J2EE

Applicatie met Enterprise Java Beans

Middleware specialisatiethema
Rob Juurlink IID7
2003 / 2004

VOORWOORD

In deze middleware opdracht wordt gebruik gemaakt van J2EE Enterprise Java Beans. De opdracht is onderdeel van een groter middleware project waarbinnen meerdere middleware technologieën aan bod komen.

Voor het ontwerpen van de classediagrammen is er gebruik gemaakt van het pakket Poseiden van Gentleware. Het is een UML ontwikkelgereedschap.

Voor de ontwikkeling van de software is gebruik gemaakt van de Sun J2EE versie 1.3 referentie applicatieserver.

Om efficiënt Java-applicaties te ontwikkelen, is een IDE (Integrated Development Environment) gebruikt. De gebruikte ontwikkelomgeving is NetBeans versie 3.5.1.

De ontwikkelingen van het Middleware project zijn ook te volgen via een website. Het adres van de website is: http://rob.juurlink.org.

INHOUDSOPGAVE

1. INLEIDING

De opdracht is het ontwerpen van een elektronisch warenhuis.

Het gebruik van Java Enterprise Beans wordt als doel gesteld. Er is een database aanwezig. De data wordt via zogenaamde Container Managed Beans in de database opgeslagen. Het opslaan van de data wordt, zoals de naam al zegt, afgehandeld door de container.

Het document begint met een uitleg over de gebruikte technologie, Java Enterprise Beans.

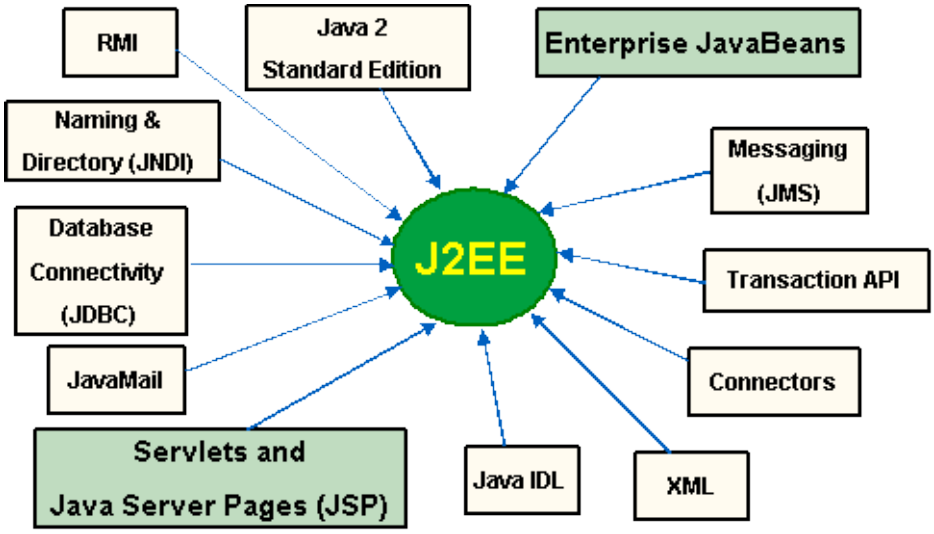
Als de lezer een beeld heeft van de techniek, volgt de gedetaileerde opdrachtomschrijving. De opdracht wordt daarna uitgewerkt. Dit levert een aantal classes op.

Na de het ontwerp volgt een uitleg voor de gemaakte keuzes en een beschrijving voor het installeren en de werking.

2. J2EE APPLICATIESERVER

De Java Platform Enterprise Edition, kortweg J2EE, biedt een op componenten gebaseerde benadering van ontwerp, ontwikkeling, en deployment.

J2EE (versie 1.3) bestaat uit 13 verschillende technologieën, waaronder Java Server Pages (JSP), Servlets, Enterprise Java Beans, XML, Java Mail enz. Zie figuur 1 voor een schematisch overzicht.



Figuur 1, een schematisch overzicht van de verschillende technologieën waaruit de J2EE applicatieserver bestaat.

De applicatieserver zorgt voor de volgende zaken:

- Persistence
- Transactions
- Security
- Error handling
- Component Framework for Business Logic
- Scalability
- Portability
- Manageability

2.1. ENTERPRISE JAVABEANS

De Enterprise Java Bean Container is een interface tussen de Enterprise Java Bean en de buitenwereld. Een cliënt maakt nooit rechtstreeks verbinding met de Enterprise Java Bean, alle communicatie met de bean gaat via door de container gegenereerde methoden.

Een Enterprise Java Bean is een component die zorgt voor het inkapselen en beheren van business logica. Een Java Bean bevindt zich in een Bean Container op de applicatieserver. Deze container handelt alle low-level zaken van de Bean af, daardoor wordt de programmeur afgeschermd van allerlei moeilijke en programmeer intensieve taken, zoals transactie management, security en database connecties.

De Java Bean Container stelt een framework aan de Bean bechikbaar voor gedistribueerde n-tier middleware. Een Enterprise Java Bean wordt ook wel een Server Bean genoemd.

In figuur 2 (op bladzijde 5) is het schema van een Enterprise Java Bean afgebeeld waarin zich meerdere containers bevinden. In één van de containers bevindt zich een Session bean, waarvan de container een EJBHome interface en een EJBObject heeft gecreeerd ten behoeve van de communicatie naar de buitenwereld.

2.1.1. Session Bean

Een session Bean is een bean waarvan de levensduur gelijk is aan die van de cliënt die de bean gebruikt. Van Session Bean's zijn twee soorten, statefull- en stateless sessionbeans.

Een Stateful Session Bean kan informatie bewaren tussen twee methode aanroepen.

Bij een Stateless Session Bean is geen opslag van de data morgelijk.

2.1.2. Entity Bean

Een Entity Bean representeert data in een database en het scheidt de business code van de persistente opslag systemen. Kort gezegd bevat een entity bean code voor validatie en garandeert het de integriteit.

Er zijn twee typen Entity Beans, Bean Managed Persistence (afgekort BMP) en Container Managed Persistence (afgekort CMP).

Elke Entity Bean heeft een primary key die de bean identificeert. Een primary key is samengesteld dmv een primary key class waarin de attributen staan waaruit de key bestaat. De attributen in deze class moeten serializable zijn en de class moet een implementatie van methoden hashCode() en equals() bevatten.

De key moet uniek zijn binnen verzameling van beans van het betreffende type.

Voor Container Managed Beans gelden nog een paar extra eisen: De class moet default constructor hebben en de de class heeft public attributen die overeenkomen met die uit de bean.

Het is een ongeschreven regel dat de entity bean geen business logica bevat dit in tegenstelling tot de versie 1.1 specificatie van EJB. Vaak werd toen de business logica in dezelfde bean gezet ivm performande issues. Er was voor de communicate alleen RMI beschikbaar en dat performde slecht. Het inbouwen van de code in dezelfde entity bean zorgde er voor dat extra communicatie werd beperkt.

Clients mogen niet rechstreeks met entity beans communiceren, maar doen dit via een session bean.

2.1.3. Bean Managed Persistence (BMP)

De relatie met de database wordt beheerd door de programmeur door code in de bean zelf. De implementatieclass bevat JDBC en SQL code om data op te halen uit de dta.

De class wordt concreet gedefinieerd met public instantie variabelen voor alle BMP velden.

2.1.4. Container Managed Persistence (CMP)

De relatie met de database wordt beheerd door de container waarin de bean leeft. De container zorgt voor de transfer van data van de database naar de instance variabelen van de bean.

Het meest handige is om CMP beans te gebruiken, maar dat is niet in alle gevallen mogelijk. De voordelen van een CMP bean.

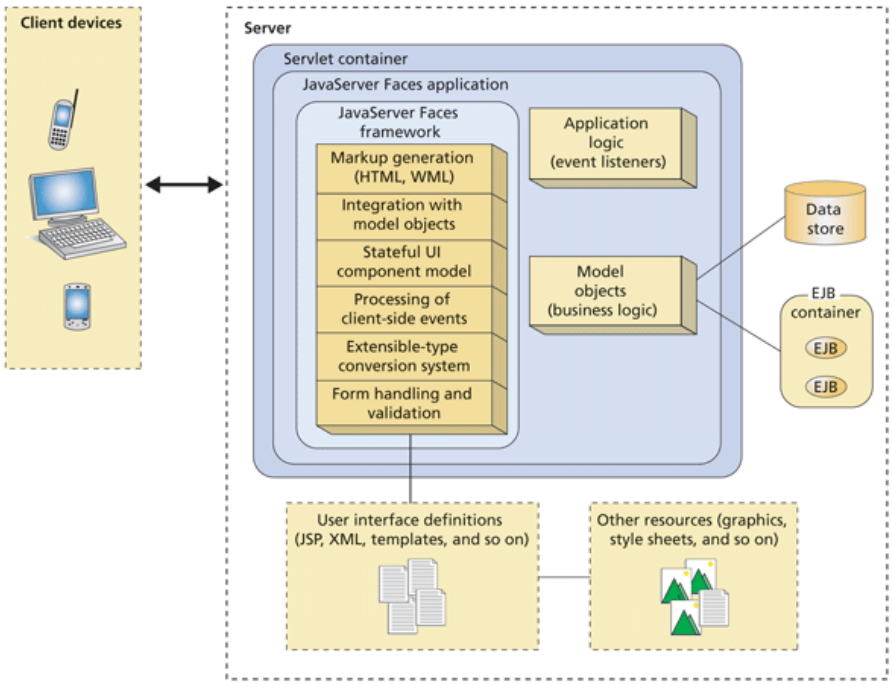
- Geen eigen JDBC- en SQL-code.
- Container kan gebruik maken van optimalisatie bij database access en caching voor verbetering performance.

De class voor Container Managed Persistence wordt abstract gedefinieerd met abstracte accessoren voor alle CMP velden. De container genereert m.b.v. informatie uit deployment descriptor een concrete class implementatie.Relaties tussen beans (1:1, 1:n, n:m) worden vastgelegd in de deployment descriptor en in CMR velden in de beans. De container zorgt voor handhaving van de referentiele integriteit, toevoeging aan en verwijdering van beans uit een relatie en navigatie door relaties m.b.v. gegenereerde SQL statements.

De programmeertechnische details die betrekking hebben op CMP entity beans zijn beschreven in een appart hoofdstuk op bladzijde 6.

2.1.5. Container Managed Transaction (CMT)

In een CMT bean handelt de bean container de transacties af. Hoe dit afgehandeld moet worden is vastgelegd in deployment descriptors (met de variable genaamd “trans-attribute”).



Figuur 2, een Enterprise Java Bean Server waarin zich meerdere EJB containers bevinden. In één van de containers bevindt zich een sessie bean waarvan de container een EJBHome interface en een EJBObject heeft gecreeerd voor de communicatie.

2.1.6. CMP entity beans

Beschrijving details bestandsformaten en deployment discriptors.

In een container managed entity bean vanaf EJB specificatie versie 2.0, worden de te gebruiken velden gedeclareerd door een set van een abstracte getter en een abstracte setter methode. Dit is op bijna dezelfde wijze als een JavaBean, behalve dat de methoden hier geen implementatie bevatten.

Een voorbeeld van de local home, local en bean implementie waarin een aantal velden worden gedeclareerd:

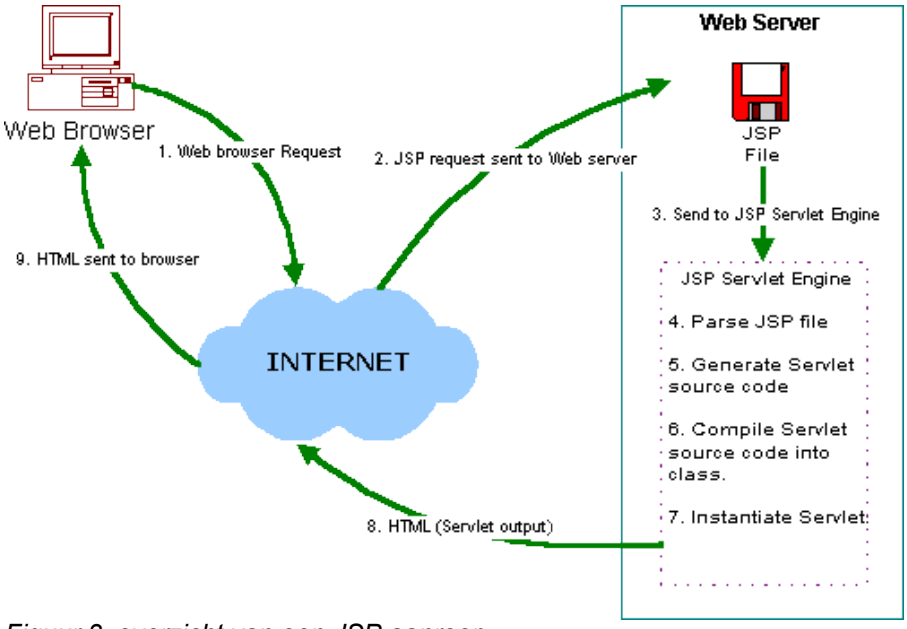
```
// Gangster Local Home Interface
public interface GangsterHome extends EJBLocalHome {
Gangster create(Integer id, String name, String nickName)
throws CreateException;
Gangster findByPrimaryKey(Integer id) throws
FinderException;
}

// Entity Local Home Interface
// Gangster Local Interface
public interface Gangster extends EJBLocalObject {
Integer getGangsterId();
String getName();
String getNickName();
void setNickName(String nickName);
}

// Entity Local Interface
// Gangster Implementation Class
public abstract class GangsterBean implements EntityBean {
private EntityContext ctx;
private Category log = Category.getInstance(getClass());
public Integer ejbCreate(Integer id, String name, String
nickName)
throws CreateException {
```

2.2. JAVA SERVER PAGES

JavaServer Pages, kortweg JSP, is een technologie die is gebaseerd op Java. JSP wordt gebruikt om dynamische, door de server gecreëerde website te beschrijven. JSP bestanden zien er uit als HTML bestanden met daarin speciale tags die Java code kunnen bevatten.

JSP is als laag bovenop Servlet technology gebouwd. Feitelijk is een JSP een HTML pagina met extra tags ingebouwd. Als extensie voor een JSP bestand wordt doorgaans `*.jsp` gebruikt ipv `*.html`.

De eerste keer dat het bestand wordt aangeroepen door de webserver, wordt het JSP bestand geparst(omgezet) naar een Java Servlet bronbestand. Dit bronbestand wordt daarna gecompileerd, dit gebeurt alleen de eerste keer dat het bestand wordt aangeroepen. De volgende keren wordt de gecompileerde code gebruikt. Dit is de reden dat de eerste keer dat het bestand aangeroepen wordt, dit merkbaar langzamer gaat.



Figuur 3, overzicht van een JSP aanroep.

In figuur 3 hierboven staan de stappen van het aanroepen van een JSP pagina afgebeeld.

1. De webbrowser op de desktopmachine doet een aanroep. (een adres ingetypt)
2. De aanroep komt binnen bij de webserver.
3. De webserver stuurt de aanroep door naar het deel dat JSP afhandelt.
4. , 5., 6., 7. Bij de eerste aanroep wordt de JSP pagina geparsd, gecompileerd en uitgevoerd. Bij de volgende aanroepen wordt de code meteen uitgevoerd.
8. De uitvoer van de webserver wordt verzonden.

9. Het antwoord komt terug bij de desktopmachine.

In JSP bevinden zich vier soorten hoofdtags. In onderstaande tabel zijn de hoofdtags inclusief een beschrijving wat ze doen afgedrukt.

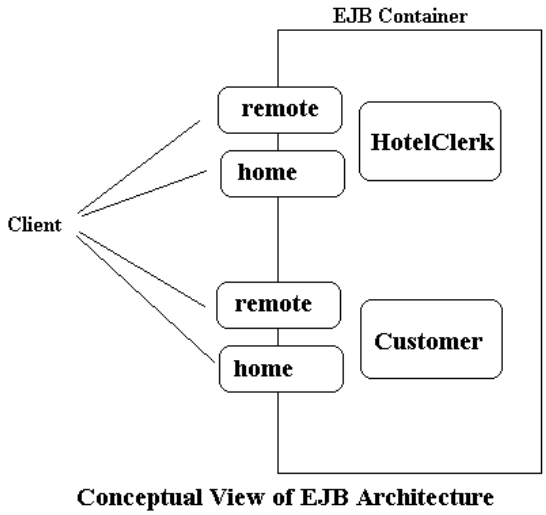
Naam	***Schrijfwijze***	***Omschrijving***
Declaration tags	<%! %>	Met deze tags worden variabelen gedeclareerd.
Expression tags	<%= %>	Tags voor korte java expressies. Eigenlijk is deze tag kort voor “out.println()”. De expressie wordt niet afgesloten met een puntkomma.
Directive tags	<%@ %>	Voor het doorgeven van speciale informatie. Include - files to be included. Tag library - tag library to be used in this page.
Scriptlet tag	<% %>	Hier komt de uitvoerbare Java code te staan. De code wordt Scriptlet genoemd.

Tabel 1, de vier standaard soorten tags in JSP.

2.3. PROGRAMMEREN VAN EEN EJB

Een Enterprise Java Bean (EJB) bestaat dus altijd uit de volgende code:

- Remote interface
- Home interface
- Enterprise Bean class
- Deployment descriptor



Figuur 4, een client maakt nooit rechtstreeks contact met de bean (in het voorbeeld de HotelClerk en Customer bean. Een client zoekt de bean op via de naming service waarin de bean staat geregistreerd. Via een de home object wordt een nieuwe instantie van de bean gecreëerd. De bean wordt daarna benaderd via de remote interface.

2.3.1. De Remote interface

Een remote interface definieert de business methoden van een Enterprise Java Bean die een (remote) client mag aanroepen. De code voor de remote interface ziet er als volgt uit:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface BeanNaam extends EJBObject {
    public Object businessMethode(int inputWaarde)
        throws RemoteException;
}
```

2.3.2. Naamgeving Enterprise Beans

Because enterprise beans are composed of multiple parts, it's useful to follow a naming convention for your applications. Table 3-2 summarizes the conventions for the example beans of this tutorial.

Item	*Syntax*	*Example*
Enterprise bean name	`<name>EJB`	`AccountEJB`
EJB JAR display name	`<name>JAR`	`AccountJAR`
Enterprise bean class	`<name>Bean`	`AccountBean`
Home interface	`<name>Home`	`AccountHome`
Remote interface	`<name>`	`Account`
Local home interface	`Local<name>Home`	`LocalAccountHome`
Local interface	`Local<name>`	`LocalAccount`

2.3.3. De Home interface

De home interface definieert de methoden die een client de mogelijkheid geven om een Enterprise Bean te creëren, te vinden of te verwijderen. Onderstaande interface bevat alleen een `create` methode welke een object van het type `remote interface` terug geeft.

```
import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;
public interface BeanNaamHome extends EJBHome {
    Converter create() throws RemoteException, CreateException;
}
```

2.3.4. De Enterprise Bean Class

De Enterprise Bean Class implementeert de business methoden die de remote interface gedefinieerd heeft. De broncode van deze class ziet er als volgt uit:

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
import java.math.*;

public class BeanNaamBean implements SessionBean {

    public Object businessMethode(int inputWaarde) {
```

```
            return new Integer(inputWaarde);
    }

    public ConverterBean() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
}
```

2.3.5. Deployment descriptor

De deplyment descriptor is een XML-bestand dan informatie over de bean bevat, zoals persistentie en transactietype enz. Bij gebruik van de Sun JDKEE1.3 creëert de deploytool de descriptor dmv een wizard.

To develop an enterprise bean, you must provide the following files:

2.3.6. JNDI

JNDI staat voor Java Naming en Directory Implementation.

Client

Als de JNDI API in dezelfde Virtual Machine wordt gebruikt als waarin de applicatieserver draait, dan is geen extra configuratie vereist voordat een `initialContext` object gecreeerd kan worden. Een `initialContext` object is een referentie naar de JNDI server. Via dit object kan de referentie naar remote Enterprise Java Beans worden opgevraagd.

Om met een client verbinding te kunnen maken met de JNDI server van de JBoss applicatie server bijvoorbeeld, moeten naast de instellingen ook de beide biblotheken jnp-client.jar en jnet.jar in het classpath staan. Vanaf Java JDK 1.4 zitten deze bibliotheken staandaard in het classpath.

3. WEBWINKEL MBV EJB'S

3.1. OPDRACHTOMSCHRIJVING

Kies gemotiveerd een J2EE omgeving en installeer die. (Je kunt gebruik maken van de Sun reference implementatie j2sdkee-1.3.1 of j2sdk-1.4-beta, maar realiseer je dat er problemen kunnen voorkomen bij geavanceerde applicaties.)

Ontwerp en implementeer voor ons [S]Aktiehuis warenhuis 3 CMP (Container Managed Persistency) beans: Product, Klant en Bestelling. Maak gebruik van N:M relaties en CMR (Container Managed Relationship) fields.

Cliënten kunnen zich opgeven als klant, productgegevens opvragen en bestellingen doen. Interactie van cliënten met de entity beans verloopt via één stateless session bean.

De cliëntapplicatie mag een eenvoudige Java applicatie, maar ook een web applicatie met servlets en JSP’s zijn.

3.2. HET ONTWERP

De gebruikte technologie stelt een aantal eisen aan het ontwerp.

De businesscode wordt geïmplementeerd in Enterprise Java Beans. Om precies te zijn wordt de data bewaard door de CMP EJB's en een stateless sessionbean bevat de business code die de CMP entity beans aanspreekt.

De client is een webapplicatie, te bereiken via een browser. De webapplicatie comminiceert alleen door middel van de stateless sessionbean. Een referentie naar deze bean wordt via verkregen via JNDI.

De Enterprise Java Beans worden gedemonstreerd via een webapplicatie. Dit is de meest eenvoudige oplossing om de applicatie te demonstreren. Er hoeven geen verbindingen via een firewall opgezet te worden.

3.2.1. Applicatieserver

De gebruikte applicatieserver is de J2EE1.3 referentie implementatie van Sun. Tijdens de start van deze opdracht was dit de laatste stabiele versie van de referentie implementatie.

De keuze is op deze applicatieserver gevallen omdat er een uitgebreide handleiding, inclusief voorbeeld programma's meegeleverd wordt. Dat deze aplicatieserver niet geschikt is voor het echte werk is in deze opdracht niet van belang.

Naast de Sun referentieimplemenatie, is ook de applicatieserver van JBoss getest. Dit schijnt een goede applicatieserver te zijn die gratis beschikbaar is. Daarnaast is deze applicatieserver open-source. Een groot nadeel van JBoss is dat er nauwelijks documentatie aanwezig is. Als beginnende gebruiker van J2EE is het dan erg moeilijk een begin te krijgen.

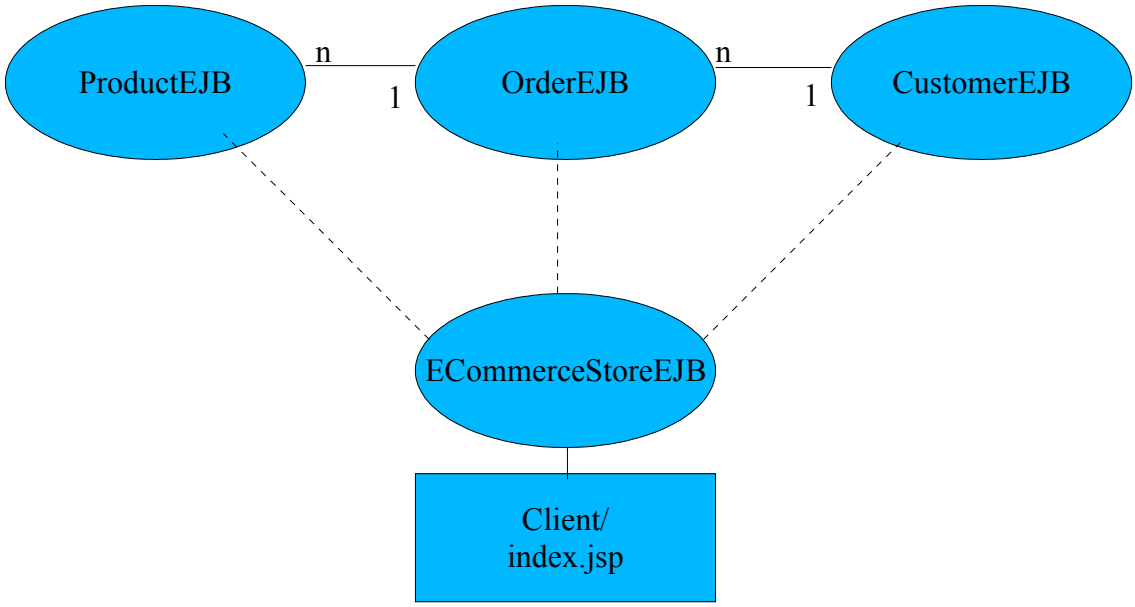
Doordat er gebruik gemaakt is gemaakt van de J2EE1.3 referentie implementatie zou de applicatie eenvoudig, met alleen wat aanpassingen in de deploymentdescriptor ook op andere J2EE1.3 applicatieservers moeten kunnen draaien. Er was echter binnen deze opdracht niet voldoende tijd om dit uit te testen.

3.2.2. Schema

Er zijn drie verschillende entity beans. Order, Product en Customer. Een order kan 1 of meerdere producten bevatten (een betere naam was misschien orderItems). Een klant kan een order opgeven. Een order kan maar aan 1 klant gekoppeld worden. Een klant kan wel meerdere orders opgeven.

De entity beans zijn niet rechtstreeks te benaderen, maar alleen via de business code die in de sessie bean genaamd ECommerceStore aanwezig is. De zichtbare interface voor de klant is een webinterface. De client kan alleen met de ECommerceStoreEJB communiceren.

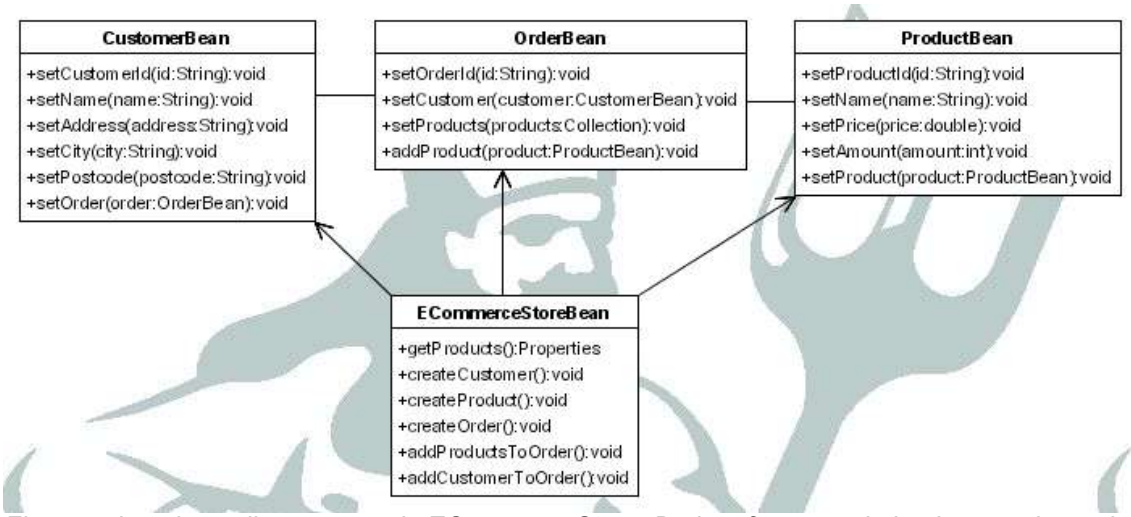
Het bovenstaande verhaal is op de volgende manier uitgewerkt.


ProductEJB
n
1
OrderEJB
n
1
CustomerEJB
ECommerceStoreEJB
Client/
index.jsp

3.2.3. Classediagram

In het onderstaande classediagram zijn alleen de interfaces afgebeeld. Dit geeft een duidelijke voorstelling van de verschillende classes en hun functie.

De relaties tussen de beans zijn vastgelegd in de deployment descriptor.



Figuur 5, het classediagram van de ECommerceStore. De interface van de businesscode en de Enterprise Beans zijn afgebeeld.

3.2.4. (Session) Enterprise Bean Code

De enterprise beans bestaan uit de volgende classes.

- Remote interface
- Home interface
- Enterprise bean class

Remote Interface

De remote interface definieert de methoden die een client mag aanroepen. De uitvoerbare code staat in de Enterprise Bean class. De broncode voor de stateless session bean ECommerceStoreBean ziet er als volgt uit:

```
/*
 * ECommerceStore.java
 */

import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.util.*;

/**
 *
 * interface van de ECommerce businesscode.
 */
public interface ECommerceStore extends EJBObject {
```

```
    public Properties getProducts() throws RemoteException;
    public void createCustomer(String customerId, String name, String address,
        String city, String postcode);
    public void createProduct(String productId, String name, double price,
        int amount);
    public void createOrder(String orderId);
    public void addProductsToOrder(Collection productIds, String orderId);
    public void addCustomerToOrder(String customerId, String orderId);
}
```

De Home Interface

In de home interface worden de methoden gedefinieerd die een client de mogelijkheid geven om een enterprise bean te creeren, zoeken of verwijderen. De code bevat een methode create(). De terugkeerwaarde is een object van het type remote interface. De code ziet er als volgt uit:

```
/*
 * ECommerceStoreHome.java
 */

import java.io.Serializable;
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;

/**
 * Home interface ECommerceStore.
 */
public interface ECommerceStoreHome extends EJBHome {

    ECommerceStore create() throws RemoteException, CreateException;
}
```

De Enterprise Bean Class

De Bean Class bevat de methoden die in de interface gedefinieerd worden. De class ziet er als volgt uit:

```
/*
 * ECommerceStoreBean.java
 */
import java.rmi.RemoteException;
import javax.ejb.*;
import java.util.*;
import java.util.logging.*;
import javax.naming.*;

/**
 * Implementatie van de ECommerce businesscode.
 */
public class ECommerceStoreBean implements SessionBean {

    /** Creeer een logger. */
    private Logger logger = Logger.getLogger(this.getClass().toString());

    /* EJB's */
    private LocalCustomerHome customerHome = null;
    private LocalOrderHome orderHome = null;
    private LocalProductHome productHome = null;
```

```
    public ECommerceStoreBean() {}

    // ======================================
    // SessionBean methoden
    // --------------------------------------

    public void ejbCreate() throws CreateException {

        logger.info("create ECommerceStoreBean");
        createOrActivateEjb();
    }

    public void ejbActivate() {

        logger.info("activate ECommerceStoreBean");
        try {
            createOrActivateEjb();
        }
        catch (Exception e) {}
    }

    private void createOrActivateEjb() throws CreateException {

        try {
            customerHome = lookupCustomer();
            orderHome = lookupOrder();
            productHome = lookupProduct();
        } catch (NamingException ex) {
            throw new CreateException(ex.getMessage());
        }
    }
   public void ejbPassivate() {

        customerHome = null;
        orderHome = null;
        productHome = null;
    }

    public void ejbRemove() {}
    public void setSessionContext(SessionContext sc) {}

    // Private methods
    public java.util.Properties getProducts() throws RemoteException {
        Properties products = new Properties();
        products.put("Product 1","1.50");
        products.put("Product 2","1.75");
        products.put("Product 3","2.00");
        products.put("Product 4","2.25");
        return products;
    }

    private LocalCustomerHome lookupCustomer() throws NamingException {

        Context initial = new InitialContext();
        Object objref = initial.lookup("java:comp/env/ejb/Customer");
        return (LocalCustomerHome) objref;
    }

    private LocalOrderHome lookupOrder() throws NamingException {

        Context initial = new InitialContext();
        Object objref = initial.lookup("java:comp/env/ejb/Order");
        return (LocalOrderHome) objref;
    }

    private LocalProductHome lookupProduct() throws NamingException {
```

```
        Context initial = new InitialContext();
        Object objref = initial.lookup("java:comp/env/ejb/Product");
        return (LocalProductHome) objref;
    }

}
```

3.2.5. (CMP Entity) Enterprise Bean Code

De code voor container managed persistent moet aan een aantal voorwaarden voldoen. Ten eerste moet de class public abstract gedefinieerd worden en moeten het volgende geïmplementeerd worden:

- `EntityBean` interface
- `ejbCreate` en `ejbPostCreate` methoden
- `get` en `set` access methoden, voor de persistente en velden voor relaties, gedefinieerd als `abstract`.
- Evt select methoden `abstract` definiëren
- De home methode
- De business methoden

De volgende methoden horen niet thuis in een CMP entity bean:

- finder methoden
- De `finalize` methode

Entityvelden

De velden die opgeslagen moeten worden door de EJB container, moeten als methoden abstract gedefinieerd worden. De code ziet er als volgt uit:

```
/*
 * An entity bean with container-managed persistence has persistent and
 * relationship fields. These fields are virtual, so you do not code them in
 * the class as instance variables. Instead, you specify them in the bean's
 * deployment descriptor. To permit access to the fields, you define
 * abstract get and set methods in the entity bean class.
 *
 * Access Methods for Persistent Fields
 */
public abstract String getCustomerId();
public abstract void setCustomerId(String id);

public abstract String getName();
public abstract void setName(String name);

public abstract String getAddress();
public abstract void setAddress(String address);

public abstract String getCity();
public abstract void setCity(String city);

public abstract String getPostcode();
public abstract void setPostcode(String postcode);

/*
 * Access Methods for Relationship Fields
 */
public abstract Collection getOrders();
public abstract void setOrders(Collection orders);
```

3.2.6. De J2EE Applicatie Cliënt

De webclient is geschreven JSP.

Omdat het bij deze opdracht alleen om de Enterprise Beans gaat, is er in het ontwerp van de webclient niet veel tijd gestoken. Het model is model 1. In echte applicatie zou altijd een model 2 (MVC) model gebruikt moeten worden. Daarnaast moet ook de opmaak gescheiden worden van de code, dat is in deze applicatie ook niet het geval, echter voor dit voorbeeld voldoet de code.

De code in index.jsp voert de volgende taken uit:

- Localiseer de home interface
- Creeer een enterprise bean instantie
- Roep de business methoden aan.

Localiseer de Home Interface

In de home interface van ECommerceStoreHome is een methode aanwezig die een nieuwe enterprise bean creëert. Echter voordat deze methode aangeroepen kan worden, moet het object gelocaliseerd worden. Dit doen we mbv JNDI:

```
<%-- ECommerce webclient --%>
<%@ page import="ECommerceStore, ECommerceStoreHome, Cart, CartItem,
javax.ejb.*, java.util.*, java.math.*, javax.naming.*, ja
vax.rmi.PortableRemoteObject, java.rmi.RemoteException" %>
<%!
   // Initialiseren te gebruiken (stateless) sessionbean.
   private ECommerceStore store = null;

   // EJB ECommerceStore opvragen
   public void jspInit() {
      try {
         InitialContext ic = new InitialContext();
         Object objRef = ic.lookup("java:comp/env/ejb/ECommerceStore");
         ECommerceStoreHome home = (EcommerceStoreHome)
             PortableRemoteObject.narrow(objRef, ECommerceStoreHome.class);
         store = home.create();
      } catch (RemoteException ex) {
            System.out.println("Couldn't create bean."+ ex.getMessage());
      } catch (CreateException ex) {
            System.out.println("Couldn't create bean."+ ex.getMessage());
      } catch (NamingException ex) {
            System.out.println("Unable to lookup home: "+ "ECommerceStore " +
                ex.getMessage());
      }
   }
```

Aanroepen Business Methoden

Nadat we de referentie naar een instantie van ECommerceStoreBean hebben, kunnen we de methoden voor het opvragen van de producten en het plaatsen van een order aanroepen:

```
<select name="item">
<%
    /* Alle aanwezige producten afdrukken in een selectlijst. */
    Properties products = store.getProducts();
    Iterator i = products.keySet().iterator();

    while (i.hasNext()) {
        String productName = (String) i.next();
        String productPrice = (String) products.getProperty(productName);

        out.println("<option value='" + productName + "'>" + productName +
            " - " + productPrice + "</option>");
    }
%>
</select>

[...]

// Order verzenden en totaalprijs opvragen.
    if ("Verzenden".equals(submit)) {
        // de ECommerceStore heeft een methode waaraan een order
        // kan worden toegevoegd.

        // Creeer het object voor de klant. key = naam klant.
        String customerId = name;
        store.createCustomer(customerId, name, address, city, postcode);
        // Creeer een bestelling. key=datum + naam klant
        String storeId = (new Date()).toString() + name;
        store.createOrder(storeId);

        // De toe te voegen producten creeren
        Collection productIds = new HashSet();
        CartItem[] products = cart.getItems();
        for (int i=0; i < products.length; i++) {
            String id = products[i].getName();
            String productName = products[i].getName();
            String price = products[i].getPrice();
            String amount = products[i].getAmount();
            store.createProduct(id, productName, price, amount);
            productIds.add(id);
        }
        store.addProductsToOrder(productIds, storeId);
        store.addCustomerToOrder(CustomerId, storeId);
```

Hulpclasses

Om tijdens te websessie de producten in het winkelwagentje te kunnen bewaren, wordt er gebruik gemaakt van een JSP sessie en twee hulp classes, namelijk `Cart` en `CartItem`. De werking van JSP sessies wordt hier verder niet op ingegaan.

4. CREËER DE J2EE APPLICATIE

De applicatie bestaat uit twee delen: De EnterpriseBeans (drie CMP entity beans en een stateless session bean) en de webcomponent.

Voordat de twee afzonderlijke delen gaan saemnstellen, creëren we het hoofdarchief waar de complete applicatie in komt te staan, de zogenaamde EAR. EAR staat voor Enterprise ARchive.

- Start de deploytool en selecteer `File -> New -> Application`.
- Klik op `Browse`.
- Navigeer naar de map waar we de applicatie willen opslaan.
- Type als naam voor de J2EE applicatie: `ECommerceStoreApp.ear`.
- Klik op `New Application`.
- Klik `Ok`.

We hebben nu onze lege applicatie (EAR) gecreëerd.

4.1. DE ENTERPRISE BEANS INPAKKEN

De enterprise beans zijn al gecompileerd. Nu gaan we via de wizard van de deploytool de EJB JAR creëren en aan onze J2EE applicatie toevoegen.

Bij het gebruik van J2SDK1.4 kan de wizard om een Enterprise Bean te creëren niet gestart worden. Er treedt een fout op. Gebruik daarom J2SDK1.3.

Kies om de wizard te starten: `File -> New -> Enterprise Bean`.

- Selecteer `Create New JAR in file application`.
- Type als JAR naam `ECommerceStoreJAR`.
- Blader naar de gecompileerde classbestanden van de Enterprise Beans en voeg deze toe aan de JAR.
- Klik `Ok`, klik `Next`.

- Selecteer het type Bean.
- Selecteer welke classes de Bean, Home en Interface zijn.
- Geef de EJB een naam, `ECommerceStoreEJB`.
- Klik op `Finish` (De overige instellingen zijn hier niet van belang).

Er is zojuist een `deployment descriptor` gecreëerd.

4.2. DE WEBCLIENT INPAKKEN

De webclient hoeft niet eerst gecompileerd te worden, omdat het alleen uit een JSP pagina bestaat. De webcontainer in de J2EE server handelt het compileren af.

Via de wizard gaan we een web archief (`WAR`) creëren.

Start de wizard: `File -> New -> Web component`.

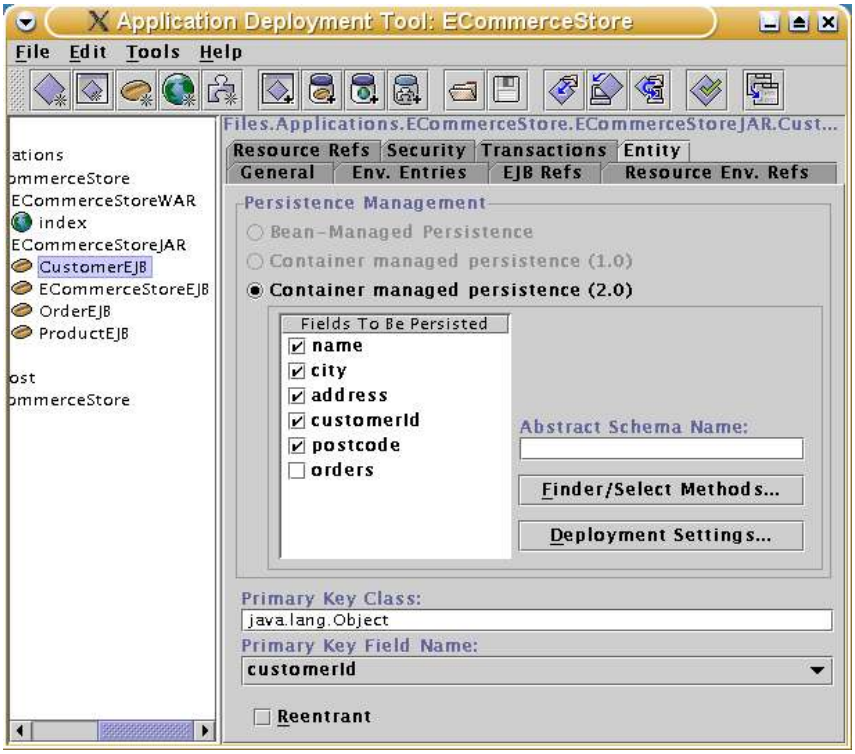
- Selecteer `Create new WAR in application`.
- Selecteer `EcommerceStoreApp`.
- Kies als naam voor de webcomponent `ECommerseStoreWAR`.
- Blader naar index.jsp en voeg deze toe aan de inhoud van het webachief.
- Klik op `Next`.
- Selecteer als type component `JSP`.
- Klik op Next.
- Selecteer `index.jsp` en klik op `Finish`.

4.3. ENTITYBEANS RELATIES

Relaties tussen de entitybeans instellen:

Figuur 6, relatie tussen een klant (customer) en een bestelling (order). Een order heeft 1 klant, maar een klant kan meerdere orders bezitten.

4.3.1. Entity tab



Figuur 7, we selecteren voor de bean "container managed persistence (2.0)". Deze versie ondersteunt relaties. De velden die zorgen voor de relaties moeten niet persistence zijn.

Selecteer de velden die in de database bewaard dienen te worden. Velden die bedoeld zijn voor het vastleggen van relaties moeten niet vastgelegd worden in de database. Deze zijn daarom ook niet aangevinkt.

4.4. REFERENTIE IN WEBCLIENT INSTELLEN

De `lookup` methode in de `JSP` pagina refereert naar de Enterprise Bean op de volgende manier:

```
Object objRef = ic.lookup("java:comp/env/ejb/ECommerceStore");
```

Dit wordt op de volgende manier ingesteld:

- Klik op de `ECommerceWAR` knoop.
- Selecteer de `EJB Refs` tab.
- Klik op `Add`.
- Voer de volgende waarden in: `ejb/ECommerceStore, Session, Remote, ECommerceStoreHome, ECommerceStore`.

4.5. INSTELLEN DATABASE KOPPELING

In de testapplicatie is de MySQL database gebruikt. Elke database waarvoor een JDBC driver beschikbaar is, is geschikt om de data in op te slaan.

De JDBC bibliotheken met de classes voor de koppeling met database zijn vanaf de officiele mysql website neer te laden. De naam van de factory class is:

```
com.mysql.jdbc.Driver
```

Dit is de naam die we in de J2EE server moeten opgeven als JDBC driver. Om de driver vast te leggen moeten we deze toevoegen aan het configuratie bestand. Dit bestand is te vinden in de config map en heet resource.properties. Hier is tevens de naam van de datasource, de database gebruikersnaam en het bijbehorende wachtwoord te definieren. Als naam voor de datasource typen we DB.

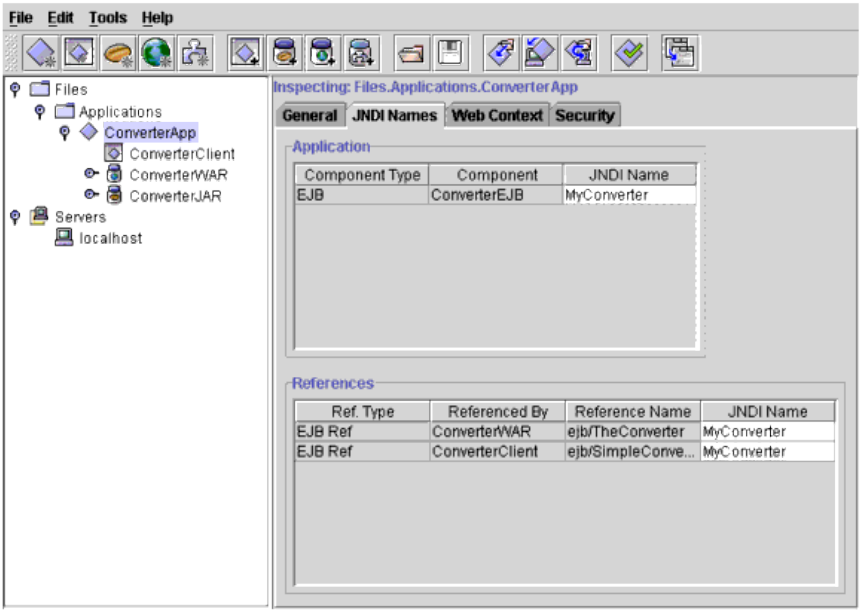
4.5.1. Referentie instellen

1. In deploytool, selecteer achtereenvolgens de entitybeans.
2. Maak de `Resource Refs` tab actief.
3. Klik op `Add`.
4. Typ in de `Coded Name field`, `jdbc/DB`.

   De JNDI namen voor de JDBC DataSource objecten worden opgeslagen in de `java:comp/env/jdbc` subcontext.
5. Select `javax.sql.DataSource` als datatype.
6. Selecteer `container` in de Authentication combo box.

4.6. JNDI NAAM INSTELLEN

Via de naam die hier ingesteld wordt, is de EJB bereikbaar. Alleen de EcommerceStore sessie bean moet beschikbaar zijn buiten de EJB container, vandaar dat we alleen voor deze bean een JNDI naam instellen.



Figuur 8, een referentie naar een EJB instellen

4.7. DEPLOY DE J2EE APPLICATIE

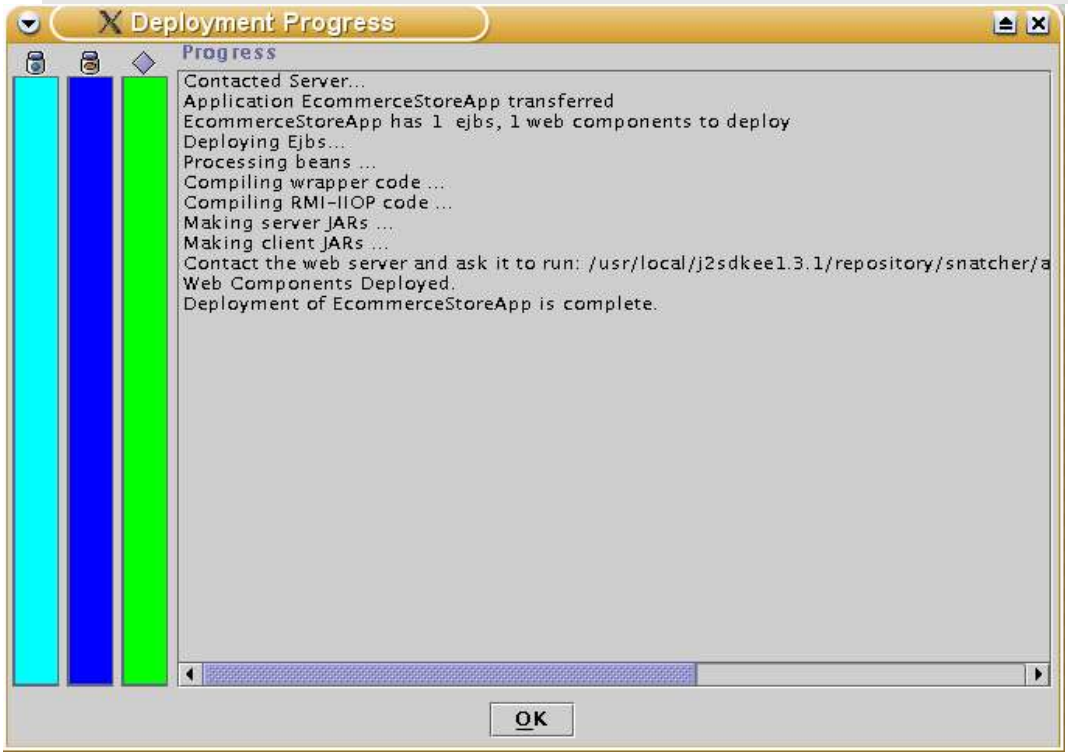
De applicatie kan nu ge-`deployed` worden, klaargezet voor de server om geïnstalleerd te worden.

- Selecteer de applicatie.
- Typ in de tab `Web Context` de naam `ecommercestore`.
- Selecteer `Tools -> Deploy`.
- Selecteer als server `localhost` en klik op `Ok`.

Na onderstaand bericht is de applicatie succesvol klaargezet voor installatie op de J2EE server.

De applicatie wordt gestart door in een webbrowser onderstaand adres in te type. (er vanuit gaande dat de server staat ingesteld op de standaard waarden)

```
http://localhost:8000/ecommercestore
```



5. CONCLUSIE

Door de grootte en complexiteit van de J2EE specificatie is het als softwareontwikelaar erg moeilijk om met J2EE een begin te krijgen. Het is op een hoger nivo goed voor te stellen hoe de op componenten gebaseerde applicaties eruit moeten zien, maar als de zaken wat concreter gaan worden, wordt het al een stuk moeilijker.

Naast het feit dat de specificatie uitgebreid en complex is, zijn er tussen de verschillende applicatieservers ook nog eens verschillen in de vorm van leverancier specifieke uitbreidingen en eigen gereedschappen.

Ik heb persoonlijk gemerkt dat J2EE nog niet uitontwikkeld is. Er staat met J2EE een goede basis en ook het feit dat het gestandaardiseerd is, is een goed punt. De voordelen die EJB's zouden moeten brengen vallen in de praktijk tegen. Vaak wordt de zaak complexer dan dat het zou moeten zijn. Ook de performance valt in de praktijk tegen bij het gebruik van EJB's.

Objecten zouden in de praktijk nooit via serialisatie aangesproken moeten worden, maar als dat mogelijk is via een referentie. Dat laatste komt de performance positief ten goede.

Ook de ontwikkelaar van de open-source applicatieserver JBoss heeft hier z'n eigen gedachten over. Het (Engelstalige) document waar Marc Fluery z'n mening over de applicatieserver en EJB's geeft “Why I love EJB's, “Blue” white paper” is te vinden op het adres http://www.jboss.org/modules/html/blue.pdf.