

JAVA FORUM 2004

stuttgart



HOB GmbH & Co. KG

Schwadmühlstraße 3

90556 Cadolzburg

Tel. 0049 (0) 9103/715-0

Fax 0049 (0) 9103/715-271

E-Mail: marketing@hob.de

Referent: **Gerhard Fleischmann**

Entwicklung Datenbank - Connectivity

HOB
SOFTWARE



Thema:

Dynamische Webinhalte mittels
JDBC-Zugriff auf Large Objects
(LOBs) einer DB2



- **Anwendungsentwicklung für Application Server am Beispiel Websphere**
- **Einsatzmöglichkeit für JDBC-Treiber**
- **Verwendung von Large Objects (LOBs)**
- **Ein Ansatz für dynamische Webseitengestaltung**

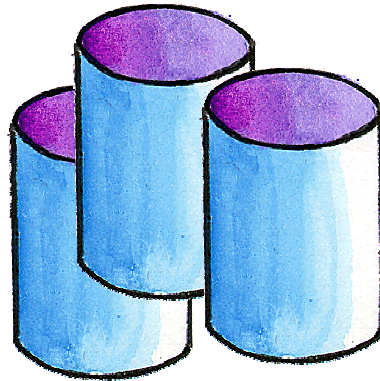


- **Zielsetzung für die Anwendung**
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



Welchen Zweck erfüllt die Anwendung?

Eine Datenbank enthält Informationen und Bilder zu Automobil Modellen, die dem Anwender in einem Browser zur Betrachtung zur Verfügung gestellt werden sollen.



Die Anwendung soll dem Benutzer die Auswahl eines Automobil Modells und Einstellungen zur Darstellung der Informationen ermöglichen.



- Zielsetzung für die Anwendung
- **Kurze Präsentation der Beispielanwendung**
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion

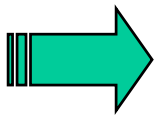


- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- **Die Architektur**
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



Grundgedanke:

Unabhängigkeit von Zugriff / Bereitstellung der Daten und deren
Visualisierung

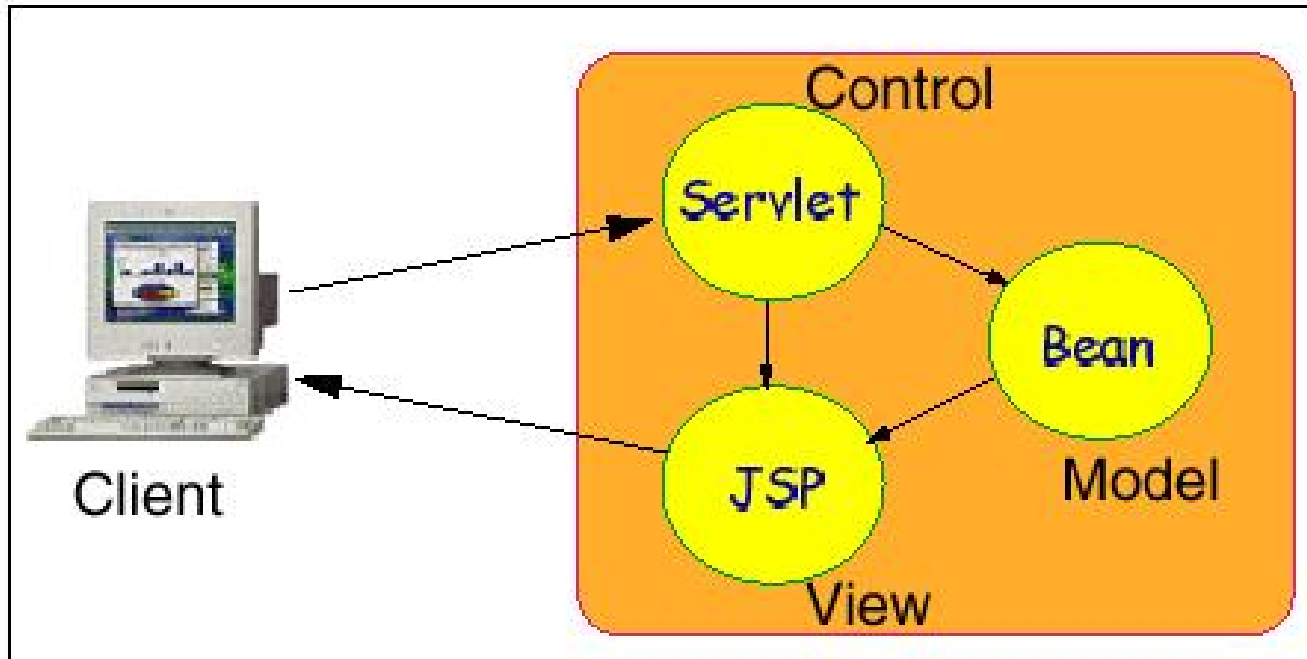


bei späterer Änderungen und Erweiterungen sind Datenmodell und
Oberfläche unabhängig voneinander austauschbar



Architektur – Model-View-Control Pattern

Design Pattern zur Trennung von Oberfläche und Datenmodell



Controller: Steuert, auf welches Datenmodell zugegriffen wird und welches JSP die Daten darstellen soll



HOB
SOFTWARE

- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- **Entwicklungs- und Laufzeitumgebung**
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



Die Umgebung – Was benötige ich?

■ Browser

Minimale Client-Anforderungen für den Benutzer
(nur Java-Plugin, JVM)

■ JAVA als Programmiersprache

- Servlet bedient Browser mit Daten
- Plattformunabhängig
- Datenbank-Zugriff aus Java-Klassen über JDBC

■ Entwicklungsumgebung

Erstellung unterschiedlichster Komponenten => 1 Tool
z.B. Borland JBuilder, Sun ONE Studio, Visual Cafe...
(hier: IBM Websphere Application Developer)



Die Umgebung – Was benötige ich?

- **Application Server**

Zentrale serverseitige Plattform für die Bereitstellung der Anwendung (hier verwendet: IBM Websphere Application Server)

- **Datenbank**

Zentrale Speicherung und Verwaltung der darzustellenden Daten und Grafikobjekte => Large Objects (LOBs)
(hier verwendet: IBM DB2 V8.1 unter Windows XP)

- **JDBC-Treiber**

Dient Java-Programmen als Schnittstelle für den Zugriff auf Datenbanken (hier verwendet: HOBLink J-DRDA 3.1)



- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- **Allgemeines über LOBs und JDBC**
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



Large Objects (LOBs) – DB2

- **Welche Arten von LOBs gibt es bei der DB2?**
 - **BLOB (Binary Large Object)**
„Binary“ oder „raw“ Daten wie z.B. Videoclips, Bilddateien diverser Grafikformate oder Sounddateien.
Vorher CHARACTER FOR BIT DATA (maximale Länge 32KB)
 - **CLOB (Character Large Object)**
Dokumente bzw. Schriftstücke mit „Single-Byte“ Zeichen fester Länge aus dem Zeichensatz der DB.
Vorher LONG VARCHAR (maximale Länge 32KB)
 - **DBCLOB (Double-Byte Character Large Object)**
Große Blöcke, aus „Double-Byte“ Zeichen variabler Länge.
Vorher LONG VARGRAPH (maximale Länge 32KB)



Large Objects (LOBs) – DB2

- **Wie groß können LOBs maximal sein?**

Bis zu 2 Giga Byte (DB2)

- **Wo kommen LOBs zum Einsatz?**

- Beim Speichern von großen ASCII Texten
- Texten in länderspezifischen Zeichen
- Bilddateien
- Sounddateien
- Videoclips
- ...



Allgemeines über LOBs und JDBC

JDBC-Treiber

- JDBC Standard als einheitliche Schnittstelle zu DBMS
(Aktueller Standard: JDBC 3.0 API)
- Verbindungen zu Datenbanken aus Java-Applikationen, Applets, Servlets und Portlets
- Unterscheidung von 4 Typen an JDBC-Treibern
(„bridge-driver“, „native-driver“, DBMS spezifisches oder allgemeines Netzwerkprotokoll)
- 2-Tier oder 3-Tier Lösungen



JDBC-Treiber Typen

- Typ 1:
 - “bridge driver“ (JDBC-ODBC-bridge)
 - Nutzt andere Technologien (z.B. ODBC)
 - Nicht in Applets einsetzbar („native“ Code)
- Typ 2:
 - „native“ JDBC-Treiber
 - Nutzt DBMS API => benötigt daher DBMS Netzwerk Software
 - Nicht in Applets einsetzbar (wie Typ 1)



Allgemeines über LOBs und JDBC

JDBC-Treiber Typen

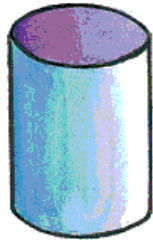
- Typ 3:
 - Nutzt Netzwerk Protokoll (Java-RMI, CORBA,...)
 - Umsetzung der Aufrufe in spezifische Datenbank Funktionen auf dem Datenbank-Server
 - Benötigt keine Client-Installation
- Typ 4:
 - DBMS spezifisches Netzwerkprotokoll (z.B. DRDA von IBM, SQL*Net von Oracle,...)
 - **Nachteil:** protokollspezifische Einschränkungen, beim Zugriff auf andere DBMS
 - **Vorteil:** optimaler Zugriff auf ausgelegtes DBMS



Allgemeines über LOBs und JDBC

Lokale Lösung (2-Tier)

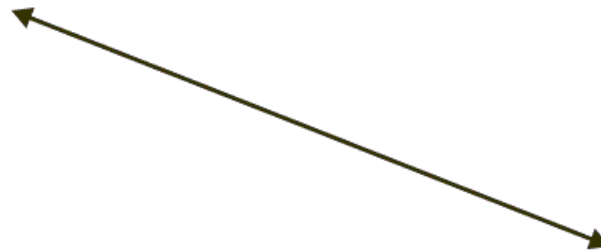
Datenbank



Client



Java-Applikation
HOBLink J-DRDA



Der JDBC Treiber wird lokal auf den Clients installiert, zur Verwendung mit ebenfalls lokal installierter Java-Applikation

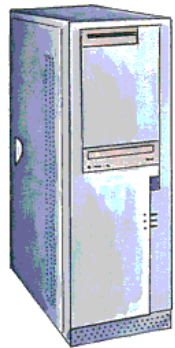


HOB
SOFTWARE

Allgemeines über LOBs und JDBC

Applet Lösung (3-Tier)

Web Server



HMTL

lädt

Applet

lädt

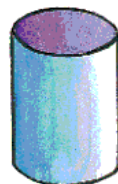
HOBLink J-DRDA

Client



Web Browser
mit JVM

Datenbank

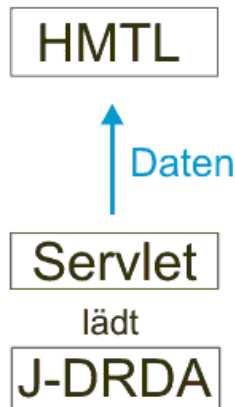
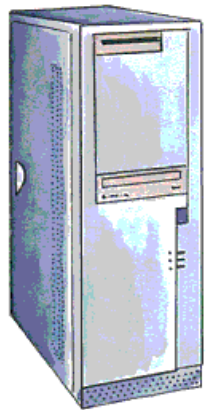


HOB
SOFTWARE

Allgemeines über LOBs und JDBC

Servlet / Portlet Lösung (3-Tier)

Web Server mit
Servlet Engine



Client



Browser

Datenbank



HOB
SOFTWARE

- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- **Schrittweise Entstehung der Anwendung**
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



SOFTWARE

Entwicklung der einzelnen Anwendungskomponenten

- **View (JSP)**

Darstellung der Daten im Browser

- **Model (Bean)**

Steuerung des Datenbankzugriffs und Bereitstellung der Daten

- **Control (Servlet)**

Steuernde Schnittstelle zwischen View und Model



SOFTWARE

Startseite der Applikation

- **HTML-Seite außerhalb der WAS Umgebung**
 - Verweis auf die URLs der Startseiten der Einzelanwendungen
 - Unternehmensanwendungen enthält z.B. nur ein Webmodul
- **HTML-Seite innerhalb der WAS Umgebung**
 - Unternehmensanwendung enthält mehrere Webmodule
 - Startseite verweist auf die einzelnen Webmodule
- **Portallösung mit Portal Server (z.B. IBM Websphere Portal Server)**
 - Servlets müssen mit geringem Zusatzaufwand zu Portlets erweitert werden
 - Alle Anwendungen sind über einheitliche Portaloberfläche zugänglich



SOFTWARE

Hauptseite(n)

- **Dargestellter Seiteninhalt ist über Attribute dynamisch veränderbar.**
Das Servlet setzt die Attributwerte anhand der Ergebnisse der Datenabfrage. Java Blöcke steuern den Aufbau der Seite.

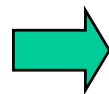
- **Getrennte Seiten für Benutzereingaben und Darstellung**

Eine Webseite ermöglicht Einstellungen und ruft Servlet auf, welches die Ergebnisse auf einer weiteren Webseite (JSP) darstellen läßt

- **Gemeinsame Seite für Selektion und Ergebnisdarstellung**

Auswertung der Benutzereinstellungen im JSP

(welche Elemente der Webseite sollen dargestellt werden?)



Aufwendigere Webseitengestaltung



Microsoft
Word-Dokument



Entwicklung der einzelnen Anwendungskomponenten

- View (JSP)

Darstellung der Daten im Browser

- **Model (Bean)**

**Steuerung des Datenbankzugriffs und
Bereitstellung der Daten**

- Control (Servlet)

Steuernde Schnittstelle zwischen View und Model



SOFTWARE

Stateless Session Bean

- **Home Interface**

Enthält die create()-Methode um eine Referenz des Remote-Objects zu erhalten

- **Remote Interface**

Enthält sämtliche Methoden Deklarationen des Beans

- **Bean Implementierung**



Microsoft
Word-Dokument

Hier sind die Methoden des Remote Interfaces implementiert.

Die query-Methoden führen die Datenbankzugriffe aus und get-Methoden stellen deren Ergebnisse den Servlets zur Verfügung



Entwicklung der einzelnen Anwendungskomponenten

- View (JSP)

Darstellung der Daten im Browser

- Model (Bean)

Steuerung des Datenbankzugriffs und Bereitstellung der Daten

- **Control (Servlet)**

Steuernde Schnittstelle zwischen View und Model



SOFTWARE

Control – Servlets als Schnittstelle zwischen View und Datenmodell

Die Servlets

InitServlet - Initialisierung



Holt verfügbare Automobil Modelle über Bean aus der Datenbank und setzt Default-Einstellungen für die Datendarstellung.

CarServlet - Steuerung der Benutzereingaben



Speichert Einstellungen, beauftragt Bean mit der Beschaffung der darzustellenden Daten und LOB Objekte für ausgewähltes Automobil



Control – Servlets als Schnittstelle zwischen View und Datenmodell

Alternative Ansätze – Nur ein Servlet

Zusammenfassung von InitServlet und CarServlet

- In der doGet() Methode müsste unterschieden werden, ob ein Request der Initialisierung oder der Datenabfrage dient

➡ Parameter zur Unterscheidung mit übergeben

- Wenige große Klassen ⇔ Viele kleine Klassen?



Control – Servlets als Schnittstelle zwischen View und Datenmodell

Alternative Ansätze – Datalink statt LOB

Nicht das Objekt selbst, sondern ein Link zu dem Objekt ist in der Datenbank gespeichert.

- **Vorteile:**
 - Kein „Zwischenspeichern“ der LOB-Daten
 - Keine Schreibrechte auf Server erforderlich
 - Keine großen Datenübertragungen zwischen DB und Server



Control – Servlets als Schnittstelle zwischen View und Datenmodell

■ Nachteile:

- Datentyp DATALINK wird z.B. von der DB2 nur unter AS/400 richtig unterstützt (BS und DB haben Kontrolle über das File / File kann z.B. nicht einfach gelöscht werden)
- DATALINK erst ab JDBC 3.0 in der API enthalten

Voraussetzung für JDBC 3.0: JVM 1.4.0 oder neuer

WAS 4.0 ermöglicht Kompilierung mit verschiedenen JDKs und nutzt intern die JVM 1.3.1.

Weitere Klassen können zur Laufzeit im Classpath ergänzt werden, aber ist es möglich die Anwendungen des WAS in einer anderen JVM ausführen zu lassen?



- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- **Schrittweise Entstehung der Anwendung**
 - Entwicklung der einzelnen Anwendungskomponenten
 - **Verteilen und Einrichten der Anwendung auf dem Application Server**
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLINK J-DRDA
- Fragen und Diskussion

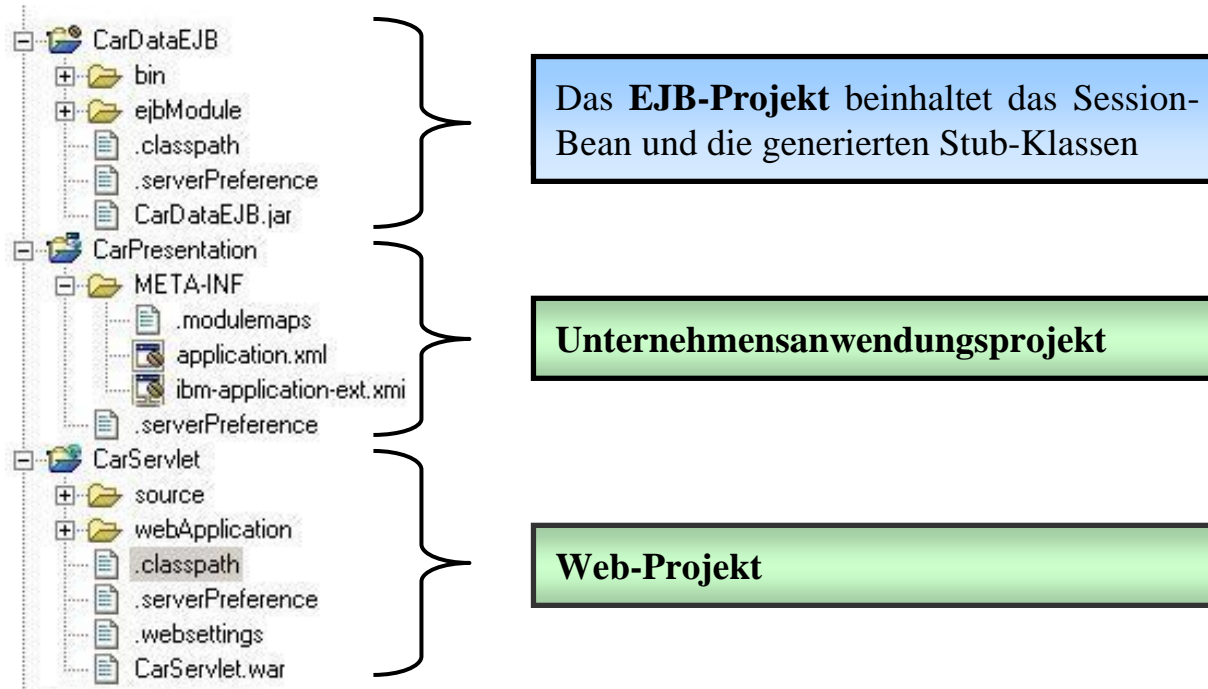


Verteilen und Einrichten der Anwendung

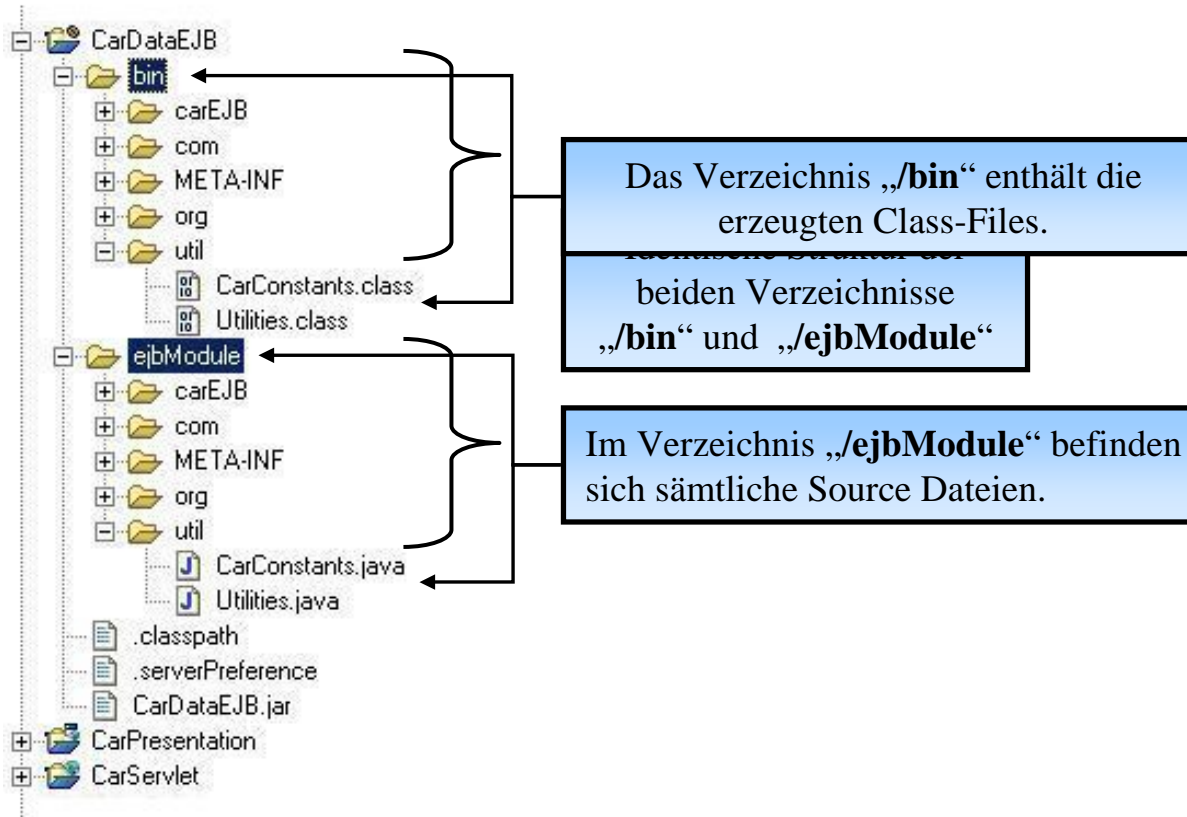
- **Anordnung der Einzelkomponenten und deren Verknüpfung zu einer Applikation**
- Einrichten einer Datenbank Verbindung im WAS
- Erzeugen der benötigten EAR-Datei und Installation der Anwendung am WAS



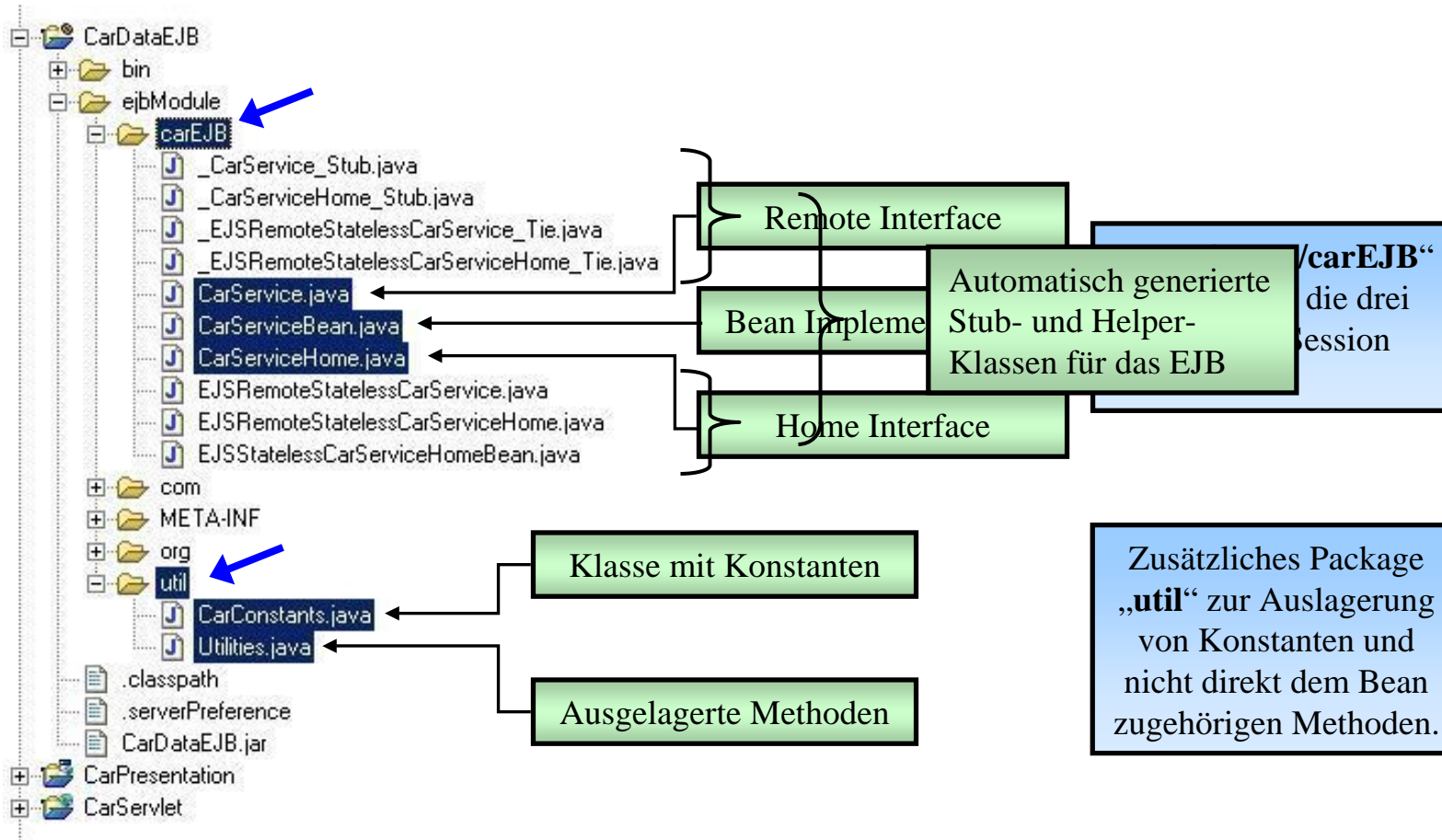
Übersicht der Anordnung



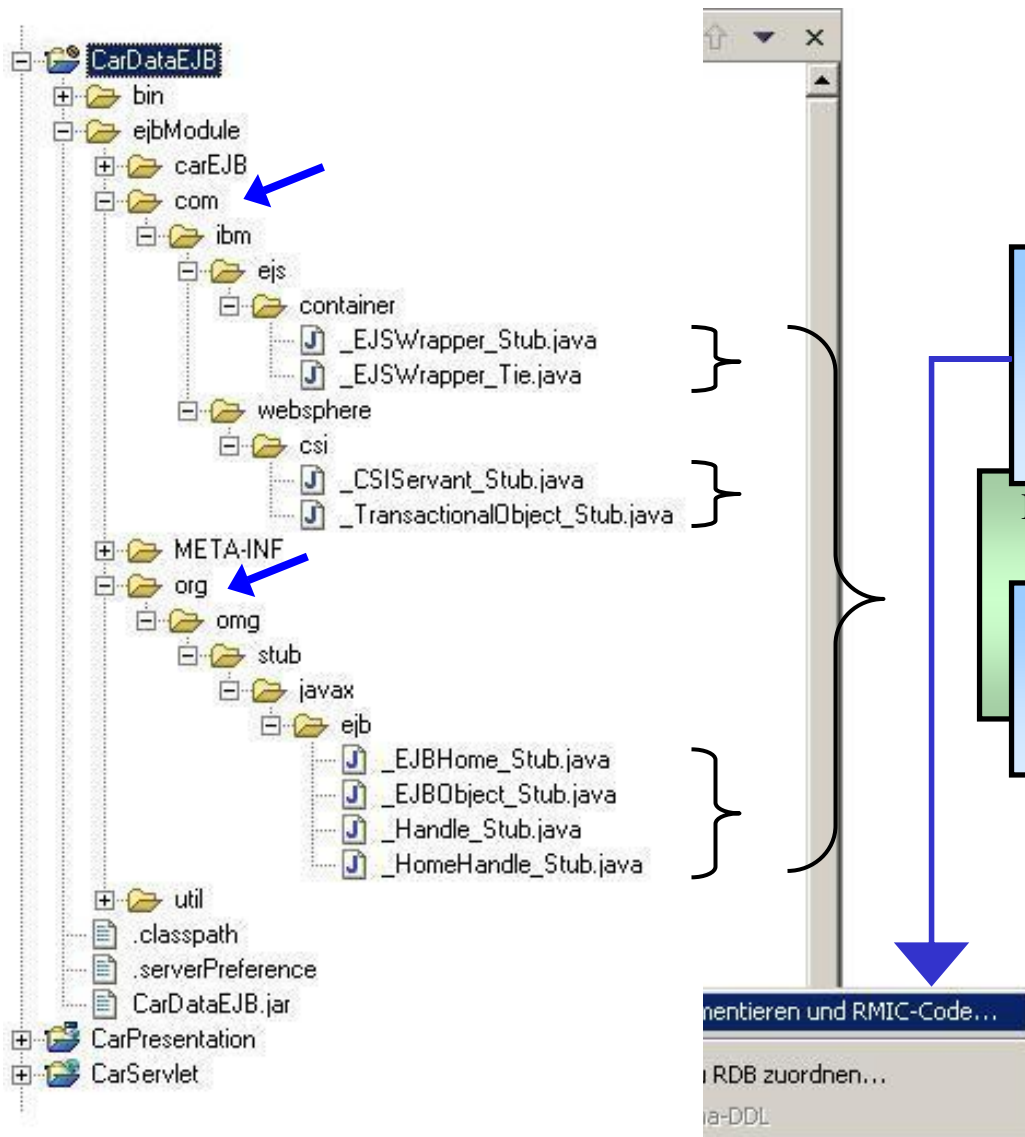
Das EJB-Projekt



Das EJB-Projekt



Das EJB-Projekt



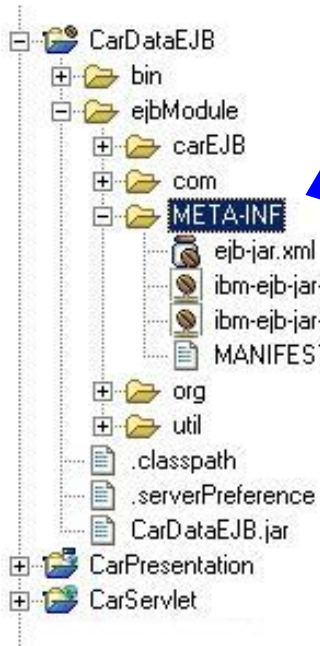
Zur Generierung der Stub-Klassen steht im Kontextmenu des EJB-Projektes eine spezielle Option zur Verfügung.

In den Verzeichnissen „./com“ und „./org“

Mit dieser Option werden sämtliche Stub- und Helper-Klassen in ihrer Verzeichnisstruktur erzeugt.



Das EJB-Projekt



ejb-jar.xml

Der Deploymentdeskriptor des EJB-Projekts enthält:

- EJB-Name
- Session- u. Transaction-Type
- Ressourcen-Referenzen
- Methodennamen

ibm-ejb-jar-bnd.xmi

Der Bindungsdiskriptor des EJB-Projekts:

Zur Ausführung auf dem WAS werden vom EJB-Projekt verwendeten Ressourcen JNDI-Namen zugewiesen.

ibm-ejb-jar-ext.xmi

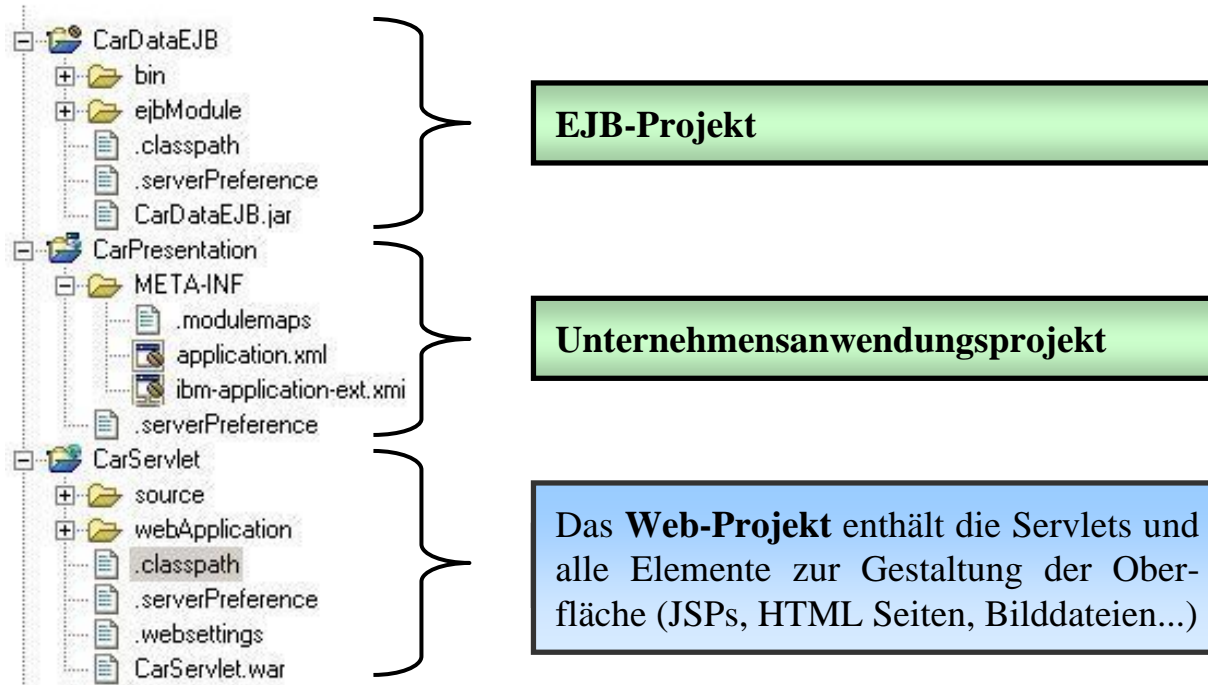
Der Erweiterungsdeskriptor des EJB-Projekts:

Erweitere Definitionen für ältere Systeme, die in der WAS Umgebung laufen sollen

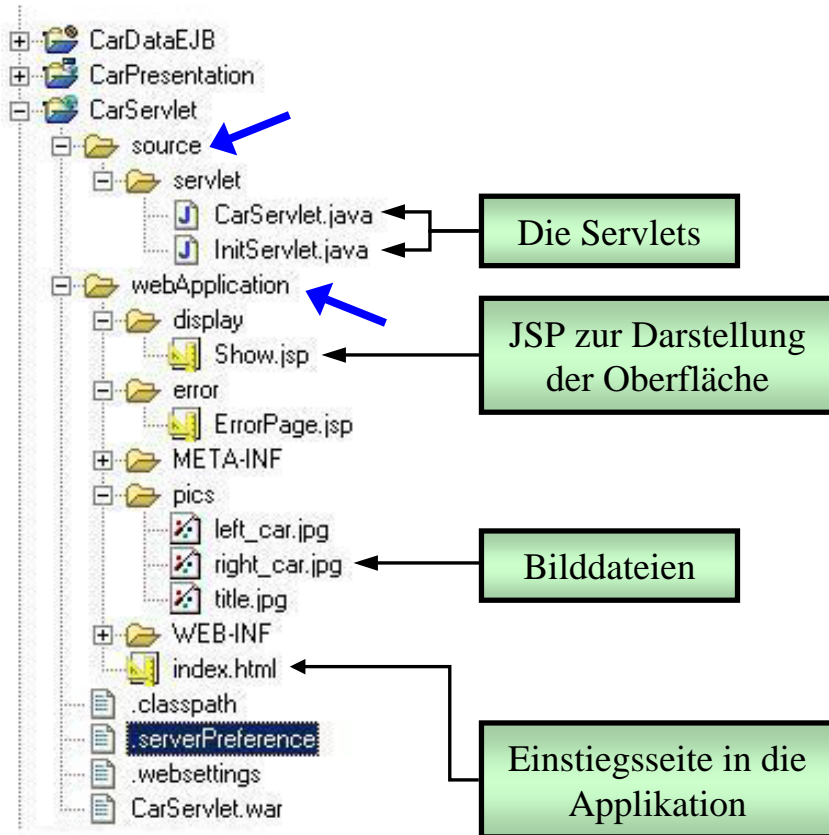
Das Verzeichnis „/META-INF“ enthält die XML-Beschreibung des Bean Moduls und dessen Referenzen zu den verwendeten Ressourcen



Übersicht der Anordnung



Das Web-Projekt



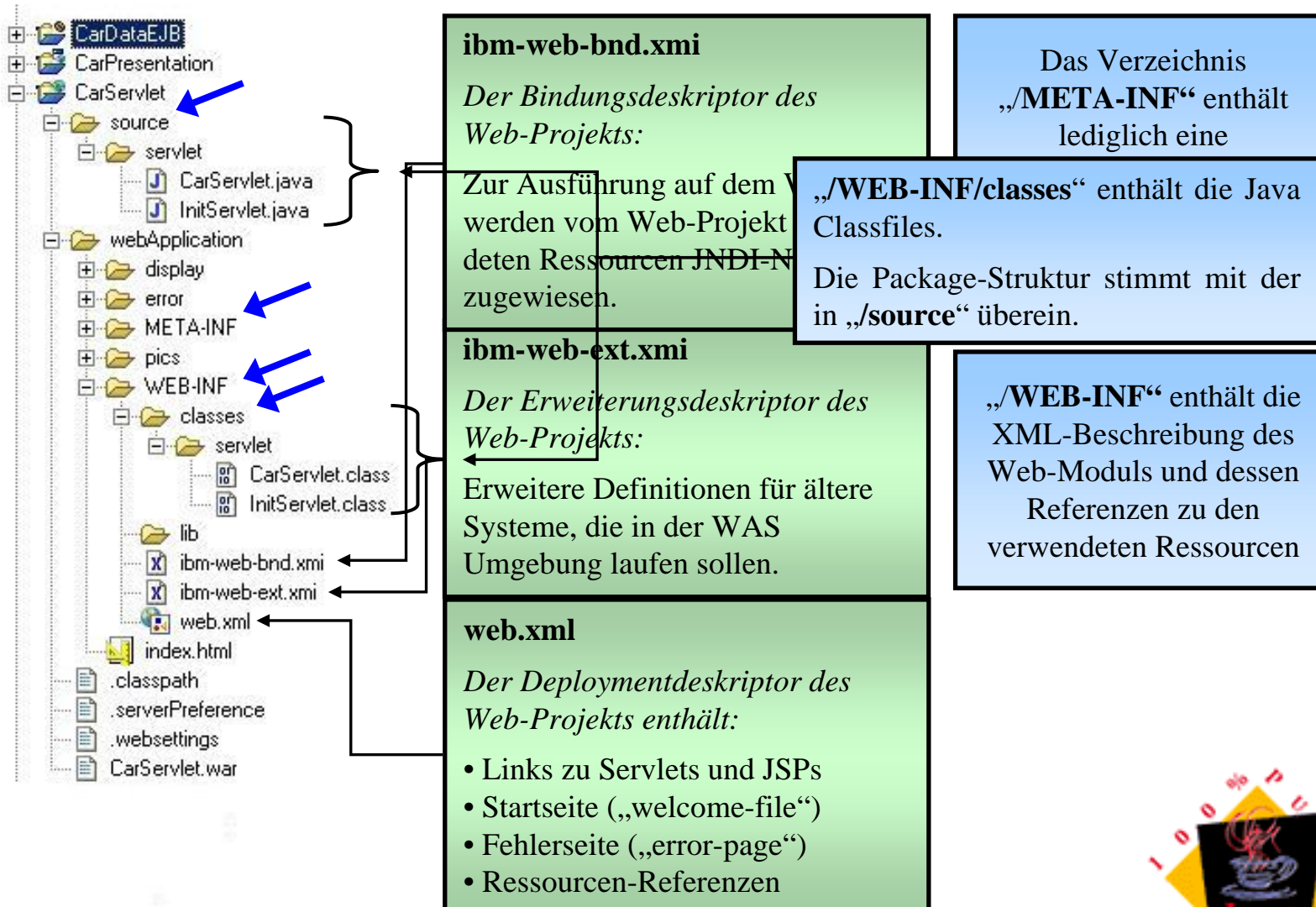
Im Verzeichnis „/source“ werden alle Java Sourcen des Web-Moduls in ihrer Package-Struktur angelegt.

Das Verzeichnis „/webApplication“ enthält sämtliche JSPs, HTML-Dateien, Bilddateien, Style-Sheets und andere Komponenten die der Darstellung der Oberfläche dienen.

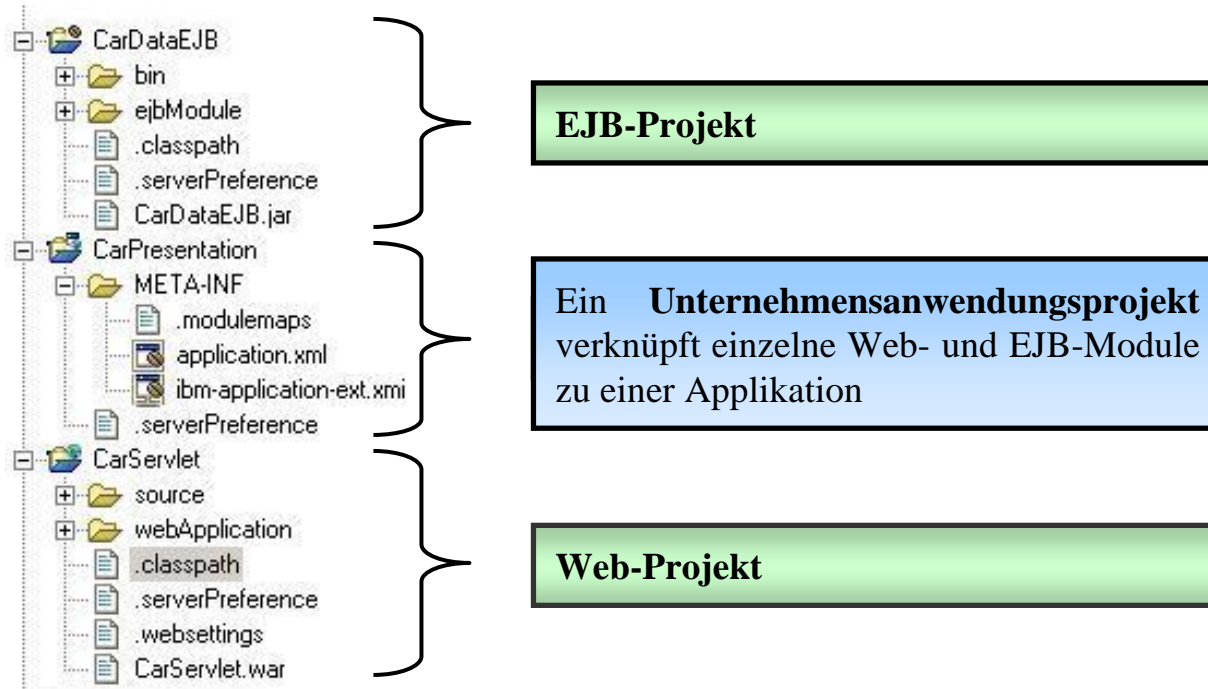
Die Verzeichnisse „/META-INF“ und „/WEB-INF“ sind mit ihrem Inhalt vorgegeben. Die weitere Verzeichnisstruktur und Dateinamen innerhalb von „/webApplication“ sind jedoch frei wählbar.



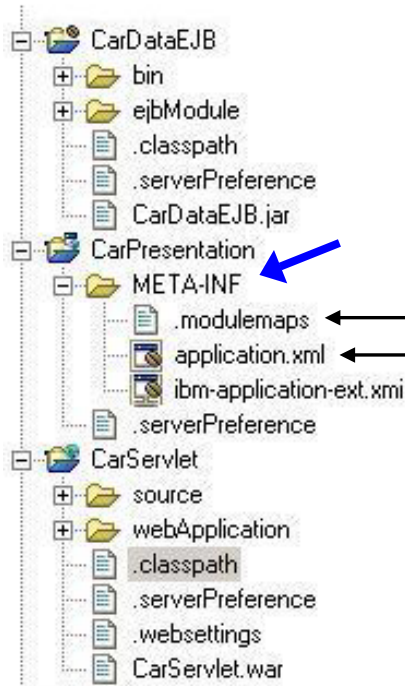
Das Web-Projekt



Übersicht der Anordnung



Die Unternehmensanwendung



.modulemaps

Enthält die Verknüpfungen der eingebundenen JAR-, WAR- und EJB-Module zu ihren jeweiligen Projekten

Die Dateien im Verzeichnis „/META-INF“ beschreiben aus welchen Einzelmodulen sich diese Applikation zusammensetzt.

application.xml

Der Deploymentdeskriptor der Unternehmensanwendung:
Enthält die Dateinamen der eingebundenen Module

ibm-application-ext.xmi

Der Erweiterungsdeskriptor der Unternehmensanwendung
Enthält die absoluten Pfade zu den Klassen und Dateien in den eingebundenen Modulen.

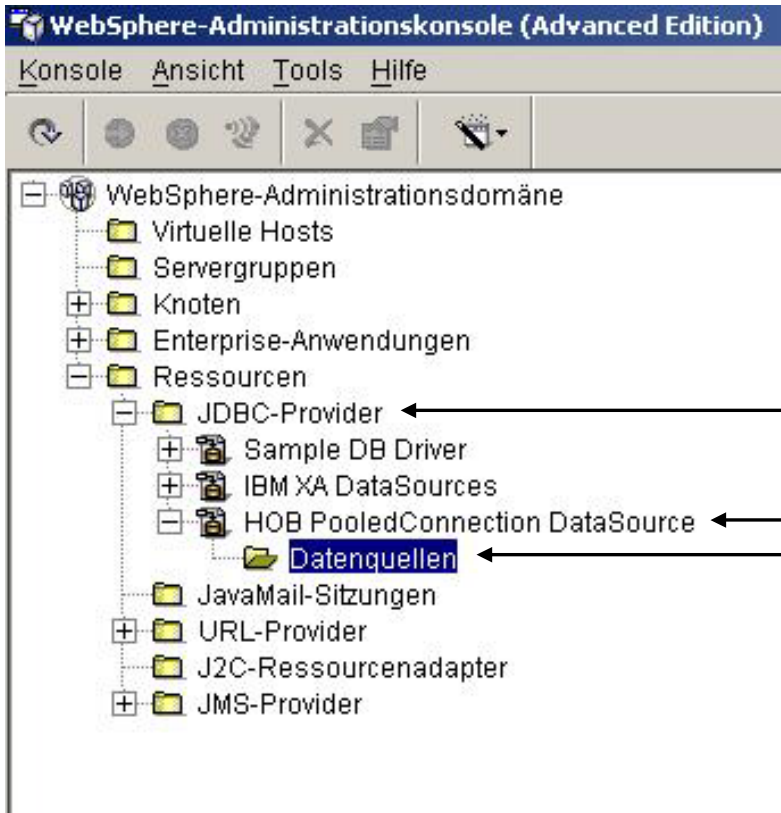


Verteilen und Einrichten der Anwendung

- Anordnung der Einzelkomponenten und deren Verknüpfung zu einer Applikation
- **Einrichten einer Datenbank Verbindung im WAS**
- Erzeugen der benötigten EAR-Datei und Installation der Anwendung am WAS



JDBC-Treiber einrichten



Erforderliche Angaben:

- Pfad zum JAR-File des Treibers
- Name und Package der DataSource Implementierungsklasse
- Knotenzuordnung

Anlegen eines neuen JDBC-Providers an der Administrationskonsole des WAS

Anschließend können neue Datenquellen, die von Applikationen auf diesem Knoten verwendet werden, angelegt werden.



DB-Verbindung anlegen

Datenquelle Merkmale

Allgemein | Verbindungs-Pooling

Der Hilfetext enthält Informationen zur Konfiguration von Datenquellen.

Name: *jdbc/NiceCarDB

JNDI-Name: jdbc/NiceCarDB

Beschreibung:

Datenbankname: SAMPLE

JDBC-Provider: *HOB PooledConnection DataSource

Benutzer-ID: transactions

Kennwort: *****

Kennwort bestätigen: *****

Benutzerdefinierte Merkmale

Name	Wert
serverName	172.22.80.125
portNumber	50000
user	transactions

Buttons: Hinzufügen, Löschen, OK, Abbrechen, Hilfe

Erforderliche Angaben beim Anlegen einer Datenquelle

1) Benennung der Datenquelle

2) JNDI-Name der Datenquelle (muß dem Eintrag in der Datei *ibm-ejb-jar-bnd.xmi* für diese Ressource entsprechen)

3) Beschreibung (optional)

4) Name der Datenbank zu der die Verbindung erstellt wird

5) Benutzer-ID und Passwort für den Datenbankzugriff

6) Individuell vom JDBC-Treiber abhängige Parameter für den Verbindungsaufbau (z.B. IP-Adresse und Port)

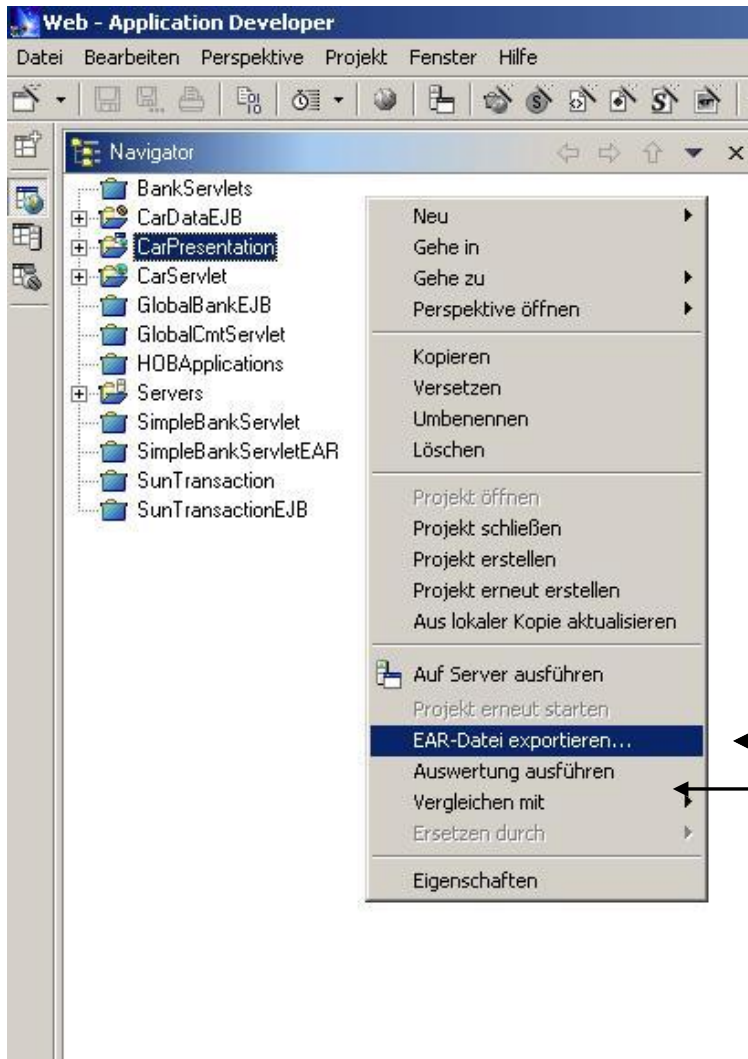


Verteilen und Einrichten der Anwendung

- Anordnung der Einzelkomponenten und deren Verknüpfung zu einer Applikation
- Einrichten einer Datenbank Verbindung im WAS
- **Erzeugen der benötigten EAR-Datei und Installation der Anwendung am WAS**



Letzte Schritte



Erzeugung der Einzelmodule im
WebSphere Application Developer

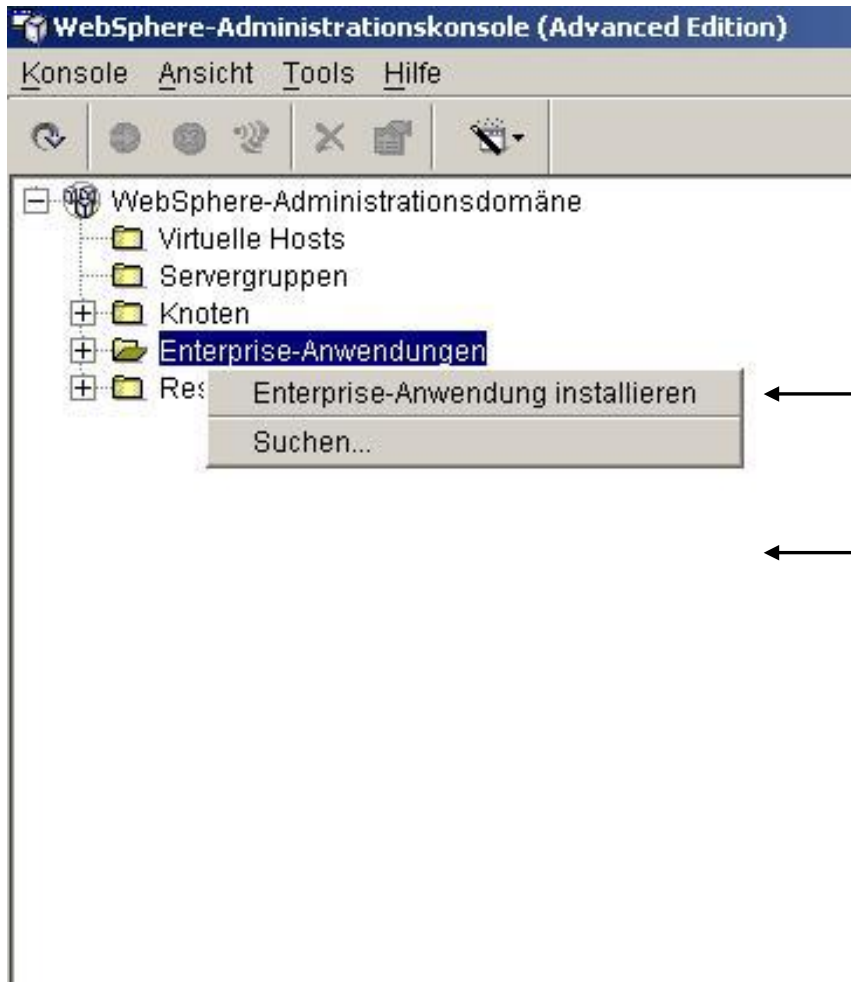
1) JAR-File des EJB-Moduls

2) WAR-File des Web-Moduls

3) EAR-File der gesamten
Unternehmensanwendung



Letzte Schritte



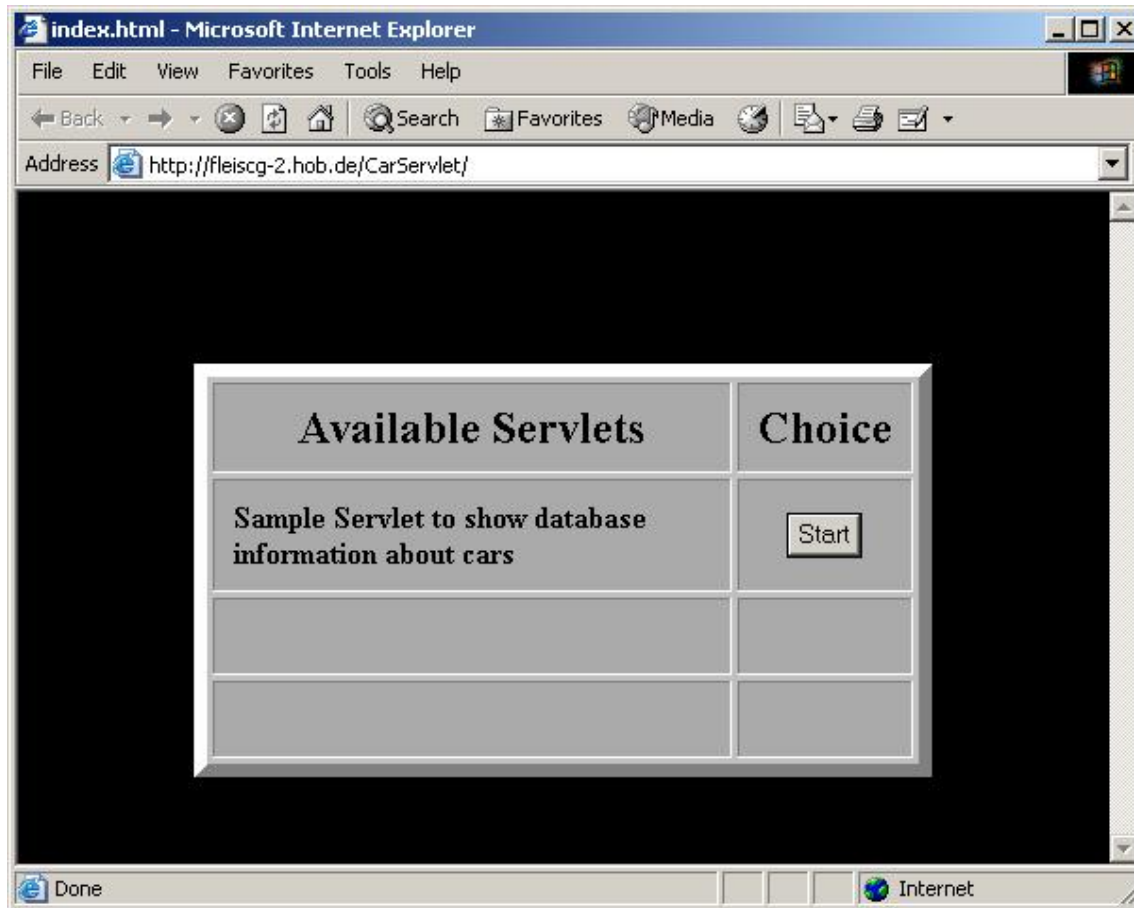
Aktivierung der Unternehmensanwendung im Websphere Application Server

1) EAR-File installieren

2) Unternehmenanwendung bereitstellen



Aufruf der Applikation



Browserfenster öffnen, über HTTP-Protokoll zum Application Server verbinden und dabei das Stammverzeichnis der Anwendung angeben.



- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- **Abschließende Präsentation der Beispielanwendung**
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- Fragen und Diskussion



- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- **Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA**
- Fragen und Diskussion



SOFTWARE

HOBLink J-DRDA

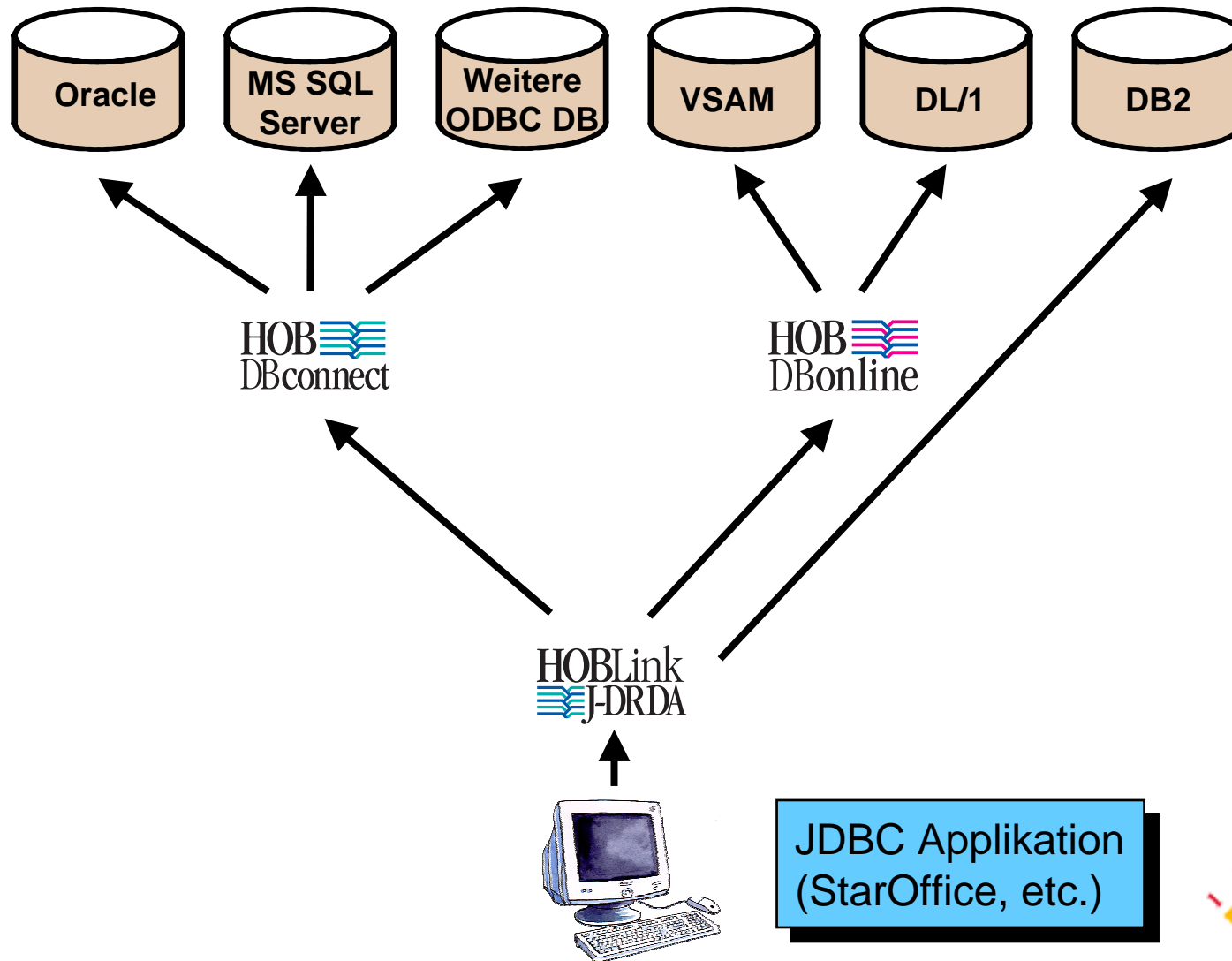
V.3.1

Die plattformunabhängige Lösung zum
Datenbankzugriff über JDBC

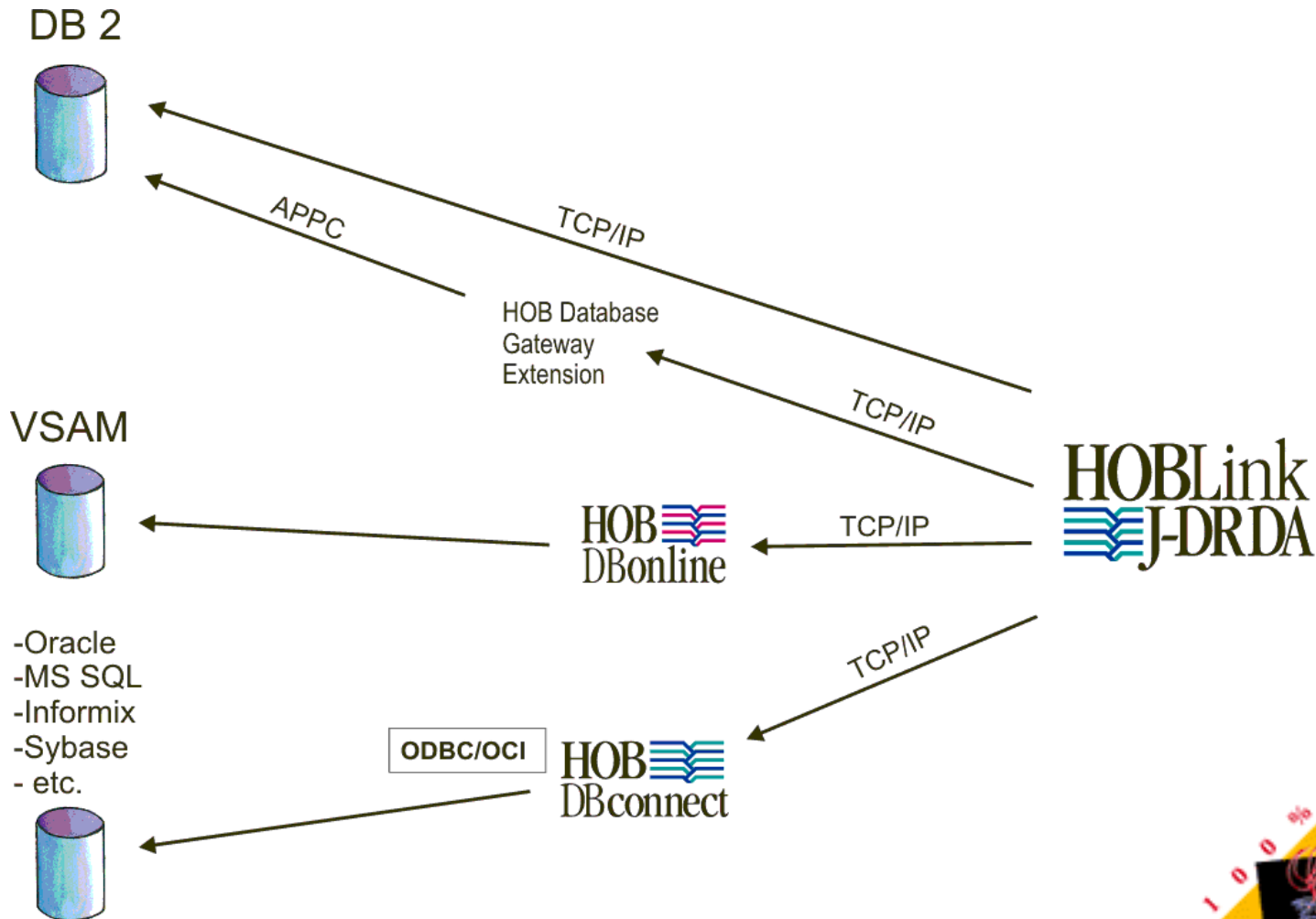


HOBS
SOFTWARE

Datenbank-Connectivity über JDBC



Zugriffsmöglichkeiten



Leistungsmerkmale HOBLink J-DRDA

- **Implementierung der JDBC 2.0 API**
 - Connection Pooling
 - LOB-Support (CLOB, BLOB)
 - Distributed Transactions (in Arbeit)

- **Implementierung der JDBC 3.0 API**
 - Savepoint Support
 - Multiple Open Resultsets
 - Cursor Holdability steuerbar



SOFTWARE

Leistungsmerkmale HOBLink J-DRDA

- **Höchste Leistung durch:**
 - Multi-threaded JDBC Treiber
 - Stored Procedures
 - Volle Transaktionskontrolle
(4 Isolationsstufen einstellbar)
- **Unterstützung aller Java-Client-Plattformen**
 - JDBC 2.0 setzt JVM 1.2 oder höher voraus
 - JDBC 3.0 Funktionen setzen JVM 1.4.0 oder höher voraus
(JDBC 3.0 Standard API erst ab JVM 1.4.0 enthalten)



SOFTWARE

Einsparungs- und Nutzenpotentiale

- **Zentrale Server-Installation und Administration**
=> keine Client-Installation erforderlich
- **“100 % Pure Java” zertifiziert**
- **Entwicklung von Datenbankanwendungen in Java für transparenten Zugriff auf DB2, VSAM, DL/1, Oracle, MS SQL Server...**
- **Datenbank-Zugriff auch über SNA-Protokoll**
(mit HOB Database Gateway Extension)



Einsparungs- und Nutzenpotentiale

- **Minimaler Ressourcenbedarf**
- **Kompakte Größe (ca. 220 KB) gerade bei Applets (nur ca. 140 KB) von Vorteil**
- **Sicherheit:**
 - digitale Signatur
 - SSL3 Verschlüsselung (Schlüssellängen bis zu 256 Bit) und Komprimierung mit HOBLink Secure (optional)



- Zielsetzung für die Anwendung
- Kurze Präsentation der Beispielanwendung
- Die Architektur
- Entwicklungs- und Laufzeitumgebung
- Allgemeines über LOBs und JDBC
- Schrittweise Entstehung der Anwendung
 - Entwicklung der einzelnen Anwendungskomponenten
 - Verteilen und Einrichten der Anwendung auf dem Application Server
- Abschließende Präsentation der Beispielanwendung
- Einige Eckdaten zum JDBC-Treiber HOBLink J-DRDA
- **Fragen und Diskussion**



Noch Fragen?



HOB
SOFTWARE