt und dass, falls für die "en_CA"-Ländereinstellung keine JSP-Seite zur Verfügung steht, die entsprechende JSP-Seite mit der Kennzeichnung "en_US" verwendet wird.
4. <en visible="false">
   <defaultCountry ...>US</defaultCountry>
   </en>
5. Wenn für das standardmäßige Land bzw. die standardmäßige Region keine JSP-Seite existiert, wird im Rahmen des Mechanismus bei Ausweichbetrieb die standardmäßige Systemländereinstellung angegeben. Diese wird als Wert des Elements "defaultSystemLocale" in der Datei **Internationalization.xml** deklariert. Wenn eine JSP-Seite in der standardmäßigen Systemländereinstellung existiert, wird diese Seite zum Generieren der Webseite verwendet.
6. Wenn schließlich keine JSP-Seite in der standardmäßigen Systemländereinstellung existiert, kommt es zu einer Ausnahmebedingung und es wird eine entsprechende Fehlerseite angezeigt.

Methoden zum Abrufen von Ländereinstellungen

In der Regel sollten Sie in der Lage sein, Gebrauch von der in Visual Modeler integrierten Unterstützung zum Anzeigen der korrekten Inhalte entsprechend den Ländereinstellungen der Benutzer zu machen. Wenn Sie manuell auf Ländereinstellungen zugreifen müssen, können Sie dazu die Klasse "ComergentI18N" verwenden. Über diese Klasse werden die folgenden Methoden bereitgestellt:

- *getDefaultLocale()*: Damit wird die standardmäßige Systemländereinstellung ausgegeben.
- *getComergentLocale(boolean b)*: Wenn für "b" der Wert "true" angegeben wird, wird die Darstellungsländereinstellung des Benutzers ausgegeben. Ansonsten wird die Sitzungsländereinstellung des Benutzers ausgegeben.
- *findPresentationLocale(Locale sessionLocale)*: Wird dazu verwendet, zu ermitteln, welche Darstellungsländereinstellung für eine bestimmte Sitzungsländereinstellung verwendet werden soll.

Eigenschaftendateien in Code verwenden

Sie können auch in Ihrem Java-Code Gebrauch von Eigenschaftendateien machen. Wenn Sie z. B. die ländereinstellungsspezifische Zeichenfolge ("String") abrufen möchten, die der in der Datei **com.comergent.reference.jsp.AdvisorBodyResources.properties** definierten Zeichenfolge "keyString" entspricht, gehen Sie wie folgt vor:

```
String temp_NamedPopertiesFile =
"com.comergent.reference.jsp.AdvisorBodyResources.properties";
ResourceBundle temp_ResourceBundle =
com.comergent.dcm.util.ComergentI18N.-
getBundle(temp_NamedPopertiesFile);
```

```
String temp_LocalisedString =
temp_ResourceBundle.getString("keyString");
```

Dabei wird die aktuelle (in der Benutzersitzung gespeicherte) Ländereinstellung des Benutzers verwendet. Möchten Sie die Verwendung einer anderen Ländereinstellung erzwingen, gehen Sie wie folgt vor:

```
Locale specific_Locale = new Locale("fr", "CA");
String temp_NamedPopertiesFile =
"com.comergent.reference.jsp.AdvisorBodyResources.properties";
ResourceBundle temp_ResourceBundle =
com.comergent.dcm.util.ComergentI18N.-
getBundle(temp_NamedPopertiesFile, specific_Locale);
String temp_LocalisedString =
temp_ResourceBundle.getString("keyString");
```

Daten für Internationalisierung

Wenn Sie davon ausgehen, dass durch Unternehmensbenutzer und Endbenutzer Daten mit Mehrfachbytezeichen eingegeben werden, müssen Sie sich Gedanken über die Länge der Datenfelder und der entsprechenden Datenbanktabellenspalten machen. Wie die Erfahrungen zeigen, können Daten, die mit Mehrfachbytezeichen in Visual Modeler eingegeben werden, in der Datenbank bis zu dreimal so lang wie die für die Ländereinstellung "en_US" verwendeten Zeichenfolgen ausfallen. Folglich sollten Sie sich die Länge der Felder, für die Sie die Eingabe von Daten mit Mehrfachbytezeichen erwarten, nochmals genau anschauen (insbesondere die Namens- und Beschreibungsfelder).

Wenn Sie die Länge von Feldern ändern möchten, müssen Sie daran denken, dass Sie diese Änderungen sowohl in der Konfigurationsdatei **DsDataElements.xml** als auch in dem SQL-Script vornehmen müssen, das zum Generieren des Schemas in der Wissensdatenbank verwendet wird.

Beispiel

Führen Sie die folgenden Schritte aus, um das Feld "Description" des Datenobjekts "Product" ausreichend lang für die Aufnahme von Mehrfachbytezeichen zu gestalten:

1. Geben Sie das für Produktbeschreibungen vorgesehene Datenfeld an. Da es sich bei dem Datenobjekt "Product" um ein lokalisierbares Datenobjekt (Localized=“y”) handelt, ist dies das Feld "Description" im Datenobjekt "ProductLocale". Die entsprechende Datenbanktabelle und -spalte lautet "CMGT_PRODUCT_LOCALE.DESCRIPTION".

   ```
   <DataField Name="Description" ExternalFieldName="DESCRIPTION"
   Mandatory="n" Writable="y"/>
   ```

2. Nehmen Sie an, dass Sie Beschreibungen mit maximal 240 Zeichen Länge zulassen möchten:

   <DataElement Name="Description" DataType="STRING" Description="Description" MaxLength="240" />

3. Ändern Sie die entsprechende SQL-Anweisung, durch die die Tabelle "CMGT_PRODUCT_LOCALE" erstellt wird, dahingehend, dass für die Spalte "DESCRIPTION" der Wert "VARCHAR2(720)" definiert wird:

   DESCRIPTION VARCHAR2(720) DEFAULT 'Not available',

4. Führen Sie die entsprechenden SDK-Zielelemente ("merge" und "createDB") aus, um die Änderungen an Ihrer Implementierung von Visual Modeler vorzunehmen.

Beachten Sie, dass das Datenfeld "Description" in diesem Beispiel häufig durch zahlreiche verschiedene Datenobjekte verwendet wird und dass es daher bei Änderung seiner Definition in der Konfiguratonsdatei **DsDataElements.xml** zu unerwarteten Nebeneffekten kommen kann. Alternativ zur genannten Vorgehensweise könnten Sie daher ein neues Datenfeld mit Namen "ProductDescription" erstellen und dieses im Datenobjekt "ProductLocale" verwenden. Dazu müssten Sie in der Datei **ProductLocale.xml** die folgende Eingabe vornehmen:

```
<DataField Name="ProductDescription"
ExternalFieldName="DESCRIPTION" Mandatory="n" Writable="y"/>
```

Anschließend müssten Sie in der Konfigurationsdatei **DsDataElements.xml** folgende Eingabe ausführen:

```
<DataElement Name="ProductDescription" DataType="STRING"
Description="This is the product description field"
MaxLength="240" />
```

Anmerkung: Wenn Sie eine Javascript-Methode zur Verfügung stellen, mit der geprüft werden kann, ob von den Benutzern gültige Daten in die Felder eingegeben werden, müssen Sie beim Überprüfen der Feldlänge die Werte in der entsprechenden Komponente "DataElement" überprüfen.

E-Mail-Schablonen

Wenn in Ihrem System neben Englisch auch noch andere Sprachen unterstützt und in Ihrer Installation von Visual Modeler E-Mail-Schablonen zum Generieren von Nachrichten für Benutzer verwendet werden, müssen Sie berücksichtigen, dass diese übersetzt werden müssen.

In Release 6.4 wurde die Funktionalität zum Verwenden von JSP-Seiten zum Generieren von E-Mail-Nachrichten eingeführt. Diese bietet Unterstützung für die Internationalisierung von E-Mail-Nachrichten, wobei ein vorhandenes Framework zur Internationalisierung von JSP-Seiten verwendet wird.

In traditionellen Anwendungen können Sie die in Visual Modeler bereitgestellten standardmäßigen Schablonen verwenden. Sie finden diese Schablonen im Verzeichnis ***debs_ausgangsverzeichnis*/Sterling/WEB-INF/templates/**.

HTML-Seiten

Statische HTML-Seiten müssen gegebenenfalls übersetzt werden. Wenn Sie gleichzeitig mehrere Sprachen unterstützen möchten, müssen Sie dafür sorgen, dass für jede Sprache separate Seiten erstellt werden. Vorausgesetzt, Sie sorgen für eine konsistente Verwaltung der Positionen dieser Seiten innerhalb Ihrer lokalen Verzeichnisstruktur, dann können die relativen Bezüge zu diesen Seiten immer korrekt entsprechend der jeweiligen HTML-Seite aufgelöst werden.

Beispielsweise werden im folgenden JSP-Fragment dynamisch URLs generiert, über die auf eine ländereinstellungsspezifische Seite **Example.html** verwiesen wird:

```
<A HREF="<cmgt:link app="catalog">
/static/Example.html
</cmgt:link>">
resourceBundle.getString("ExamplePage")
</A>
```

In diesem Beispiel wird ein Ressourcenpaket dazu verwendet, den für den Link anzuzeigenden Text zu bestimmen.

Bilder

Gewöhnlich verwenden Sie Bilder ohne integrierten Text. Dadurch ist sichergestellt, dass Sie dieselben Bilder im Rahmen mehrere unterschiedlicher Ländereinstellungen verwenden können, wodurch Kosten für die Lokalisierung und Wartung eingespart werden können.

Wenn erforderlich, sollten Sie aber auch für lokalisierte Versionen der Bilder sorgen. Wie schon bei den statischen HTML-Seiten können Sie auch hier über relative URLs sicherstellen, dass ländereinstellungsspezifische Bilder entsprechend der jeweiligen JSP-Seite aus den korrekten Positionen abgerufen werden können.

In erster Linie müssen Sie daran denken, dass alle Schaltflächen auf extern sichtbaren Seiten Bildschaltflächen mit Text sind. Daher müssen Sie (falls erforderlich) lokalisierte Versionen der verschiedenen Schaltflächen erstellen. Die URLs der Bilderquellen können dann wie folgt generiert werden:

```
<IMG ALT="Locale-specific alternate text goes here"
SRC="../images/button.gif"></A>
```

Javascript

Berücksichtigen Sie, dass in Ihrem JavaScript verwendeter anzuzeigender Text lokalisiert werden muss. So sollte z. B. Text in Alert-Dialogfenstern entsprechend der Ländereinstellung des betreffenden Benutzers angezeigt werden.

- Einige Javascript-Dateien werden über die folgenden Zeilen in die Webseiten integriert:

  ```
  <script language='JavaScript' src='../js/genericUtil.js'>
  </script>
  ```

  Sie müssen diese JavaScript-Dateien für alle Ländereinstellungen verfügbar machen, sodass der Browser in der Lage ist, diese Dateien korrekt in die generierten Webseiten zu integrieren.
- Wenn Javascript innerhalb einer JSP-Seite oder eines integrierten JSP-Fragments definiert wird, muss der jeweilige Anzeigetext über das Tag "text" eingeschlossen werden. Beispiel:

  ```
  alert("<cmgt:text id="*">Product ID is missing.</cmgt:text>");
  ```

  Wenn diese Tags als Teil des SDK-Tools verarbeitet werden, wird über das Attribut "id" eine eindeutige ID generiert und ID und Text des Tags werden zum Ressourcenpaket für die JSP-Seite oder das JSP-Fragment hinzugefügt.

Visual Modeler-Lokalisierung: JSP-Seiten

Generell müssen alle Lokalisierungen für Bezeichnungen, erklärende Texte, ausgefüllte Listen und ländereinstellungsspezifische Formatierungen von Datumsangaben und Währungen in den für eine Ländereinstellung erstellten JSP-Seiten wiederzufinden sein.

Eine sinnvolle Organisation erreichen Sie durch Definieren einer Hashzuordnung aller lokalisierten Zeichenfolgen auf einer Seite, wobei von allen Teilen der Seite darauf Bezug genommen werden kann. Beispiel:

```
HashMap localized = new HashMap();
localized.put("TaskListHeader",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_3","Task List:"));
localized.put("QuickSearchTitle",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_4","Search for Tasks"));
localized.put("TaskID",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_5","ID"));
localized.put("TaskName",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_6","Name"));
localized.put("Status",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_7","Status"));
localized.put("Priority",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_8","Priority"));
localized.put("CreateDate",
cmgtText("cmgt_taskMgr/TaskWorkspaceData_9","Create Date"));
request.setAttribute("localized", localized);
```

Sie können in den folgenden Zeilen mithilfe der Scripting-Funktionalität auf diese Zeichenfolgen Bezug nehmen:

```
<cic:span css="banner" value="${localized['TaskListHeader']}"/>
```

Dieses Verfahren bietet den Vorteil, dass JSP-Seiten leichter zu lesen sind, dass Sie lokalisierte Zeichenfolgen ohne großen Aufwand wiederverwenden können und dass das Verfahren eher dem JSF-Modell entspricht.

Informationen darüber, wie diese UI-Komponente zu lokalisieren ist, finden Sie unter „Kalenderwidget” auf Seite 99. Füllen Sie z. B. wie folgt eine Dropdown-Liste mit den Wochentagen für eine französische Ländereinstellung aus:

```
<SELECT Name="DayOfWeek">
<OPTION VALUE=0>dimanche</OPTION>
<OPTION VALUE=1>lundi</OPTION>
<OPTION VALUE=2>mardi</OPTION>
<OPTION VALUE=3>mercredi</OPTION>
<OPTION VALUE=4>jeudi</OPTION>
<OPTION VALUE=5>juin</OPTION>
<OPTION VALUE=6>vendredi</OPTION>
<OPTION VALUE=7>samedi</OPTION>
</SELECT>
```

Sie können mithilfe von Ressourcenpaketen auch ländereinstellungsspezifische Anzeigeinformationen steuern. Beispielsweise wäre dies eine alternative Methode zum Ausfüllen einer Dropdown-Liste mit den Wochentagen entsprechend dem gregorianischen Kalender:

```
<SELECT Name="DayOfWeek">
<OPTION VALUE=0><%= resourceBundle.getString("Sunday") %></OPTION>
<OPTION VALUE=1><%= resourceBundle.getString("Monday") %></OPTION>
<OPTION VALUE=2><%= resourceBundle.getString("Tuesday") %></OPTION>
<OPTION VALUE=3><%= resourceBundle.getString("Wednesday") %></OPTION>
<OPTION VALUE=4><%= resourceBundle.getString("Thursday") %></OPTION>
<OPTION VALUE=5><%= resourceBundle.getString("Friday") %></OPTION>
<OPTION VALUE=6><%= resourceBundle.getString("Saturday") %></OPTION>
</SELECT>
```

Kalenderwidget

Wenn Sie das Kalenderwidget in einer JSP-Seite verwenden möchten, müssen Sie es lokalisieren. Dazu müssen Sie die JavaScript-Datei **I18N.js** im folgenden Verzeichnis der Ländereinstellung entsprechend anpassen: ***debs_ausgangsverzeichnis*/Sterling//*la*/*CO*/js/**. Soll z. B. die Ländereinstellung "de_DE" unterstützt werden, müssen Sie eine Datei ***debs_ausgangsverzeichnis*/Sterling/de/DE/js/I18N.js** mit folgendem Inhalt erstellen:

```
// DEFAULT LOCALE (English)
var MONTH_NAMES = new Array('Januar', 'Februar', 'Maerz', 'April', 'Mai',
'Juni', 'Juli', 'August', 'September', 'Oktober', 'November', 'Dezember',
'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Okt',
'Nov', 'Dez');
var DAYOFWEEK_HEADER_NAMES = new Array("So","Mo","Di","Mi","Do","Fr","Sa");
var WEEK_START_DAY = 0;
// Create CalendarPopup object
var popupCal = new CalendarPopup();
```

Style-Sheets

Visual Modeler verwendet zum Definieren der Formatierung von HTML-Elementen Cascading Style Sheets. Wenn Sie für eine bestimmte Ländereinstellung verschiedene Schriften verwenden, müssen Sie ein Style-Sheet mit diesen Schriften erstellen. Speichern Sie dieses ländereinstellungsspezifische Style-Sheet dann für alle Ländereinstellungen an der gleichen relativen Position.

In JSP-Seiten können Sie ein ländereinstellungsspezifisches Cascading Style Sheet (z. B. **customer.css**) wie folgt integrieren:

```
<LINK rel="stylesheet" href="../css/customer.css" type="text/css">
```

Systemeigenschaften

Im Allgemeinen enthalten die Konfigurationsdateien nur Daten für Administratoren. Beim Lokalisieren dieser Dateien müssen Sie zwar keine Namen oder Werte von Elementen ändern, doch Sie sollten in Betracht ziehen, die Hilfetexte für Elemente entsprechend anzupassen. Beachten Sie, dass zu jeder Version von Visual Modeler nur eine bestimmte Anzahl von Konfigurationsdateien existiert. Daher sollten Sie die Sprache der für das Standardsystem gültigen Ländereinstellung für diese Dateien verwenden.

Ressourcenpakete und Formate

"PropertyResourceBundles" und Eigenschaftendateien

Visual Modeler macht beim Verwalten von ländereinstellungsspezifischen Daten extensiven Gebrauch von Eigenschaftendateien. Diese werden nun anstelle von Java-Klassen für Ressourcenpakete ("ResourceBundles") verwendet. Details hierzu finden Sie unter „Visual Modeler-Lokalisierung - Übersicht" auf Seite 89.

Ressourcenpakete

Ein sinnvoller Mechanismus zum Steuern der Lokalisierung ist die Verwendung von Java-Ressourcenpaketen ("ResourceBundles").

Anmerkung: Die Verwendung von Ressourcenpaketklassen in Visual Modeler wird nicht weiter unterstützt. Stattdessen sollten Sie entsprechend der Beschreibung unter „Visual Modeler-Lokalisierung - Übersicht" auf Seite 89 Eigenschaftendateien verwenden.

Dabei handelt es sich um Klassen, über die ländereinstellungsspezifische Informationen gesteuert werden. Über die in Visual Modeler verwendeten "ResourceBundle"-Klassen werden die "ListResourceBundle"-Klassen erweitert. In diesen wird die Zuordnung zwischen den Namenszeichenfolgen und den Wertezeichenfolgen definiert, die nach Aufrufen der Methode *getString (String nameString)* ausgegeben werden.

Unter Berücksichtigung der Namenskonventionen für Ressourcenpakete können Sie ländereinstellungsspezifische Ressourcenpakete für alle zu unterstützenden Ländereinstellungen erstellen. Beispielsweise können Sie die folgenden Ressourcenpakete für die Verwendung in einer neuen Anwendung mit Namen "Inventory" erstellen:

- InventoryResourceBundle
- InventoryResourceBundle_fr
- InventoryResourceBundle_fr_FR
- InventoryResourceBundle_fr_CA

Über das folgende Scriptlet kann das entsprechende Ressourcenpaket zur Verwendung in einer JSP-Seite abgerufen werden:

```
<%
String baseName = "AdvisorResourceBundle";
ResourceBundle resourceBundle =
AdvisorResourceBundle.getBundle (baseName,
session.getLocale());
%>
```

Nummernformate und Datumsformate

Mithilfe der Klasse "NumberFormat" können Sie Nummern in ländereinstellungsspezifischer Form anzeigen lassen. Sie können eine Instanz der Klasse "NumberFormat" erstellen, indem Sie die jeweilige Ländereinstellung an den Konstruktor weiterleiten.

Beispielsweise wird mit dem folgenden Scriptlet die Gesamtzahl der Einkaufswagen in einem der betreffenden Ländereinstellung entsprechenden Format angezeigt:

```
<%
NumberFormat numberFormat =
NumberFormat.getInstance(session.getLocale());
int number = request.getParameter("ShoppingCartsTotal");
%>
<P>The number of active shopping carts in use is:
<%= numberFormat.format(number) %>
</P>
```

In vergleichbarer Weise wird die Klasse "DateFormat" dazu verwendet, Sie bei der Anzeige von Datumsangaben in einem der betreffenden Ländereinstellung entsprechenden Format zu unterstützen. Sie können eine Instanz der Klasse "DateFormat" erstellen, indem Sie die jeweilige Ländereinstellung an den Konstruktor weiterleiten.

Beispielsweise wird mit dem folgenden Scriptlet das aktuelle Datum in einem der betreffenden Ländereinstellung entsprechenden Format angezeigt:

```
<%
DateFormat dateFormat =
DateFormat.getInstance(session.getLocale());
Date todaysDate = new Date();
%>
<P>It is now:
<%= dateFormat.format(todaysDate) %>
</P>
```

Kapitel 16. Steuerelemente anpassen

Über Steuerelemente wird festgelegt, wie Optionsklassen und Optionsartikel angezeigt werden sollen und wie deren Verhalten in der Benutzerschnittstelle sein soll. Sie können ein bereits vorhandenes Steuerelement modifizieren oder ein neues Steuerelement erstellen.

Jedes Steuerelement entspricht einer JSP-Seite und dem Verhalten der Optionsartikel. Diese Entsprechung wird in der Konfigurationsdatei `controls.properties` im Ordner `Comergent/WEB-INF/properties` im Implementierungsverzeichnis definiert.

Es folgt ein in der Datei `controls.properties` definierter Mustereintrag:

```
RADIO.name=Radio Button
RADIO.jsp=controls/radio.jsp
RADIO.behavior=single
```

In diesem Beispiel wird beim Steuerelement "Radio Button" die JSP-Seite `radio.jsp` dazu verwendet, die Optionsklasse in der Benutzerschnittstelle wiederzugeben. Über die Eigenschaft `behavior` wird festgelegt, wie Sterling Configurator bei diesem Steuerelement mit Entnahmen umgehen soll. Entsprechend der Definition der Eigenschaft `behavior` werden Entnahmen von Sterling Configurator wie folgt gehandhabt:

- entry - Wird bei vom Benutzer eingegebenen Steuerelementen verwendet.
- expand - Gibt an, dass bei Auswahl des Steuerelements alle untergeordneten Elemente eingeblendet werden sollen.
- multiple - Gibt an, dass über dieses Steuerelement ein oder mehrere Optionsartikel entnommen werden können.
- single - Wird ein Optionsartikel entnommen, werden alle zuvor getätigten Entnahmen aus dieser Optionsklasse entfernt.

Steuerelement ändern

Informationen zu diesem Vorgang

Sie können ein vorhandenes Steuerelement anpassen, indem Sie den entsprechenden Eintrag in der Datei `controls.properties` ändern.

So ändern Sie ein vorhandenes Steuerelement:

Vorgehensweise

1. Führen Sie das folgende Zielelement aus, um die Datei `controls.properties` zwecks Anpassung abzurufen:

   ```
   sdk customize WEB-INF/properties/controls.properties
   ```

   Durch Ausführen dieses Zielelements wird die Datei `controls.properties` in Ihrem Anpassungsprojekt platziert.
2. Ändern Sie die Einträge in der Datei `controls.properties` nach Bedarf.
3. Führen Sie das folgende Zielelement aus, um die Anpassungen in den Build aufzunehmen:

   ```
   sdk merge
   ```

4. Wenn Sie die Visual Modeler-Anwendung als WAR-Datei implementieren, führen Sie die folgenden Schritte aus:
   a. Führen Sie das folgende Zielelement aus, um die WAR-Datei neu zu erstellen:
      ```
      sdk distWar
      ```
   b. Implementieren Sie die .war-Datei auf Ihrem Anwendungsserver.

Ergebnisse

Nachdem Sie diese Schritte ausgeführt haben, müssen Sie die erforderlichen Änderungen in Sterling Selling and Fulfillment Foundation vornehmen. Weitere Informationen hierzu finden Sie unter *Sterling Configurator: Application Guide*.

Steuerelement hinzufügen

Informationen zu diesem Vorgang

Sie können ein neues Steuerelement definieren, indem Sie den Namen des betreffenden Steuerelements zu der Liste der deklarierten Steuerelemente hinzufügen und dann die Eigenschaften für das neue Steuerelement definieren.

So fügen Sie ein neues Steuerelement hinzu:

Vorgehensweise

1. Führen Sie das folgende Zielelement aus, um die Datei `controls.properties` zwecks Anpassung abzurufen:
   ```
   sdk customize WEB-INF/properties/controls.properties
   ```
   Durch Ausführen dieses Zielelements wird die Datei `controls.properties` in Ihrem Anpassungsprojekt platziert.
2. Fügen Sie den Namen des neuen Steuerelements zu der durch Kommas getrennten Liste mit Werten für das Attribut `controls` hinzu.
   Möchten Sie beispielsweise ein neues Steuerelement mit Namen "ABC_CUSTOM" hinzufügen, können Sie das Attribut `controls` wie folgt definieren:
   ```
   controls=ABC_CUSTOM,RADIO,CHECKBOX,COMBOBOX,LISTBOX,MULTISELLISTBOX,
   ALLPICKED,UEV,DISPLAY
   ```
3. Definieren Sie die Eigenschaften des neuen Steuerelements. So können Sie z. B. die Eigenschaften des neuen Steuerelements "ABC_CUSTOM" wie folgt definieren:
   ```
   ABC_CUSTOM.name=Matrix Custom Control
   ABC_CUSTOM.jsp=controls/ABCCustom.jsp
   ABC_CUSTOM.behavior=single
   ```
4. Führen Sie das folgende Zielelement aus, um die Anpassungen in den Build aufzunehmen:
   ```
   sdk merge
   ```
5. Wenn Sie die Visual Modeler-Anwendung als WAR-Datei implementieren, führen Sie die folgenden Schritte aus:
   a. Führen Sie das folgende Zielelement aus, um die WAR-Datei neu zu erstellen:
      ```
      sdk distWar
      ```
   b. Implementieren Sie die .war-Datei auf Ihrem Anwendungsserver.

Ergebnisse

Nachdem Sie diese Schritte ausgeführt haben, müssen Sie die erforderlichen Änderungen in Sterling Selling and Fulfillment Foundation vornehmen. Weitere Informationen hierzu finden Sie unter *Sterling Configurator: Application Guide*.

Kapitel 17. Funktionshandler anpassen

Funktionshandlerklassen sind Java-Klassen, über die angepasste Funktionen definiert werden können, die dann von der Sterling Configurator-Regelsteuerkomponente aufgerufen werden. Funktionshandlerklassen können von Ihnen angepasst werden.

Die Funktionshandler werden in der Konfigurationsdatei `functionHandlers.properties` im Ordner `Comergent/WEB-INF/properties` im Implementierungsverzeichnis definiert. In dieser Datei sind die Namen der verschiedenen Funktionshandler sowie das Verzeichnis, in dem sich die betreffende Funktionshandlerklasse befindet, erfasst.

Hier sehen Sie ein Musterfragment der Datei `functionHandlers.properties`:

```
WEB-INF/classes/com/comergent/apps/configurator/functionHandlers=
CheckLookupFunctionHandler,ChildSum,CountFunctionHandler,
IsSelectedHandler,LengthFunctionHandler,ListFunctionHandler,
LookupFunctionHandler,MaxFunctionHandler,MinFunctionHandler,
ParentFunctionHandler,PropValHandler,SumFunctionHandler,
ValueFunctionHandler,WebServiceLookupCheckLookupFunctionHandler=
com.comergent.apps.configurator.
function-Handlers.CheckLookupFunctionHandler
```

Funktionshandlerklasse hinzufügen

Informationen zu diesem Vorgang

Sie können eine neue Funktionshandlerklasse hinzufügen.

So fügen Sie eine neue Funktionshandlerklasse hinzu:

Vorgehensweise

1. Führen Sie das folgende Zielelement aus, um die Datei `functionHandlers.properties` zwecks Anpassung abzurufen:

   ```
   sdk customize WEB-INF/properties/functionHandlers.properties
   ```

   Durch Ausführen dieses Zielelements wird die `functionHandlers.properties` in Ihrem Anpassungsprojekt platziert.
2. Erstellen Sie über die "com.comergent.apps.configurator.functionHandlers"-Paketdeklaration eine neue Java-Klasse. In der Klassendeklaration muss deklariert werden, dass durch diese Klasse die Klasse "AbstractRuleFunctionHandler" erweitert wird.

   Anmerkung: Die neue Java-Klasse muss im Klassenpfad der Visual Modeler-Anwendung bereitgestellt werden.

   Die neue Java-Klasse muss die folgenden Methoden implementieren:

   - public String getFuncName(): Ausgabe des Funktionsnamens (z. B. "sum" oder "max"). Dabei muss die Groß-/Kleinschreibung beachtet werden: "sum" und "SUM" können mit unterschiedlichen Funktionshandlern gesteuert werden.
   - public int getType(): Ausgabe des von der Funktion zurückgegebenen Wertetyps. Dabei muss es sich um eine in der Klasse "com.comergent.api.appsservices.rulesEngine.Value" definierte Konstante

handeln. Über die Klassenmethode "AbstractRuleFunctionHandler" wird der Typ "Value.STRING" zurückgegeben. Daher ist es bei Ausgabe eines anderen Typs durch die Funktion erforderlich, diese Methode zu überschreiben.

- public Value handle(State state, String prop): Ausgabe des für die Funktion berechneten Werts.
- public boolean isPublicHandler(): Ausgabe des Werts "true", wenn der Funktionshandler von jeder beliebigen Clientanwendung verwendet werden kann. Ansonsten Ausgabe des Werts "false". Im Falle der Klassenmethode "AbstractRuleFunctionHandler" wird der Wert "true" ausgegeben. Daher muss diese Methode nur überschrieben werden, wenn es sich um einen privaten Funktionshandler handelt.

Ergebnisse

Nachdem Sie diese Schritte ausgeführt haben, müssen Sie die erforderlichen Änderungen in Sterling Selling and Fulfillment Foundation vornehmen. Weitere Informationen hierzu finden Sie unter *Sterling Configurator: Application Guide*.

Kapitel 18. Ausnahmebedingungen

"ComergentException"-Hierarchie

"Exception"-Stamm

Im vorliegenden Abschnitt werden folgende Themen behandelt:

- ComergentException
- ICCException
- ComergentRuntimeException

ComergentException

Alle in der Produktionssoftware deklarierten Ausnahmebedingungsklassen für die Kompilierzeit müssen letztendlich aus der Klasse "com.comergent.dcm.util.ComergentException" übernehmen. Durch diese Klasse wird "java.lang.Exception" dahingehend erweitert, dass Verketten und eine unabhängige Benutzernachricht bereitgestellt werden.

ICCException

Mit "ICCException" wird eine dem Benutzerkomfort dienende Unterklasse von "ComergentException" bereitgestellt. Statt also für ein Subsystem eine Gruppe von Ausnahmebedingungsklassen zu erstellen, können Sie die Klasse "ICCException" durchgehend in einem Subsystem verwenden.

ComergentRuntimeException

Alle Laufzeitausnahmebedingungsklassen müssen aus der Klasse "com.comergent.dcm.util.ComergentRuntimeException" übernehmen. Durch diese Klasse wird "java.lang.RuntimeException" dahingehend erweitert, dass identische Funktionalität bereitgestellt wird.

Subsystemgruppierung

Ein Subsystem von Visual Modeler wird immer entweder als eigene trennbare Anwendung oder als Anwendungsebene oder Systemebenenservice definiert. Bei einem Subsystem handelt es sich um eine logische Organisation. Diese kann mehrere Pakete in der Java-Pakethierarchie umfassen oder aus Teilen eines Pakets bestehen.

Für jedes logische Subsystem wird erwartet, dass in Bezug auf die Ausnahmebedingungen eine eigene Stammklasse deklariert wird. Diese Stammklasse erbt aus "ComergentException" und bildet die übergeordnete Klasse für alle Ausnahmebedingungen zur Kompilierzeit innerhalb des Subsystems. Das Subsystem wird entweder als eigene trennbare Anwendung oder als Anwendungsebene oder Systemebenenservice definiert. Bei einem Subsystem handelt es sich um eine logische Organisation. Diese kann mehrere Pakete in der Java-Pakethierarchie umfassen oder aus Teilen eines Pakets bestehen. (Trotzdem sollten Sie Ihre Paketstruktur in Übereinstimmung mit der logischen Subsystemorganisation definieren.)

Beispiel: Angenommen, es existiert ein Subsystem mit Namen "Foo". Dann muss es auch eine Klasse mit Namen `FooException` geben:

```
public class FooException extends ComergentException
{
public FooException(String msg)
{
super(msg);
}
public FooException(String msg, Exception ex)
{
super(msg, ex);
}
}
```

Angenommen, "Foo" antwortet auf einen Initialisierungsfehler durch Ausgabe von `BadInitializationException` für alle nachfolgenden Anforderungen. Diese Ausnahmebedingung würde aus der Klasse `FooException` erben:

```
public class BadInitializationException extends FooException
{
...
}
```

Subsystem nach Subsystem - Ausnahmebedingungsstrategie

Jedes Subsystem muss eine in sich konsistente Strategie zum Differenzieren von Ausnahmebedingungen implementieren. Entweder sollte dazu für jeden individuellen Ausnahmebedingungstyp eine Unterklasse zur Ausnahmebedingungsklasse des Subsystems existieren (dies ist standardmäßige Java-Strategie) oder die Stammausnahmebedingung des Subsystems sollte aus der Klasse "ICCException" erben und zum Unterscheiden zwischen den verschiedenen Ausnahmebedingungen den Statusparameter definieren (dies ist "ICCException"-Strategie).

Wenn z. B. für das Subsystem "Foo" die Java-Ausnahmebedingungsstrategie ausgewählt wird, dann wird durch die Klasse "FooException" die Klasse "ComergentException" erweitert. Wenn für das Subsystem "Bar" eine "ICCException"-Strategie ausgewählt wird, dann wird durch die Klasse "FooException" die Klasse "ICCException" erweitert (die dann wiederum die Klasse "ComergentException" erweitert).

```
public class BarException extends ICCException
{
...
}
```

Ausnahmebedingungsverkettung

Für jedes Subsystem gilt: Es werden nur Ausnahmebedingungen aus dem Subsystem selbst für das aufrufende Programm ausgelöst. Wenn durch einen zugrunde liegenden Service eine Ausnahmebedingung ausgelöst wird, die vom zuständigen Subsystem nicht bearbeitet werden kann, muss das Subsystem diese Ausnahmebedingung abfangen und erneut eine Ausnahmebedingung auslösen, die dann aber in dem Kontext eine Bedeutung hat. In dieser neuen Ausnahmebedingung muss ein Verkettungskonstruktor zum Einsatz kommen,

durch den die ursprüngliche Ausnahmebedingung einbezogen wird, sodass bei der abschließenden Bearbeitung und Protokollierung der Ausnahmebedingung die ursprüngliche Ausnahmebedingung nicht verloren geht.

Beispiel: Angenommen, durch das Subsystem "Foo" wird versucht, eine Eigenschaftendatei zu öffen und dabei kommt es zu einer Ein-/Ausgabeausnahmebedingung. Wird eine Java-Ausnahmebedingungsstrategie implementiert, kann eine neue Ausnahmebedingungsklasse "FooPropertyFileException" deklariert werden, durch die die Klasse "FooException" erweitert wird. Durch die Anweisung zum Abfangen der Ein-/Ausgabeausnahmebedingung wird dann eine neue Ausnahmebedingung "FooPropertyFileException" mit einem Konstruktor ausgelöst, durch den eine entsprechende Nachricht und die ursprüngliche E/A-Ausnahmebedingung weitergeleitet wird.

```
try
{
...
Properties props = new Properties();
props.load(input);
...
}
catch (IOException ex)
{
// chain the io exception
throw new FooPropertyFileException("Loading file" + filename, ex);
}
```

Ausnahmebedingungen auslösen, abfangen und protokollieren

Ausnahmebedingungen auslösen

Ausnahmebedingungen können immer dann ausgelöst werden, wenn die "Vereinbarungen" zwischen einer Methode und dem aufrufenden Programm nicht erfüllt werden können. Diese Verwendung entspricht der Festlegung in der Java Language Specification. Leider kann daraus nur wenig abgeleitet werden, denn die genannten "Vereinbarungen" können entweder so umfassend formuliert sein, dass Ausnahmebedingungen erst gar nicht entstehen oder sie werden so eng gefasst, dass es sehr häufig zu Ausnahmebedingungen kommt. Generell sollten mit Ausnahmebedingungen die folgenden einander entgegengesetzten Zielsetzungen gleichberechtigt verfolgt werden:

Ausnahmebedingungen dürfen nicht zur Regel werden.

- Zu Ausnahmebedingungen gehört das Erstellen eines zusätzlichen Objekts. Daher ist es schon allein unter Leistungsaspekten problematisch, wenn es häufig zu Ausnahmebedingungen kommen kann.
- Ein Vermischen von Daten und Steuerelementen ist zu vermeiden. Als Alternative zum Auslösen einer Ausnahmebedingung bietet sich häufig das Rückmelden eines Nullwerts durch eine Methode an. Das heißt, dass in dem Rückgabewert zwei Bedeutungen eingebunden sind (Erfolg oder Misserfolg oder was immer die Daten bei der Ausgabe auch bedeuten mögen). Es ist gute Programmiertradition, diese Art der Verwendung (wenn möglich) zu vermeiden.
- Wenn der Wert null für den von einer Methode verfolgten Zweck ein angemessener Wert ist oder wenn in Bezug auf eine bestimmte Methode erwartet wird, dass diese im normalen Betrieb häufig fehlschlägt, dann ist es

sinnvoll, den Fehlschlag über den Wert null anzugeben. Ansonsten wäre es besser, eine Ausnahmebedingung auszulösen.

Ausnahmebedingungen zur Lauf- oder Kompilierzeit auslösen

Entsprechend der Festlegung in der Java Language Specification sollten Ausnahmebedingungen zur Laufzeit immer dann ausgelöst werden, wenn durch das aufrufende Programm fehlerhafte Eingabedaten bereitgestellt (und damit im Prinzip die die Methode betreffenden "Vereinbarungen" gebrochen) werden und es umständlich wäre, eine Ausnahmebedingungen zur Kompilierzeit auszulösen. Wenn durch ein aufrufendes Programm z. B. eine Methode aufgerufen wird, durch die ein negativer Wert an einen Parameter weitergegeben wird, bei dem es sich um einen Feldgruppenindex handelt, ist es angemessen, eine Laufzeitausnahmebedingung auszulösen. Ansonsten wird eine Kompilierzeitausnahmebedingung ausgelöst.

Abfangklauseln und Auslösedeklarationen

Abfangklauseln und Auslösedeklarationen sollten nicht zu allgemein gehalten sein. Wenn durch die aufgerufene Methode z. B. eine Ausnahmebedingung des Typs "FileNotFoundException" ausgelöst wird, dann muss durch das aufrufende Programm die Ausnahmebedingung "FileNotFoundException" und nicht "Exception" oder "Throwable" abgefangen werden. Wenn sich nämlich der zugrunde liegende Code dahingehend ändert, dass eine neue Ausnahmebedingung ausgelöst oder dass das Auslösen dieser Ausnahmebedingung beendet wird, dann ist es wünschenswert, dass es durch die Änderung zu einem Kompilierungsfehler kommt, durch den der Programmierer dazu veranlasst wird, von einer neuen Situation auszugehen.

Wo es auf Anwendbarkeit ankommt, gibt es Ausnahmen von dieser Regel. Wenn es eine große Vielfalt an auszulösenden Ausnahmebedingungen gibt und für alle Fälle die gleiche Antwort gilt, gibt es keine Veranlassung, alle einzeln abzufangen.

Ausnahmebedingungen protokollieren

Wenn von einer Methode eine Ausnahmebedingung abgefangen und bearbeitet wird (d. h. die Ausnahmebedingung wird nicht erneut ausgelöst), sollte eine Protokollierung erfolgen. Vermutlich ist der Methode die Signifikanz der betreffenden Ausnahmebedingung bekannt und sie ist daher in der Lage, diese mit der Wertigkeit eines Fehlers oder einer anderen niedrigeren Wertigkeit zu protokollieren. Leere Abfanganweisungen sind immer mit Vorsicht zu betrachten.

Folgende Angaben sind unbedingt zu vermeiden:

```
catch (SomeException ex)
{
}
```

Gehen Sie stattdessen so:

```
catch (SomeException ex)
{
Global.logVerbose(ex);
}
```

Oder so vor:

```
catch (SomeException ex)
{
ex.printStackTrace(Global.debugStream);
}
```

Wenn Ausnahmebedingungen von zugrunde liegenden Subsystemen oder aus Paketen anderer Anbieter abgefangen werden und mit einer neuen Ausnahmebedingung verkettet werden, gibt es keine Veranlassung, diese Ausnahmebedingung zu protokollieren. Möglicherweise wird sie durch einen in der Hierarchie weiter oben angesiedelten Prozess abgefangen und verarbeitet, wobei eine Protokollierung innerhalb dieses Prozesses erfolgt.

Ausnahmebedingungen anzeigen

In der Regel werden den Benutzern von Visual Modeler Ausnahmebedingungen nicht angezeigt. Es ist Aufgabe des zugrunde liegenden Subsystems, ordnungsgemäß mit einer Ausnahmebedingung umzugehen, indem entsprechend auf die Fehlerbedingung reagiert wird.

Auf den Visual Modeler-Fehlerseiten wird die Stack-Trace für die Ausnahmebedingungen zwischen HTML-Kommentaren platziert. Durch Betrachten der Quelle zur angezeigten Webseite können Sie die Stack-Trace lesen.

Wenn eine Stack-Trace für Ausnahmebedingungen an eine JSP-Seite weitergeleitet wird, müssen Sie berücksichtigen, dass möglicherweise durch die Grenzwerte bezüglich des JSP-Seitenpuffers verhindert wird, dass eine komplette Ausnahmebedingungsnachricht an die Webseite übertragen wird. Wird eine lange Stack-Trace für Ausnahmebedingungen an eine JSP-Seite übertragen, können Sie sie anzeigen, indem Sie am Puffer der betreffenden JSP-Seite entsprechende Änderungen vornehmen. Verwenden Sie den Tag "buffer" wie folgt:

```
<%@ page buffer=1024kb %>
```

Sobald die Fehlerbedingung diagnostiziert und korrigiert wurde, sollten Sie den Tag wieder entfernen, da er Einfluss auf das Leistungsverhalten hat.

Kapitel 19. Cron-Jobs

Cron-Jobs in Visual Modeler implementieren

Bestimmte Tasks innerhalb einer Implementierung von Visual Modeler werden nicht als Reaktion auf eine Benutzereingabe initialisiert. So wird z. B. die stündliche Synchronisation von Auftragsdaten mit einem externen System oder der wöchentliche Import der Katalogdaten eines anderen Anbieters vorzugsweise ohne Benutzereingriff ausgeführt. Die Ausführung dieser Jobs in passenden Intervallen können Sie mithilfe der in Visual Modeler bereitgestellten "Job-Scheduler"-Funktion planen.

Cron-Jobs können als System-Cron-Jobs oder als Anwendungs-Cron-Jobs definiert werden.

- Ein System-Cron-Job wird von Visual Modeler ausgeführt und hat keinen Bezug zu einem Benutzer. Von einem System-Cron-Job werden die Visual Modeler-Klassen direkt aufgerufen. Ein System-Cron-Job muss über eine Klasse ausgeführt werden, durch die die abstrakte Klasse "SystemCron" erweitert wird. In der Regel werden von System-Cron-Jobs Tasks wie die Bereinigung des Cachespeichers ausgeführt.
- Anwendungs-Cron-Jobs werden im Namen von Benutzern ausgeführt: Benutzername und Kennwort des betreffenden Benutzers werden bei Erstellung des Cron-Jobs über die "Job-Scheduler"-Benutzerschnittstelle bereitgestellt. Im Rahmen von Anwendungs-Cron-Jobs werden XML-Nachrichten an Visual Modeler übergeben und dann vom System verarbeitet. Ein Anwendungs-Cron-Job muss über eine Klasse ausgeführt werden, durch die die abstrakte Klasse "ApplicationCron" erweitert wird. In der Regel werden Anwendungs-Cron-Jobs dazu verwendet, die notwendigen Verwaltungstasks auszuführen, die sich (wie die Synchronisation von Auftragsdaten) auf Benutzer- oder Produktdaten beziehen.

  Anmerkung: Von einem System-Cron-Job selbst dürfen keine Operationen des Typs *restore()* und *persist()* ausgeführt werden. Der Cron-Job-Klasse ist nämlich kein Benutzer zugeordnet, sodass durch die in die Datenzugriffsmethoden integrierte Zugriffsprüfung eine Ausnahmebedingung ausgelöst wird.

Die Klassen "CronManager" und "CronScheduler"

Die Definition und Erstellung von Cron-Jobs erfolgt über die Klasse "CronManager". Die Cron-Job-Konfigurationsdaten werden im Speicher durch die Data-Bean "CronConfigBean" dargestellt. Die Definition der Cron-Jobs wird über die Wissensdatenbank gesteuert.

Die Terminierung und Ausführung von Cron-Jobs erfolgt über die Klasse "CronScheduler". Diese Singleton-Klasse wird beim Serverstart instanziiert.

"CronJob"-Schnittstelle

Jeder Cron-Job entspricht einer Java-Klasse, über die die "CronJob"-Schnittstelle implementiert wird:

```
public interface CronJob extends java.lang.Runnable
{
/**
* Specify the Cron Configuration bean object.
*
* @param config Cron configuration bean object.
*/
public void setCronConfiguration(CronConfigBean config);
/**
* Return the Cron Configuration bean object.
*
* @return CronConfigBean object.
*/
public CronConfigBean getCronConfiguration();
/**
* Initialization function. This function is called
* immediately after the object is created.
*
* @return true if initialization success, false otherwise.
*/
public boolean init();
/**
* Return the current scheduled time.
*
* @return Current schedule time in Calendar object.
*/
public Calendar getSchedule();
/**
* Reschedule the cron to reflect the changes made to the
* cronfiguration parameter. This function is called by the
* Cron Manager whenever cron configuration changes.
*/
public void reschedule();
/**
* Whether the job needs to be run again. This function is
* useful if there is some problem in the current run and you
* want to retry at specified time.
*
* @return true if the job is allowed to retry if the job
* did not run successfully
* on the last time of execution
*/
public boolean retry();
/**
* Determines whether to stop this cron job from running.
*
* @return true if the job has been slated to not run again
*/
public boolean stopRun();
/**
* Compute next cron run time: this is usually based on the cron
* run interval.
*/
public void computeNextSchedule();
/**
* Check to determine if the cron job is
```

```
* in a good state to run before triggering the thread to run.
*
* @return true or false.  True means ready to run.
*/
public boolean isOKtoRun();
/**
* Is called when the thread starts.
*
* @return false if the job needs to be stopped. Return true to
* continue running.
*/
public boolean service();
/**
* Checks whether the next run time is later than the end run date.
*
* @return true if next run time greater than end run time
*/
  public boolean isExpired();
}
```

Cron-Job in Visual Modeler erstellen

Informationen zu diesem Vorgang

Führen Sie die folgenden Schritte aus, um einen neuen Cron-Job zu erstellen:

Vorgehensweise

1. Erstellen Sie eine "CronJob"-Klasse: Dazu muss entweder die Klasse "SystemCron" oder die Klasse "ApplicationCron" erweitert werden. Bei beiden Klassen handelt es sich um abtrakte Klassen, die die abstrakte Klasse "AbstractCronJob" erweitern.

   Die einzige Methode, die Sie implementieren müssen, ist die Methode *service()*. Dabei handelt es sich um die Methode, über die die von der Klasse "CronScheduler" initiierten ankommenden Beiträge verarbeitet werden.

   - Wenn dem Job Parameter übergeben werden, die mithilfe der "Job-Scheduler"-Benutzerschnittstelle definiert werden, können Sie diese Parameter mit den Methoden *getParameter(String s)* und *getParameters()* der Klasse "AbstractCronJob" abrufen. Diese Methoden verhalten sich identisch zu den entsprechenden Methoden der Klasse "HttpServletRequest".
   - Wenn Sie das Ergebnis des Jobs in der Datenbank speichern lassen möchten, muss über die Methode *service()* die Methode *setExecutionOutcome(String s)* aufgerufen werden.
   - Sie können angeben, dass der Cron-Job zu einem späteren Zeitpunkt erneut ausgeführt werden soll. Dazu müssen Sie die Methode *setRetry(Calendar c)* der Klasse "AbstractCronJob" aufrufen. Verwenden Sie den Parameter "Calendar", um anzugeben, wann der Job erneut ausgeführt werden soll.
2. Definieren Sie den Cron-Job mithilfe der als Bestandteil der Systemverwaltungsanwendung bereitgestellten "Job-Scheduler"-Benutzerschnittstelle. Geben Sie dazu die Cron-Job-Klasse, die zeitliche Planung für die Ausführung sowie alle Parameter an, die zur Laufzeit an den Cron-Job übergeben werden sollen. Wenn der Cron-Job als Anwendungs-Cron-Job ausgeführt werden soll, müssen Sie zusätzlich den Benutzernamen und das Kennwort des jeweiligen Benutzers bereitstellen.

Die Übergabe von Parametern an den Cron-Job erfolgt mithilfe derselben Syntax wie bei HTTP-Anforderungsparameter. Beispiel: Name1=Value1 &Name2=Value2.

Kapitel 20. Filter - Übersicht

Ein Filter ist ein Objekt, über das Filterungstasks in Bezug auf die Anfrage an eine Ressource (Servletinhalt oder statischer Inhalt) oder die Antwort durch eine Ressource oder beide ausgeführt werden. Diese werden als Teil der J2EE 2.3-Spezifikation definiert.

Eine Filterung erfolgt im Rahmen der Methode *doFilter()*. Jeder Filter hat Zugriff auf ein "FilterConfig"-Objekt, über das die erforderlichen Initialisierungsparameter bezogen werden können. Dabei handelt es sich um einen Bezug zum Servlet-Kontext ("ServletContext"), der z. B. dazu verwendet werden kann, die für die Filterungstasks erforderlichen Ressourcen zu laden.

Filter werden im Implementierungsdeskriptor einer Webanwendung konfiguriert. Beispiele für typische Filter sind (u. a.):

- Authentifizierungsfilter
- Protokollierungs- und Überwachungsfilter
- Bildkonvertierungsfilter
- Datenkomprimierungsfilter
- Verschlüsselungsfilter
- Filter zum Zerlegen in Tokens
- Filter zum Auslösen von Ressourcenzugriffereignissen
- XSLT-Filter
- MIME-Typ-Verkettungsfilter

Kapitel 21. Visual Modeler-Filter

Visual Modeler bietet die folgenden Filter, die Bestandteil des "com.comergent.dcm.core.filters"-Pakets sind:

- „DosFilter”
- „WSDLFilter”

DosFilter

Dieser Filter kann als Basis für Filter zum Schutz der Webanwendung vor Denial-of-Service-Attacken verwendet werden.

Erstellen Sie zum Verwenden dieses Filters eine Klasse, über die die Klasse "com.comergent.dcm.core.filters.DosFilter" erweitert wird. Setzen Sie darin die Methode *isRequestDenied()* außer Kraft, um die Logik zu implementieren, über die Denial-of-Service-Attacken aufgedeckt und blockiert werden sollen.

Modifizieren Sie dann die Konfigurationsdatei **web.xml** dahingehend, dass Sie Ihre Implementierungsklasse wie folgt als einen Filter deklarieren:

```
<filter>
<filter-name>DosFilter</filter-name>
<filter-class>
com.comergent.dcm.messaging.CustomDosFilter
</filter-class>
</filter>
```

```
und
<filter-mapping>
   <filter-name>DosFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

WSDLFilter

Die Klasse "WSDLFilter" wird dazu verwendet, die Web-Service-WSDLs zu transformieren, wenn auf diese über die standardmäßigen URLs (http://server:port/s/dXML/5.0/OrderInterface.wsdl) etc. zugegriffen wird.

Kapitel 22. Eingeschränkte Felder verwalten und anzeigen

Eingeschränkte Datenfelder können immer nur einen von mehreren Werten aufnehmen. Beispiele dafür sind (u. a.) Partnerstufen (wie “Gold”, “Silber” etc.), Partnergebiete (wie “Nordwest”, “Benelux” etc.) und Fertigkeitsstufen (wie “Experte”, “Mit Zertifizierung” etc.). Sie können diese Datenfelder in Visual Modeler auf unterschiedliche Weise steuern. Für welche Option Sie sich entscheiden, hängt davon ab, wie die Felder verwaltet und verwendet werden sollen.

Optionen

Zur Angabe eines eingeschränkten Datenfelds und der dafür zulässigen Datenwerte stehen Ihnen die folgenden Optionen zur Verfügung:

- Verwalten des Datenfelds als Gruppe von Werten in einer Datenbanktabelle. Zuordnen von Werten zu Geschäftsobjekten entweder über eine Querverweistabelle oder (für jeden Wert in der Geschäftsobjekttabelle) über den Bezug auf einen Schlüssel.
- Verwalten der Werte als Einschränkungselement im XML Schema (Deklaration in der Datei **DsConstraints.xml**). Angeben der Einschränkung als Attribut des zum Datenfeld gehörigen Datenelements.
- Einbetten der zulässigen Werte als Werte eines <SELECT>-Maskenelements in einer HTML-Schablone.

Es wird empfohlen, die für ein Feld zulässigen Werte als Datenbanktabelle zu verwalten, es sei denn:

- Die Werte sollen zur Laufzeit nicht modifiziert werden.
- Das Datenfeld kann in jedem Geschäftsobjekt immer nur einen Wert aufnehmen.
- Die Werte können in einer natürlichen Reihenfolge angezeigt werden, die sich aus den Werten selbst (z. B. ihrer alphabetischen Reihenfolge) ergibt.

Aus folgenden Gründen wird von der Verwendung der dritten Option abgeraten:

- Wenn Sie die Liste der zulässigen Datenwerte modifizieren möchten, ergibt sich daraus ein Wartungsproblem hinsichtlich der Aktualisierung von Schablonen oder Anwendungscode.
- Da Benutzer möglicherweise Änderungen am HTML-Code vornehmen, aufgrund derer unzulässige Werte zurückgegeben werden, ergibt sich daraus ein Sicherheitsproblem. Daher müssen Sie zum Überprüfen der Auswahl entweder Javascript hinzufügen (kann vom Benutzer entfernt werden) oder den zurückgegebenen Wert als Teil der Geschäftslogik überprüfen lassen.

Kriterien

Welche Auswahl Sie treffen, hängt von der Funktionalität des betreffenden Datenfelds ab. Um zu ermitteln, wie ein Datenfeld verwendet werden soll, beantworten Sie die folgenden Fragen:

1. Können Sie einem Geschäftsobjekt nur einen oder mehrere Werte aus einem eingeschränkten Datenfeld zuordnen?

   Wenn Sie hier antworten, dass ein und demselben Geschäftsobjekt mehrere Werte (Beispiel: ein Partner, der in mehreren Bereichen tätig ist) zugeordnet

werden können, dann *müssen* Sie für die Feldwerte eine Datenbanktabelle und für die Zuordnung der Werte zum Geschäftsobjekt eine Querverweistabelle verwenden.

2. Können Sie beim Erstellen eines neuen Geschäftsobjekts neue Werte des Datenfelds eingeben oder müssen Sie überprüfen lassen, ob ein für das Datenfeld eingegebener Wert zur Gruppe der eingeschränkten Werte gehört?

   Wenn nur Einzelwerte zulässig sind und Ihre Antwort auf Frage 2 lautet, dass neue Werte zulässig sind, dann *müssen* Sie für die Feldwerte eine Datenbanktabelle verwenden. Es ist aber keine Querverweistabelle für die Zuordnung der Datenfeldwerte zum Geschäftsobjekt erforderlich. Sie können über die aktuelle Visual Modeler-Schnittstelle nicht dynamisch Werte zur Liste der zulässigen Werte für ein Einschränkungselement hinzufügen.

   Werden die für ein eingeschränktes Datenfeld möglichen Werte dynamisch verwaltet oder werden sie einmal beim Start eingelesen?

3. Wenn Ihre Antwort auf Frage 1 Einzelwert lautete und Sie bei Frage 2 geantwortet haben, dass neue Werte nicht zulässig sind, Sie aber dynamische Aktualisierungen benötigen, dann *müssen* Sie eine Datenbanktabelle verwenden. Wenn die eingeschränkten Werte nach dem Start von Visual Modeler unverändert bleiben, können Sie ein Einschränkungselement verwenden.

   Müssen Sie die eingeschränkten Datenwerte für die Anzeige sortieren? Ist das der Fall, findet diese Sortierung dann nach Wert (etwa alphabetisch) oder einer anderweitig definierten Reihenfolge statt, die nicht aus den Werten selbst abgeleitet werden kann?

4. Wenn die Datenfeldwerte in einer Reihenfolge sortiert werden müssen, die sich nicht aus den Werten selbst erschließt, müssen diese Angaben zur Sortierung in einer Datenbanktabelle verwaltet werden. Wenn Sie die Werte allerdings nur in einer Reihenfolge sortieren möchten, die sich aus den Werten selbst erschließt (z. B. alphabetisch), können Sie die Option für ein Einschränkungselement verwenden.

Index

F

G

H

I

J

K

T

U

V

W

X

Z

Bemerkungen

Die vorliegenden Informationen wurden für Produkte und Services entwickelt, die auf dem deutschen Markt angeboten werden.

Möglicherweise bietet IBM die in dieser Dokumentation beschriebenen Produkte, Services oder Funktionen in anderen Ländern nicht an. Informationen über die gegenwärtig im jeweiligen Land verfügbaren Produkte und Services sind beim zuständigen IBM Ansprechpartner erhältlich. Hinweise auf IBM Lizenzprogramme oder andere IBM Produkte bedeuten nicht, dass nur Programme, Produkte oder Services von IBM verwendet werden können. Anstelle der IBM Produkte, Programme oder Services können auch andere, ihnen äquivalente Produkte, Programme oder Services verwendet werden, solange diese keine gewerblichen oder anderen Schutzrechte von IBM verletzen. Die Verantwortung für den Betrieb von Produkten, Programmen und Services anderer Anbieter liegt beim Kunden.

Für die in diesem Handbuch beschriebenen Erzeugnisse und Verfahren kann es IBM Patente oder Patentanmeldungen geben. Mit der Auslieferung dieses Handbuchs ist keine Lizenzierung dieser Patente verbunden. Lizenzanforderungen sind schriftlich an folgende Adresse zu richten (Anfragen an diese Adresse müssen auf Englisch formuliert werden):

IBM Director of Licensing

IBM Corporation

Tour Descartes 2, avenue Gambetta 92066 Paris La Defense

France

Trotz sorgfältiger Bearbeitung können technische Ungenauigkeiten oder Druckfehler in dieser Veröffentlichung nicht ausgeschlossen werden. Die hier enthaltenen Informationen werden in regelmäßigen Zeitabständen aktualisiert und als Neuausgabe veröffentlicht. IBM kann ohne weitere Mitteilung jederzeit Verbesserungen und/oder Änderungen an den in dieser Veröffentlichung beschriebenen Produkten und/oder Programmen vornehmen.

Verweise in diesen Informationen auf Websites anderer Anbieter werden lediglich als Service für den Kunden bereitgestellt und stellen keinerlei Billigung des Inhalts dieser Websites dar. Das über diese Websites verfügbare Material ist nicht Bestandteil des Materials für dieses IBM Produkt. Die Verwendung dieser Websites geschieht auf eigene Verantwortung.

Werden an IBM Informationen eingesandt, können diese beliebig verwendet werden, ohne dass eine Verpflichtung gegenüber dem Einsender entsteht.

Lizenznehmer des Programms, die Informationen zu diesem Produkt wünschen mit der Zielsetzung: (i) den Austausch von Informationen zwischen unabhängig voneinander erstellten Programmen und anderen Programmen (einschließlich des vorliegenden Programms) sowie (ii) die gemeinsame Nutzung der ausgetauschten Informationen zu ermöglichen, wenden sich an den Hersteller.

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA 95141-1003

U.S.A.

Die Bereitstellung dieser Informationen kann unter Umständen von bestimmten Bedingungen - in einigen Fällen auch von der Zahlung einer Gebühr - abhängig sein.

Die Lieferung des in diesen Informationen beschriebenen Lizenzprogramms sowie des zugehörigen Lizenzmaterials erfolgt auf der Basis der IBM Rahmenvereinbarung bzw. der Allgemeinen Geschäftsbedingungen von IBM, der IBM Internationalen Nutzungsbedingungen für Programmpakete oder einer äquivalenten Vereinbarung.

Alle in diesem Dokument enthaltenen Leistungsdaten stammen aus einer kontrollierten Umgebung. Die Ergebnisse, die in anderen Betriebsumgebungen erzielt werden, können daher erheblich von den hier erzielten Ergebnissen abweichen. Einige Daten stammen möglicherweise von Systemen, deren Entwicklung noch nicht abgeschlossen ist. Eine Gewährleistung, dass diese Daten auch in allgemein verfügbaren Systemen erzielt werden, kann nicht gegeben werden. Darüber hinaus wurden einige Daten unter Umständen durch Extrapolation berechnet. Die tatsächlichen Ergebnisse können davon abweichen. Benutzer dieses Dokuments sollten die entsprechenden Daten in ihrer spezifischen Umgebung prüfen.

Alle Informationen zu Produkten anderer Anbieter stammen von den Anbietern der aufgeführten Produkte, deren veröffentlichten Ankündigungen oder anderen allgemein verfügbaren Quellen. IBM hat diese Produkte nicht getestet und kann daher keine Aussagen zu Leistung, Kompatibilität oder anderen Merkmalen machen. Fragen zu den Leistungsmerkmalen von Produkten anderer Anbieter sind an den jeweiligen Anbieter zu richten.

Aussagen über Pläne und Absichten von IBM unterliegen Änderungen oder können zurückgenommen werden und repräsentieren nur die Ziele von IBM.

Alle von IBM angegebenen Preise sind empfohlene Richtpreise und können jederzeit ohne weitere Mitteilung geändert werden. Händlerpreise können u. U. von den hier genannten Preisen abweichen.

Diese Veröffentlichung dient nur zu Planungszwecken. Die in dieser Veröffentlichung enthaltenen Informationen können geändert werden, bevor die beschriebenen Produkte verfügbar sind.

Diese Veröffentlichung enthält Beispiele für Daten und Berichte des alltäglichen Geschäftsablaufs. Sie sollen nur die Funktionen des Lizenzprogramms illustrieren und können Namen von Personen, Firmen, Marken oder Produkten enthalten. Alle diese Namen sind frei erfunden; Ähnlichkeiten mit tatsächlichen Namen und Adressen sind rein zufällig.

COPYRIGHTLIZENZ:

Diese Veröffentlichung enthält Musteranwendungsprogramme, die in Quellensprache geschrieben sind und Programmiertechniken in verschiedenen Betriebsumgebungen veranschaulichen. Sie dürfen diese Musterprogramme kostenlos kopieren, ändern und verteilen, wenn dies zu dem Zweck geschieht, Anwendungsprogramme zu entwickeln, zu verwenden, zu vermarkten oder zu verteilen, die mit der Anwendungsprogrammierschnittstelle für die Betriebsumgebung konform sind, für die diese Musterprogramme geschrieben werden. Diese Beispiele wurden nicht unter allen denkbaren Bedingungen getestet. Daher kann IBM die Zuverlässigkeit, Wartungsfreundlichkeit oder Funktion dieser Programme weder zusagen noch gewährleisten. Die Musterprogramme werden ohne Wartung (auf "as-is"-Basis) und ohne jegliche Gewährleistung zur Verfügung gestellt. IBM übernimmt keine Haftung für Schäden, die durch die Verwendung der Musterprogramme entstehen.

Kopien oder Teile der Musterprogramme bzw. daraus abgeleiteter Code müssen folgenden Copyrightvermerk beinhalten:

Wird dieses Buch als Softcopy (Book) angezeigt, erscheinen keine Fotografien oder Farbabbildungen.

Marken

IBM, das IBM Logo und ibm.com sind Marken oder eingetragene Marken der International Business Machines Corporation. Weitere Produkt- und Servicenamen können Marken von IBM oder anderen Unternehmen sein. Eine aktuelle Liste der IBM Marken finden Sie auf der Webseite “Copyright and trademark information” unter http://www.ibm.com/legal/copytrade.shtml.

Adobe, das Adobe-Logo, PostScript und das PostScript-Logo sind Marken oder eingetragene Marken der Adobe Systems Incorporated in den USA und/oder anderen Ländern.

IT Infrastructure Library ist eine eingetragene Marke der Central Computer and Telecommunications Agency. Die Central Computer and Telecommunications Agency ist nunmehr in das Office of Government Commerce eingegliedert worden.

Intel, das Intel-Logo, Intel Inside, das Intel Inside-Logo, Intel Centrino, das Intel Centrino-Logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium und Pentium sind Marken oder eingetragene Marken der Intel Corporation oder ihrer Tochtergesellschaften in den USA und anderen Ländern.

Linux ist eine eingetragene Marke von Linus Torvalds in den USA und/oder anderen Ländern.

Microsoft, Windows, Windows NT und das Windows-Logo sind Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

ITIL ist als eingetragene Marke und eingetragene Gemeinschaftsmarke des Office of Government Commerce beim US Patent and Trademark Office registriert.

UNIX ist eine eingetragene Marke von The Open Group in den USA und anderen Ländern.

Java und alle auf Java basierenden Marken und Logos sind Marken oder eingetragene Marken der Oracle Corporation und/oder ihrer verbundenen Unternehmen.

Cell Broadband Engine wird unter Lizenz verwendet und ist eine Marke der Sony Computer Entertainment, Inc. in den USA und/oder anderen Ländern.

Linear Tape-Open, LTO, das LTO-Logo, Ultrium und das Ultrium-Logo sind Marken von HP, der IBM Corp. und Quantum in den USA und anderen Ländern.

Connect Control Center®, Connect:Direct®, Connect:Enterprise, Gentran®, Gentran:Basic®, Gentran:Control®, Gentran:Director®, Gentran:Plus®, Gentran:Realtime®, Gentran:Server®, Gentran:Viewpoint®, Sterling Commerce™, Sterling Information Broker® und Sterling Integrator® sind Marken oder eingetragene Marken der Sterling Commerce, Inc., einem IBM Unternehmen.

Weitere Unternehmens-, Produkt- und Servicenamen können Marken oder Servicemarken anderer Hersteller sein.

Gedruckt in Deutschland