VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VÝVOJOVÉ PROSTŘEDÍ PRO APLIKACI ZALOŽENOU NA JAVA EE

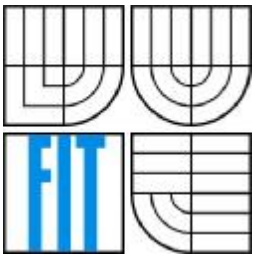
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL CHUDÝ

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

VÝVOJOVÉ PROSTŘEDÍ PRO APLIKACI ZALOŽENOU NA JAVA EE

DEVELOPMENT ENVIROMENT FOR JAVA EE BASED APPLICATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL CHUDÝ
AUTHOR

VEDOUCÍ PRÁCE — Ing. BORIS ŠUŠKA
SUPERVISOR

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá základními technologiemi specifikace Java Enterprise Edition (Java EE). Ve stručnosti popisuje jejich architekturu, využití a implementační detaily. Dále je tato práce návodem pro přípravu vývojového prostředí a prostředí počítače pro aplikace založené na těchto technologiích. Součástí práce jsou i jednoduché, ukázkové aplikace demonstrující jednotlivé technologie a možnosti jejich použití. Práce popisuje jak tyto projekty vytvořit, sestavit a spustit.

Abstract

This bachelor thesis deals with essential Java Enterprise Edition specification technologies. It briefly describes their architecture, usage and implementation details. Furthermore this thesis is a tutorial for development and computer enviroment preparation for Java EE based technologies. There are added simple applications as a part of this thesis which demonstrate individual technologies and their usage options. It describes how to create, build and run this projects.

Klíčová slova

Java, Java EE, Maven 2, JavaServlet, JSP, JSF, EJB 3.0, JPA, Hibernate, webové služby, JBoss aplikační server, Eclipse IDE

Keywords

Java, Java EE, Maven 2, JavaServlet, JSP, JSF, EJB 3.0, JPA, Hibernate, web services, JBoss application server, Eclipse IDE

Citace

Chudý Michal: Vývojové prostředí pro aplikaci založenou na Java EE, bakalářská práce, Brno, FIT VUT v Brně, 2010

Vývojové prostředí pro aplikaci založenou na Java EE

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Borisa Šušku. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Michal Chudý
19.5.2010

Poděkování

Rád by som v prvom rade poďakoval Ing. Borisovi Šuškovi za ochotu, pomoc a trpezlivosť počas vypracovávania bakalárskej práce. Ďalej by som chcel poďakovať mojej rodine a mojej priateľke Zuzane za podporu, ktorú mi poskytli počas môjho štúdia na FIT VUT v Brne.

Obsah

1 Úvod

Technológia Java je založená na platforme Java Platform, Standard Edition (momentálne vo verzii 1.6), ktorá je v súčasnosti jednou z najpoužívanejších a najžiadanejších programovacích platforiem na trhu [27]. Pre vývoj webových aplikácií, podnikových (enterprise) a informačných systémov bola vytvorená platforma Java Enterprise Edition (Java EE, predtým J2EE), ktorá je momentálne distribuovaná vo verzii 5. Táto práca popisuje základné technológie špecifikácie Java EE a obsahuje postup ako pripraviť vývojové prostredie a prostredie počítača pre aplikácie založené na týchto technológiách.

Práca je rozdelená do dvoch veľkých logických celkov. Prvým (kapitola 2) je už spomínaný popis technológií Java EE, ktorý má pripraviť teoretický základ pre druhú časť práce (kapitola 3 a 4), ktorá popisuje a názorne za pomoci obrázkov prevádza čitateľa vytváraním jednoduchých aplikácií postavených na daných technológiách.

Úvod kapitoly 2 má za úlohu predstaviť platformu Java EE a vysvetliť jej základné špecifiká a pre ňu typické pojmy, ako napríklad „viacvrstvová architektúra“, „kontajner“, „komponenty“, atď. Jadro tejto kapitoly opisuje už samotné technológie. Táto časť je logicky členená na podkapitoly, z ktorých každá opisuje technológie jednej vrstvy viacvrstvového aplikačného modelu. Kapitola 3 sa venuje príprave prostredia počítača pre vývoj aplikácií založených na Java EE. Podáva informácie o tom, odkiaľ získať nástroje pre vývoj a beh týchto aplikácií, ako ich nakonfigurovať a používať. Na túto kapitolu priamo nadväzuje kapitola 4, v ktorej sa čitateľ dozvie ako za pomoci týchto nástrojov vytvoriť, zostaviť a spustiť aplikácie implementované za pomoci základných Java EE technológií.

Súčasťou tejto práce je aj DVD disk s plne nakonfigurovaným vývojovým prostredím a so zdrojovými kódmi ukážkových aplikácií. Z dôvodu obmedzeného rozsahu práce v nej nebolo možné uvádzať rozsiahlejšie úrivky zdrojových kódov, preto je pre detailné pochopenie vysvetľovanej problematiky potrebné popri čítaní tejto práce nazerať do zdrojových súborov na priloženom disku.

Jednou z hlavných úloh tejto práce je oboznámiť čitateľa, začiatočníka v programovaní na platforme Java EE, so samotnou platformou a pomôcť mu pri vytváraní aplikácií založených na technológiách, ktoré platforma Java EE ponúka.

2 Platforma Java Enterprise Edition a prostriedky pre tvorbu Java EE aplikácií

2.1 Úvod do Java Enterprise Edition

Aplikácie postavené na platforme Java EE používajú distribuovaný viacvrstvový aplikačný model. Komponenty takejto distribuovanej aplikácie môžu bežať vo viacerých virtuálnych strojoch na jednom alebo viacerých serveroch. Samozrejme je možné navrhnúť a vytvoriť aj nedistribuovanú aplikáciu [9].

Podľa [9] je pre Java EE typická troj alebo štvor-vrstvová architektúra. Druhá menovaná je však zriedkavejšia. Viacvrstvové architektúry sa ukázali ako omnoho škálovateľnejšie a flexibilnejšie ako dvoj-vrstvová, klient-server architektúra. Každá vrstva by mala byť závislá len na vrstve nachádzajúcej sa priamo pod ňou a procesy špecifické pre danú vrstvu by mali byť pred ostatnými vrstvami skryté. V troj-vrstvovej architektúre rozoznávame:

- Webovú vrstvu (*Web tier*) – jej úlohou je zobrazovať informácie pre klienta. Jej obsahom sú napríklad technológie *JavaServlet* (kapitola 2.2.1.1), *JavaServer Pages* (kapitola 2.2.1.2) alebo *JavaServer Faces* (2.2.1.4).
- Vrstvu aplikačnej logiky (*Business tier*) – obsahuje komponenty, ktoré zabezpečujú prenos dát medzi webovou a integračnou vrstvou a ich prípadné spracovanie. Tieto komponenty sú nezávislé na použitom užívateľskom rozhraní. Jedná sa napríklad o *Enterprise JavaBeans* komponenty (kapitola 2.2.3.1)
- Integračnú vrstvu (*Enterprise Information System tier – EIS tier*) – obsahuje hlavne *systémy pre správu databáz / databázové systémy (Database Management System - DBMS)*. Táto vrstva nie je pod kontrolou Java EE serveru ale Java EE obsahuje rôzne technológie pre komunikáciu s ňou. Sú to napríklad *Java DataBase Connectivity (JDBC API)* pre komunikáciu s relačnými databázami alebo *Java Connector Architecture (JCA)* pre pripojenie ku iným EIS systémom.

Vo štvor-vrstvových aplikáciách je pridaná *klientská vrstva (client tier)*. Jedná sa o aplikácie, ktoré využívajú namiesto webového klienta (webový prehliadač), klienta aplikačného, ktorý beží

na klientskom počítači a zvyčajne obsahuje *grafické užívateľské rozhranie (GUI)* a časť aplikačnej logiky [1].

Java EE aplikácie sú zložené z komponentov. Sú to samostatné programové jednotky, ktoré sú kompilované ako obyčajné Java triedy, avšak verifikované podľa špecifikácie Java EE a ich životný cyklus je riadený Java EE serverom. Medzi tieto komponenty patria napríklad JavaServlet, JavaServer Pages (JSP), JavaServer Faces (JSF) alebo Enterprise JavaBeans (EJB). Pre každý typ komponenty poskytuje Java EE server služby v podobe *kontajnera*. Kontajnery vykonávajú nekonfigurovateľné služby, napríklad správa životného cyklu komponenty a služby konfigurovateľné, napríklad zabezpečenie komponenty. Java EE server poskytuje dva druhy kontajnerov:

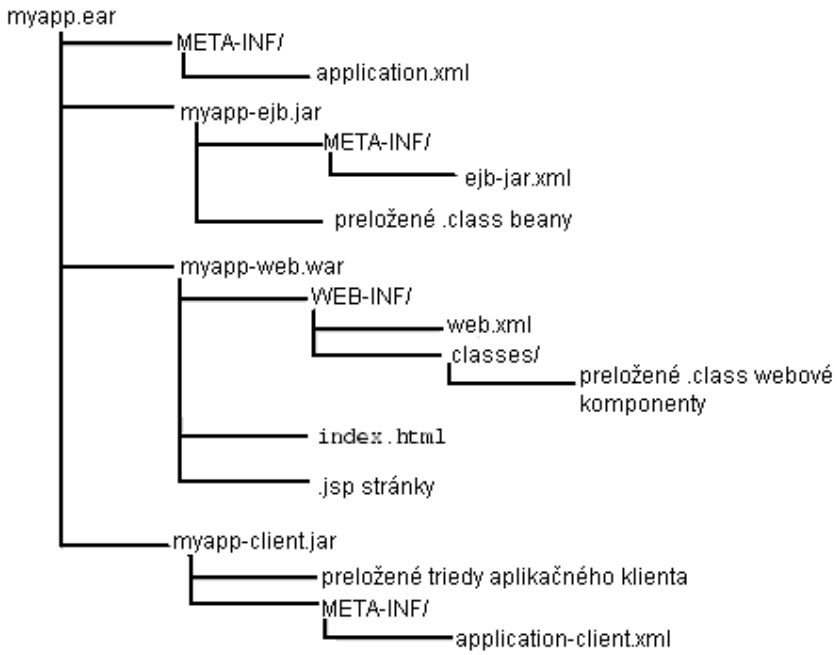
- Webový kontajner – stará sa o životný cyklus webových komponentov (napr. Servlety, JSP, JSF, Webové služby)
- Enterprise JavaBeans (EJB) kontajner – stará sa o životný cyklus EJB komponentov

Mimo Java EE server sa nachádzajú:

- Kontajner aplikačného klienta – stará sa o beh aplikačného klienta na klientskom počítači
- Kontajner pre applety – zabezpečuje beh appletov na klientskom počítači, skladá sa z webového prehliadača a *Java Plug-in* u.

Odstavec prebratý z [1].

V [9] sa uvádza, že Java EE aplikácie sa nedodávajú v podobe holých zdrojových súborov ale sú zabalené do archívov. Najčastejšie sa stretávame s *web archívmi (súbory s príponou .war)* a s *EJB JAR archívmi (*.jar)*. Oba tieto archívy obsahujú zdrojové kódy a *popisovač nasadenia (deployment descriptor* – v tejto práci budem používať anglický termín*)*, web archívy obsahujú aj JSP stránky, Servlety, obrázky alebo statické HTML stránky a podobne. Deployment descriptor je dokument, napísaný v jazyku XML, ktorý poskytuje Java EE serveru informácie o konfigurácii aplikácie a jej komponentov. Pre web archívy sa nazýva *web.xml*. Ak máme v aplikácii web aj EJB archív, je ich možné zabaliť do jedinej jednotky, takzvaného *podnikového balíku (Enterprise Archive - *.ear)*. Ten musí obsahovať ešte jeden deployment descriptor nazvaný *application.xml*, v ktorom sú popísané Java EE moduly, z ktorých sa skladá daná aplikácia. Obrázok 2.1 znázorňuje hierarchiu podnikového balíka.



Obrázok 2.1: Hierarchia podnikového balíka. Obrázok prebraný z [16].

2.2 Java EE technológie

2.2.1 Technológie webovej vrstvy

Keďže webová vrstva je pri vývoji aplikácie subjektom najčastejších zmien, je dôležité aby v nej bol jasne oddelený aplikačný kód, ktorý má za úlohu spracovanie požiadaviek, od zobrazovaných dát a prezentačného kódu. Tento stav je možné dosiahnuť pomocou architektúry *Model-View-Controller (MVC)*. MVC obsahuje tri typy objektov. Model je objekt obsahujúci dáta, ktoré sa majú zobraziť pre klienta, view zobrazuje dané dáta napríklad pomocou JSP stránok a kontroler je objekt reagujúci na klientské požiadavky a podľa toho upravuje model [9]. MVC architektúra sa často nazýva Model – 2 architektúrou keďže striktne oddeľuje prezentačný kód od aplikačnej logiky, zatiaľ čo napríklad JavaServelty obsahujú obe zložky a sú preto nazývané aj Model – 1 architektúrou [1].

Pre udržanie informácii týkajúcich sa jedného klienta sa vo webových aplikáciách používa *sedenie (session)*. Je implementované za pomoci rozhrania *javax.servlet.http.HttpSession* a je dostupné z objektu reprezentujúceho požiadavku pomocou volania metódy *getSession()*. Do sedenia môžme uložiť objektové hodnoty atribútov

namapované podľa názvu. Sedenie je jeden z takzvaných *priestorových objektov (scope objects)*, ktoré slúžia na zdieľanie informácií (objektových hodnôt) v podobe atribútov daného priestorového objektu. Ďalšími priestorovými objektami sú *webový kontext*, ktorý je prístupný z webových komponentov v rámci danej webovej aplikácie, požiadavka *(request)* a *stránka (page)*. Odstavec prevzaný z [1].

2.2.1.1 JavaServlet Technology

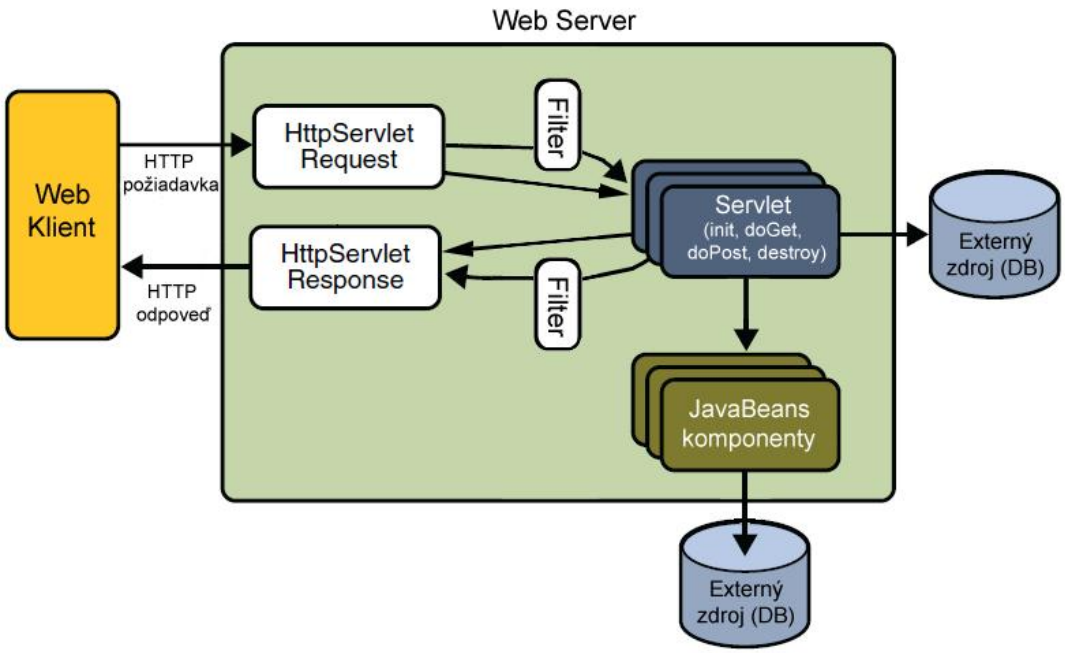
„JavaServlet je trieda programovacieho jazyka Java, ktorá sa používa na rozšírenie schopností serverov, ktoré hosťujú aplikácie založené na aplikačnom modeli typu požiadavka – odpoveď (request - response) [1].“ Pre zjednodušenie budeme v nasledujúcom texte namiesto pojmu JavaServlet používať pojem servlet. Inak povedané, servlety v spojení s webovými servermi (dokážu odpovedať na rôzne typy požiadaviek – nie len http) dynamicky spracuvávajú požiadavky a generujú odpovede pre webových klientov. V dobe svojho nástupu mali servlety spolu so skriptovacími jazykmi (PHP, Perl) za úlohu nahradiť CGI skripty, ktoré boli od začiatku 90. rokov jediným spôsobom generovania dynamického obsahu webových stránok [4].

Keďže servlety sú Java triedy, bežia v rámci virtuálneho stroja na webovom serveri a tým pádom sú prenosné na všetky operačné systémy, ktoré podporujú platformu Java. Servlety nevyžadujú nainštalovanú Javu u klienta [4]. Podporu pre servlety poskytujú knižnice *javax.servlet* a *javax.servlet.http*. Pri implementácii musí programátor dbať na zachovanie dedičnosti. Každý servlet musí rozširovať buď triedu *javax.servlet.GenericServlet* alebo *javax.servlet.http.HttpServlet*, ktoré obsahujú metódy, potrebné pre vytvorenie všeobecných alebo http servletov. Trieda *HttpServlet* rozširuje *GenericServlet* [1]. V nasledujúcich odstavcoch sa budeme venovať servletom spracuvávajúcim http požiadavky.

O celý životný cyklus servletu sa stará webový kontajner (Obrázok 2.1). Ak je vyvolaná požiadavka a servlet, ktorý ju má obslúžiť neexistuje, webový kontajner vytvorí jedinú inštanciu triedy implementujúcej daný servlet, pred prijatím http požiadavky zavolá metódu *init*, ktorá môže byť prispôsobená (napríklad sa uskutoční inicializácia perzistentných dát). Požiadavka môže byť danému servletu predaná priamo alebo jej predaniu bude predchádzať prechod *filtrom*. Následne dôjde ku volaniu servisnej metódy, ktorá obslúži danú požiadavku a vygeneruje odpoveď. Servisné metódy abstrahujú funkcionalitu základných metód http protokolu (Get, Delete, Options, Post, Put, Trace). Typickým vzorcom pre správanie servisných metód je získať informácie z http požiadavky, pristúpiť ku externým zdrojom (dnes často vykonávané za pomoci EJB komponentov) a naplniť objekt predstavujúci odpoveď. Požiadavka je po prijatí webovým kontajnerom konvertovaná na objekt implementujúci rozhranie *javax.servlet.http.HttpServletRequest*, ktorý obsahuje všetky potrebné informácie (parametre

a k nim priradené objektové hodnoty prenášajúce dáta od klienta, informácie o http protokole) pre spracovanie požiadavky a vytvorenie odpovede, ktorá je reprezentovaná objektom implementujúcim rozhranie *javax.servlet.http.HttpServletResponse.* Odpoveď, tak ako požiadavka, môže byť predávaná klientovi priamo alebo prechodom cez filter. Ak je servlet ďalej nepotrebný, kontajner zavolá jeho *destroy* metódu, ktorá spôsobí uvoľnenie všetkých zdrojov alokovaných servletom.

Filter je objekt, ktorý môže meniť hlavičku alebo obsah http požiadavky. Jeho úlohou je poskytovať akúsi doplnkovú funkcionalitu webovým komponentám a predpripraviť pre nich požiadavku, prípadne upraviť odpoveď pre klienta. Požiadavka môže prechádzať jedným alebo aj celou *reťazou* filtrov. Mapovanie filtrov na URL vzory sa špecifikuje vo web.xml [1].



Obrázok 2.2: Proces prijatia a spracovania požiadavky a vygenerovanie odpovede servletom, čiastočne prevzaté z [1].

Čo sa týka implementácie, existuje mnoho otvorených *frameworkov* (aplikačných rámcov), ktoré rozširujú a zdokonaľujú JavaServlet Technology. Najznámejšie a najpoužívanejšie z nich, podľa [3], sú:

- J2EE BluePrints – nejedná sa v pravom slova zmysle o framework. Jeho obsahom sú návrhové vzory a doporučenia pre používanie Java EE technológií [5].
- Apache Struts – jeden z najpopulárnejších MVC frameworkov.
- JavaServer Faces – technológia z produkcie Sun-u, popísaná v kapitole 2.2.1.4.

- Apache Turbine – jeden z najstarších na servletoch založených frameworkov. Pre implementáciu frontendu používa MVC model a zapuzdruje komponenty, ktoré môžu byť použité aj samostatne (napr. ORM nástroj *Apache Torque*) [6].

2.2.1.2 JavaServer Pages Technology

Technológia JavaServer Pages (JSP), vznikla z potreby oddelenia kódu zobrazovacej vrstvy od kódu aplikačného, ktorý generuje dynamický obsah stránok. Tento problém sa vyskytuje najmä u servletov. Zdrojový kód servletu je pri zložitejších stránkach zle čitateľný a ťažko udržiavateľný [7].

JSP technológia definuje textové dokumenty písané v jazyku z rodiny SGML tzv. JSP stránky, ktoré okrem statických dát, vyjadrených pomocou jedného z týchto jazykov, obsahujú aj JSP elementy, ktoré vytvárajú dynamický obsah (JSP elementy sú bližšie popísané v kapitole 2.2.1.3). JSP stránky sú v mnohých prípadoch spájané s architektúrou MVC, v ktorej slúžia ako zobrazovacia vrstva (view). Samozrejme, JSP stránky je možné využiť aj samostatne (mimo architektúry MVC), keďže majú schopnosť vyvolávať JavaBeans komponenty a ich metódy za pomoci elementov pre štandardné akcie, napr. <*jsp:useBean*> [7]. Rozhodnutie, či použiť JSP, servlety alebo JSP v kombinácii s MVC záleží na pomere HTML ku Java kódu. Ak máme veľa HTML a málo Java kódu mali by sme použiť JSP, ak je pomer opačný mal by byť použitý servlet a ak je pomer rovnaký mali by sme zvážiť použitie rámca MVC [2].

Ak je požiadavka mapovaná na danú JSP stránku, dôjde ku prekladu JSP stránky na servlet. Z tohto dôvodu je životný cyklus JSP veľmi podobný životnému cyklu servletu. Jednou z výhod JSP oproti servletom je, že JSP sú kompilované automaticky, čo zvýhodňuje najmä vývoj. Pri zmene zdrojového kódu servletu bolo potrebné celú aplikáciu preložiť, zostaviť a nasadiť na server, zatiaľ čo JSP sú automaticky prekladané na servlety za behu aplikácie. Paradoxne, v tejto vlastnosti tkvie aj nevýhoda JSP. Implementácia aplikačnej logiky vstavaná priamo do JSP je náchylná na chyby, keďže ku prekladu na servlet dochádza až za behu aplikácie. Eliminovať túto vlastnosť je možné napríklad pomocou MVC rámca, kde je väčšina Java kódu umiestnená v kontroleri a JSP stránky sú použité výhradne na zobrazovanie [3].

V JSP stránkach sa od verzie 2.1 využíva unifikovaný výrazový jazyk (Unified Expression Language - EL). Jeho hlavnou úlohou je dynamicky sprístupňovať dáta JavaBeans komponentov. Ilustrujem to na príklade e-shopu, v ktorom máme JavaBean shoppingCart, reprezentujúci nákupný košík klienta. Košík obsahuje atribút numberOfItems obsahujúci počet produktov v košíku. Vývojár je schopný z JSP stránky sprístupniť daný atribút pomocou jednoduchej bodkovej notácie: `${sessionScope.shoppingCart.numberOfItems}`.

V príklade pristupujeme ku objektu *shoppingCart* cez objekt *sessionScope*. Jedná sa o jeden z implicitných priestorových objektov definovaných v EL (predstavených v kapitole 2.2.1) a okrem *sessionScope* rozoznávame ešte napríklad *pageScope, requestScope, applicationScope* a viacero iných, ktorých popis však nie je predmetom tejto práce, podrobnejšie informácie môžete nájsť v [1] strana 162. V spomenutých objektoch sú namapované názvy premenných vyskytujúcich sa v danom priestore ku ich hodnotám. EL je schopný ako okamžitého, tak aj odloženého vyhodnotenia výrazov (aj metód), ktoré sa používa najmä v technológii JavaServer Faces, ktorá podporuje viacfázový životný cyklus stránky. Odstavec prevzatý z [1].

Na skupiny JSP stránok, ktoré obsahujú ten istý kód alebo pre ktoré chceme špecifikovať rovnaké vlastnosti, môžeme aplikovať návrhový vzor dekorátor. Vo web.xml špecifikujeme skupinu JSP stránok a na nich namapujeme jeden alebo viac URL vzorov, ktoré budú pre nich platné. Často sa jedná napríklad o hlavičky stránok, nastavenie kódovania a podobne. Dekorátory používa napríklad aj aplikačný rámec *SiteMesh*, ktorý je užitočným nástrojom pri vývoji zložitých a rozsiahlych webových stránok.

2.2.1.3 JavaServer Pages Standard Tag Library

Technológia JavaServer Pages Standard Tag Library (JSTL) poskytuje značky, ktoré zapuzdrujú základnú funkcionalitu JSP stránok. Z dôvodu lepšej prehľadnosti a oddeleniu funkcionality sú značky rozdelené do piatich základných skupín.

Oblasť	Možnosti	Predpona
Jadro (Core)	Podpora premenných	c
	Riadenie toku programu	
	Manažment URL	
	Ostatné	
XML	Jadro (parsovanie, prístup k dátam)	x
	Riadenie toku programu	
	Transformácia	
I18N	Nastavenie lokálneho prostredia	fmt
	Formátovanie správ	
	Formátovanie dátumu a čísel	
Databáza (Database)	Práca s SQL	sql
Funkcie (Functions)	Dĺžka kolekcie	fn
	Manipulácia reťazcov	

Tabuľka 2.1: Prehľad základných skupín JSTL značiek, prevzaté z [1]

- Core Tag Library (Jadro) – obsahuje značky pre podporu základných operácií pri práci s premennými, s tokom programu a prácou s URL. Pomocou týchto značiek je napríklad možné nastaviť hodnotu premennej, vytvoriť *if–else* štruktúru alebo uskutočniť prechod kolekciou. Značky pre prácu s tokom programu eliminujú potrebu používania *skriptletov* (popísané nižšie v tejto kapitole).
- XML Tag Library (XML) – poskytuje značky pre podporu práce s XML dokumentmi. Podobne ako u Core Tag Library, aj tu sa nachádzajú značky pre prácu s premennými alebo s tokom programu, prispôsobené však technológii XML.
- Internationalization Tag Library (I18N) – slúži na nastavenie lokálneho prostredia, nastavenie správ pre užívateľa a na formátovanie dátumových, číselných, menových a iných hodnôt.
- SQL Tag Library (Databáza) – slúži na sprístupnenie perzistentých zdrojov priamo z JSP. Tieto značky je doporučené používať len v jednoduchých aplikáciách, pre zložitejšie je prístup k databázovým systémom obyčajne sprostredkovaný prostredníctvom JavaBeans komponentov [1].
- JSTL Functions (Funkcie) – poskytuje značky pre zistenie dĺžky kolekcií a pre manipuláciu s reťazcami.

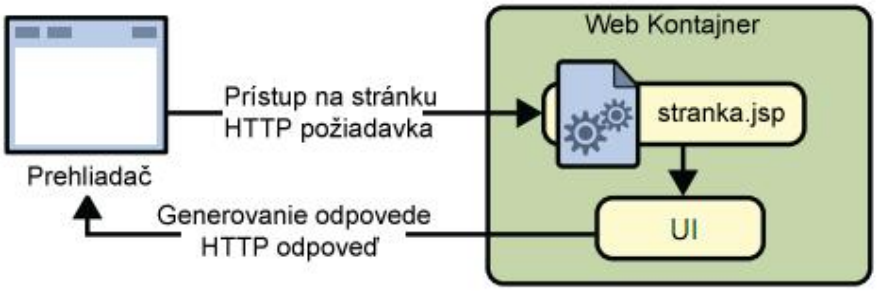
Keďže JSTL podporuje základné operácie s JSP stránkami, technológia JSP umožňuje užívateľom definovať vlastné značky implementujúce inú funkcionalitu. Vývojár si môže vytvoriť vlastné JSP značky, ktoré môžu byť následne použité aj v iných aplikáciách alebo použiť jednu z mnohých otvorených užívateľských knižníc dostupných na internete. Sú to napríklad knižnice pre prácu s AJAX-om, knižnice pre tvorbu dynamických menu od tvorcov frameworku Struts alebo knižnicu pre spracovanie formulárov vo frameworku Spring a mnoho iných. Tieto a ďalšie príklady na JSTL knižnice sú uvedené v [8].

Pomocou *skriptletov* môžeme písať Java kód (prípadne kód iného programovacieho jazyka) priamo do kódu JSP stránky. Keďže jedným z podnetov pre vznik JSP bolo oddelenie aplikačného kódu od prezentačného, tak použitie väčšieho množstva skriptletov je skôr krokom späť ku technológii JavaServlet [7].

2.2.1.4 JavaServerFaces Technology

Nasledujúca kapitola je prevzatá z [1]. Technológia JavaServer Faces (JSF) je podobne ako JSP, rámcom pre vytváranie užívateľských rozhraní (User Interface - UI). Jej dvoma základnými súčasťami sú aplikačné programové rozhranie (Application Programming Interface - API), definujúce UI komponenty a popisujúce ich správanie a knižnice značiek pre JSP, slúžiace na

reprezentáciu UI komponentov a na ich naviazanie na podporné objekty. Aplikácia postavená na technológii JSF obsahuje JSP stránky s JSF značkami a užívateľské rozhranie, ktoré spravuje objekty odkazované z týchto stránok. Toto rozhranie je súborom UI komponentov, validátorov, prevodníkov údajov a JavaBeans komponentov (Obrázok 2.3).



Obrázok 2.3: JSF aplikácia. Prevzaté z [1].

Každá JSF aplikácia musí vo svojom deplyment descriptor-e (web.xml) deklarovať inštanciu triedy *FacesServlet*, ktorý prijíma prichádzajúce HTTP požiadavky, predáva ich na spracovanie a inicializuje zdroje.

Aby bolo možné používať JSF komponenty v JSP stránkach, je nutné deklarovať v týchto stránkach dve štandardné knižnice značiek. Prvou, je knižnica HTML komponentov identifikovaná v značkách prefixom *„h“*. Druhou, je knižnica značiek reprezentujúca funkcie jadra technológie JSF, používajúca prefix *„f“*. Každá JSF stránka je reprezentovaná stromom komponentov nazvaným *view – pohľad*. Značka *view* je definovaná v základnej knižnici značiek a reprezentuje koreňový prvok zobrazenia. Všetky ostatné JSF značky (komponenty) sa musia nachádzať v jej vnútri.

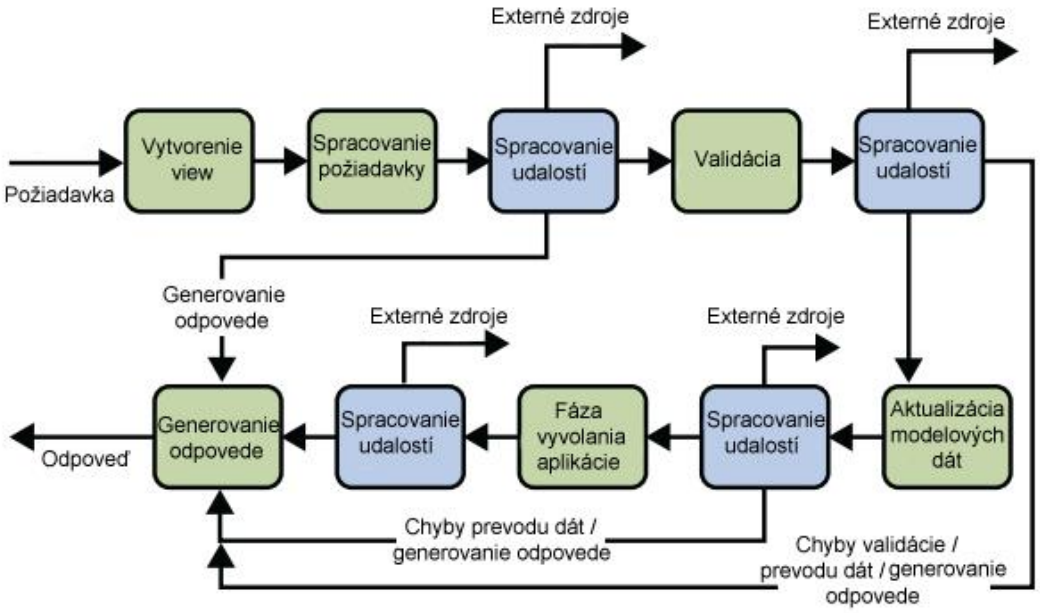
Komponenty sú konfigurovateľné a znovupoužiteľné elementy, ktoré môžu byť jednoduché (napr. vstupné textové pole), alebo zložené (napr. tabuľka). Väčšina UI komponentov je reprezentovaná v HTML stránkach jednou alebo viacerými HTML značkami (napr. komponenta *UIForm* je analogická s HTML značkou *form*). Architektúra JSF aplikácií je zložená z nasledujúcich súčastí:

- Skupina tried definujúcich stav a správanie UI komponentov – triedy a rozhrania pre popis funkcionality komponentov. Keďže tieto triedy sú plne rozšíriteľné, je možné definovať vlastné UI komponenty.
- Model generovania komponentov – zatiaľ čo základná funkcionalita UI komponentov je popísaná triedami a rozhraniami, generovanie komponentov majú na starosti oddelené generátory. Výhodou je, že jednu komponentu môžeme vykresliť rôznymi spôsobmi za

pomoci rôznych generátorov (napr. komponenta *UISelectOne* môže byť do HTML generovaná ako skupina *radio buttonov*, ako *combo box* alebo ako *list box*).

- Model prevodu dát – v JSF aplikáciách je možné previazať komponentu so serverovým objektom (napr. JavaBean-om). V tomto prípade existujú dva pohľady na dáta obsiahnuté v komponente. Prvým je prezentačné zobrazenie – najčastejšie textový reťazec, ktorý sa zobrazí v prehliadači a druhým je modelové zobrazenie, v ktorom sú dáta reprezentované pomocou dátových typov (int, boolean, ...). Technológia JSF automaticky konvertuje dáta medzi týmito dvoma pohľadmi. V prípade, že užívateľ potrebuje previesť dáta iné ako štandardné, je tu možnosť vytvorenia vlastných konvertorov.
- Model udalostí a naslúchačov (listeners) – slúži na zachytenie udalosti (napr. kliknutie na tlačidlo) a stará sa o jej spracovanie.
- Validačný model – sa stará o kontrolu (validáciu) vstupných dát zadávaných užívateľom. Tak ako model prevodu dát aj validačný model obsahuje súbor vstavaných tried pre validáciu bežných dát a je v ňom možné definovať vlastné validátory.
- Navigačný model – definuje za pomoci pravidiel špecifikovaných v konfiguračnom xml súbore navigáciu medzi stránkami. Zjednodušene povedané, stará sa o výber nasledujúcej stránky ak bolo kliknuté na hypertextový odkaz alebo tlačidlo.
- Podporné objekty – sú objekty dodržujúce JavaBeans špecifikáciu (vysvetlené v kapitole 2.2.3) poskytujúce najčastejšie validáciu dát, obsluhu udalostí a rozhodovací proces, ktorý určí nasledujúcu stránku. Každý takýto objekt je potrebné zaregistrovať a nakonfigurovať v JSF konfiguračnom xml súbore.

Životný cyklus JSF stránky je podobný ako u JSP stránok s tým rozdielom, že medzi prijatím požiadavky a odoslaním odpovede JSF stránkou je jej životný cyklus rozdelený na niekoľko fáz, čo znázorňuje Obrázok 2.4.



Obrázok 2.4: Štandardný životný cyklus JSF stránky. Prevzaté z [1].

Po prijatí požiadavky, JSF stránka vytvorí úplne nový view a začne s jej spracovaním. Ak sa jedná o prvý prísup klienta na stránku, takzvaný ***initail request***, JSF vytvorí view a prechádza do fázy generovania odpovede pretože ostatné fázy sú v tomto prípade zbytočné. V prípade, že sa jedná o odoslanie formulára, tzv. ***postback***, vykonávajú sa aj ostatné fázy životného cyklu stránky. Ak sú pri spracovaní požiadavky vyvolané udalosti, sú volané príslušné metódy na ich obslúženie. V prípade, že je požiadavka spracovaná, vygeneruje sa odpoveď a odošle klientovi. Je možné, že JSF stránka potrebuje pristúpiť ku iným zdrojom, webovým službám alebo vygenerovať odpoveď, ktorá neobsahuje JSF komponenty. V tomto prípade je preskočená fáza generovania odpovede a aplikácia pristupuje k týmto zdrojom (znázornené šípkou „Externé zdroje“). Najčastejšie však JSF spracováva požiadavku vytvorenú odoslaním formulára, čiže postupne dochádza ku validácii a prevodu dát a ich aktualizácii v dátovom modeli. Obe tieto fázy nemusia skončiť generovaním odpovede, ale môžu pristupovať ku externým zdrojom. Fáza vyvolania aplikácie spracováva udalosti ako presmerovanie na inú stránku. Konečnou fázou životného cyklu je generovanie odpovede a jej zaslanie klientovi, najčastejšie v podobne HTML stránky.

2.2.2 Webové služby

Webové služby sú aplikácie dostupné na internete pomocou štandardných internetových protokolov ako HTTP a SMTP. Základom webových služieb je technológia XML, ktorá poskytuje jazykovo a platformne neutrálne vyjadrenie dát, z čoho vyplýva, že webové služby sú využívané hlavne na prepájanie a spoluprácu medzi systémami postavenými na rôznych technológiách (napr. Java EE - .NET). Jednoduchým príkladom na webovú službu môže byť rezervačný systém požičovne automobilov integrovaný do stránky leteckej spoločnosti [10].

Podľa [10] sú pre webové služby charakteristické nasledujúce vlastnosti:

- Dátová vrstva založená na XML – zabezpečuje neutralitu na platforme, jazyku a transportnom protokole.
- Voľne zviazané s klientom – konzument (klient) webovej služby nie je spojený so službou priamo. Zatiaľ čo sa rozhranie služby môže zmeniť, neovplyvní to klientovu schopnosť komunikovať s webovou službou.
- Možnosť prijímať synchrónne aj asynchrónne požiadavky.
- Podporujú *RPC (Remote Procedure Call – vzdialené volanie procedúr)* – umožnujú klientom vzdialene vyvolávať procedúry, funkcie a metódy za pomoci protokolu postaveného na XML.
- Podporujú výmenu dokumentov – XML je schopné reprezentovať nie len dáta ale aj komplexné dokumenty (adresa, kniha, zmluva, ...).

Z dôvodu štandardizácie, by mal každý užívateľ alebo poskytovateľ webových služieb podporovať nasledujúce technológie:

- *SOAP (Simple Object Access Protocol)* – protokol, ktorý poskytuje obálky pre transport XML dokumentov cez internetové protokoly ako HTTP, SMTP, FTP a zároveň definuje štruktúry pre vyvolávanie RPC volaní.
- *WSDL (Web Service Description Language)* – jazyk pre popis webových služieb založený na technológii XML, popisujúci rozhranie webových služieb.
- *UDDI (Universal Description, Discovery and Integration)* – je celosvetový register poskytujúci informácie o webových službách, vyhľadávanie a integráciu webových služieb.

Prebrané z [10].

Podľa [1] je pre implementáciu webových služieb v Java EE použitý štandard *JAX-WS (JAVA API pre XML webové služby)*. Implementácia koncového bodu webovej služby vyžaduje nasledujúce požiadavky:

- Implementujúca trieda musí byť anotovaná anotáciou *javax.jws.WebService* alebo *javax.jws.WebServiceProvider*.
- Metódy implementujúce business logiku musia byť *public* a nesmú byť *static* alebo *final*.
- Metódy implementujúce business logiku prístupné klientom webových služieb musia byť anotované *javax.jws.WebMethod* anotáciou a musia mať vstupné parametre a návratové hodnoty kompatibilné s dátovými typmi podporovanými *JAXB*.
- Implementujúca trieda nesmie byť *final* ani *abstract*.
- Implementujúca trieda musí obsahovať *public* bezparametrický konštruktor a nesmie obsahovať *finalize* metódu.

Implementujúca trieda môže obsahovať anotácie *javax.annotation.PostConstruct* alebo *javax.annotation.PreDestroy* na svojich metódach.

2.2.2.1 JAXB

JAXB (Java Architecture for XML binding) je technológia poskytujúca nástroje pre automatickú konverziu XML dokumentov na Java objekty a naopak. Proces prevodu XML dokumentov na Java objekty sa nazýva anglickým termínom *unmarshalling(synonymum: čítanie - reading)* pričom opačný proces sa nazýva *marshalling(synonymum: zapisovanie - writing)*. Pri oboch procesoch JAXB poskytuje možnosť validovať zdorojové súbory (XML dokumenty alebo Java objekty) pred prevodom na cieľové súbory.

Jednotlivé komponenty XML dokumentov definujúce jednoduché dátové typy sú kompatibilné s dátovými typmi v Jave. Jedná sa napríklad o mapovanie elementu *xsd:string* na *java.lang.String* alebo *xsd:int* na primitívny typ *int*.

JAXB poskytuje možnosti pre konfiguráciu generovaných Java objektov ako aj XML dokumentov. Pre konfiguráciu generovaných objektov z XML dokumentov môžme využiť riadkové anotácie v zdrojových XML súboroch alebo externý konfiguračný XML súbor, ktorý je predaný JAXB prekladaču. Pre konfiguráciu generovaných XML súborov sú využívané anotácie z balíka *javax.xml.bind.annotations*. Tento balík poskytuje anotácie napríklad pre definíciu koreňového elementu XML dokumentu, definíciu mapovania atribútov Java objektov alebo mapovania Java typu *enum*. Podkapitola prebraná z [1].

2.2.2.2 StAX

StAX (Streaming API for XML – Prúdové API pre XML) je technológia poskytujúca obojsmerný, udalosťami riadený, XML *pull* [1] parser. Jeho implementácia je jednuduchšia ako SAX a nemá tak vysoké pamäťové nároky ako DOM.

Vo všeobecnosti existujú dva prístupy ku spracovaniu XML dokumentov: *DOM – Data Object Model* a *streaming – prúdové čítanie dokumentu* (v práci budem používať anglický termín). DOM vytvorí v pamäti kompletnú reprezentáciu XML dokumentu ako objekt, čo na jednej strane poskytuje maximálnu flexibilitu pri prístupe ku hociktorej časti dokumentu avšak na strane druhej, vyžaduje vysoké nároky na procesor a pamäť. Streaming spracováva XML dokument sekvenčne, čo spôsobuje menšie nároky na systémové zdroje avšak poskytuje pohľad len na jednu časť dokumentu v danom čase.

Samotné StAX API je zložené z *cursor a iterator API*. Cursor API poskytuje kurzor, ktorým je možné pohybovať sa po XML dokumente od začiatku po koniec, vždy len smerom vpred a ukazujúci len na jeden XML element v jednom momente. Iterator API reprezentuje XML stream ako sadu objektov reprezentujúcich XML udalosti v poradí ako sú vyvolávané pri čítaní dokumentu. Medzi XML udalosti môžme zaradiť napríklad narazenie na začiatok dokumentu, začiatok elementu, komentár, atribút, atď. Podkapitola prebraná z [1].

2.2.2.3 SAAJ

Technológia SAAJ (SOAP with Attachments API for Java – API pre SOAP s prílohami pre Javu) je využívaná hlavne na spracovanie (prijímanie, odosielanie, písanie, čítanie) SOAP správ na pozadí webových služieb implementovaných pomocou JAX-WS.

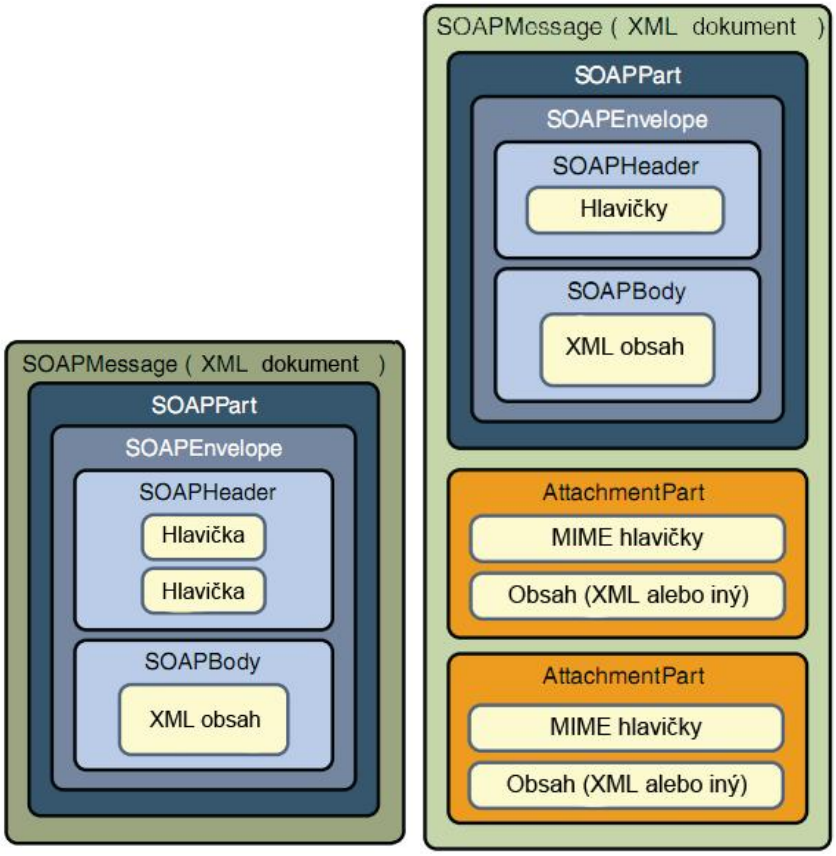
SAAJ správy dodržujú SOAP štandardy, ktoré predpisujú formát a vlastnosti pre tieto správy. Podľa týchto štandardov sa SOAP správa skladá z nasledujúcich prvkov (podľa [10]):

- *SOAP Envelope – obálka* – je koreňový XML element, ktorá definuje menný priestor a definíciu schémy (DTD) pre danú správu.
- *SOAP header – hlavička* – hlavička a telo SOAP správy sú syntakticky rovnaké a štandard SOAP nedefinuje, čo by sa malo nachádzať v hlavičke a čo v tele. Hlavička zvyčajne slúži na definíciu odkazov pre SOAP procesor, ktorý príjme danú správu.
- *SOAP body – telo* – telo XML dokumetu.

[1] Rozdiel medzi *pull* a *push* parsovaním je, že pri *pull* parsovaní, klient volá metódy XML parsera, len keď potrebuje získať dáta z XML dokumetu, zatiaľ čo pri *push* parsovaní XML parser zasiela (tlačí) dáta klientovi bez ohľadu na to, či je klient pripravený ich spracovať alebo nie.

SAAJ poskytuje pre reprezentáciu SOAP správy triedu *SOAPMessage*, ktorá po inštanciovaní obsahuje objekt *SOAPPart*, obsahujúci *SOAPEnvelope*, ktorý automaticky obsahuje objekty *SOAPHeader* a *SOAPBody*. Existujú dva hlavné typy SOAP správ:

- Správa bez príloh – jednoduchý dokument, ktorý neobsahuje žiadne binárne dáta.
- Správa s prílohami – SOAP správa obsahujúca okrem dát vyjadriteľných v XML aj binárne dáta vyjadrené štandardom MIME. Pre vyjadrenie príloh, SAAJ poskytuje triedu *AttachmentPart*.



Obrázky 2.5 a 2.6: SOAP správa bez a s prílohami. Prebrané z [1].

SOAP správy sú posielané a prijímané cez *point-to-point* spojenie, ktoré je reprezentované objektom triedy *SOAPConnection*. Správy sú odosielané volaním metódy *call* a následne je spojenie blokované, kým neprijme odpoveď. Podkapitola prebraná z [1].

2.2.3 Technológie business vrstvy

Business vrstva alebo vrstva aplikačnej logiky je niečo ako prostredník medzi webovými komponentmi a databázovým systémom a zároveň výkonným jadrom Java EE aplikácie.

Poskytuje operácie ako napríklad prístup do databázy, komunikáciu s inými systémami a samotnú aplikačnú, výpočtovú logiku (napríklad vypočítanie ceny položiek v košíku, zaslanie potvrdzovacieho emailu pri registrácii užívateľa a pod.). Podľa [1] je aplikačná logika implementovaná pomocou technológie *Enterprise JavaBeans* (kapitola 2.2.3.1), avšak existujú aj iné metódy. Je to napríklad použitím zjednodušených (lightweight) *JavaBean* komponentov [12].

JavaBeans sú Java triedy, ktoré obsahujú klasické metódy, atribúty sprístupňované za pomoci metód *get* a *set* a bezparametrický konštruktor. Rozdiel medzi EJB a JavaBeans je podľa [11] ten, že JavaBeans sú vývojové komponenty, ktoré nie je možné na rozdiel od EJB nasadiť (deploy) do EJB kontajnera. JavaBeans sa preto využívajú najmä na vytváranie väčších komponent, ktoré už sú nasaditeľné.

2.2.3.1 Enterprise JavaBeans

Enterprise JavaBean (alebo skrátene enterprise bean) je serverový Java EE komponent, ktorý obsahuje metódy pre vykonávanie aplikačnej (business) logiky [1]. Enterprise bean môže pozostávať z jednej alebo viacerých Java tried ale pre klientov sú ich implementácie skryté a navonok viditeľné je len rozhranie enterprise beanu. Klientom môže byť napríklad servelt, kontroler alebo iný enterprise bean. V poslednom prípade môže dôjsť k vyvolávaniu metód z rady ďalších enterprise beanov, čo umožňuje rozdelenie riešenia úloh rôznych druhov medzi nimi [11].

Ako už bolo spomenuté v kapitole 2.1, enterprise beany bežia v EJB kontajneri a ten sa stará o ich životný cyklus a poskytuje im systémové služby, čo umožnuje vývojárovi sústrediť sa v plnej miere na riešenie problémov týkajúcich sa business logiky. Ďalšou výhodou je, že ich použitie podporuje nasadenie tenkých (thin) klientov, ktoré neobsahujú takmer žiadnu aplikačnú logiku a koncentrujú sa len na prezentáciu užívateľského rozhrania. Keďže EJB sú znovupoužiteľné a prenosné komponenty, vývojár môže pri tvorbe novej aplikácie použiť už existujúce enterprise beany [1].

Podľa [11] poznáme tieto typy enterprise beanov:

- *Session* – definujú aplikačné procesy a poskytujú služby (vykonávajú akcie) pre klientov. Tieto akcie môžu byť už spomenutý prístup do databázy, komunikácia s inými systémami a samotná aplikačná logika.
- *Message-Driven* – sú podobné ako session beany s tým rozdielom, že klient s nimi nekomunikuje cez rozhranie ale len za pomoci správ. Message-driven beany neobsahujú žiadne rozhranie, len triedy. Správy (messages) sú najčastejšie prijímané a odosielané asynchrónne za pomoci *JMS API* (kapitola 2.2.4.1). Session beany

dokážu taktiež príjmať a odosielať správy, avšak na rodiel od message-driven beanov len synchrónne [1].

- *Entity* – modelujú aplikačné dáta a reflektujú obraz databázových tabuliek na Java objekty. Používajú sa v JPA (kapitola 2.2.4.2)

Nasledujúce princípy EJB technológie budem ilustrovať na príklade session beanu. Klient môže komunikovať so session beanom len cez jeho rozhranie, ktoré je jediným pohľadom na bean, prístupným pre klienta. Výhodou implementácie rozhrania je, že klient nie je závislý na zmenách v implementácii session beanu. Implementácia metód sa môže zmeniť, ale klientovi je vystavené len nezmenené rozhranie, takže nie je potrebné meniť jeho zdrojový kód. Klient sa k enterprise beanu môže pripojiť vzdialene alebo lokálne. Ak sa jedná o vzdialené pripojenie, rozhranie beanu musí byť anotované anotáciou @Remote alebo jeho implementujúca trieda anotovaná tou istou anotáciou spolu s názvom rozhrania - @Remote(NázovRozhrania.class). Klientom môže byť webová komponenta, aplikačný klient alebo iný eneterprise bean. Pri lokálnom pripojení môže byť klientom webová komponenta alebo enterprise bean. Lokálne rozhranie alebo jeho implenetujúca trieda môže ale nemusí byť anotovaná ako @Local [1]. Lokálne rozhrania sa podľa [11] využívajú z dôvodu zvýšenia rýchlosti na vyvolávanie metód objektov existujúcich v jednej aplikácii, pričom cez vzdialené (remote) rozhranie je možné vyvolávať metódy odkiaľkoľvek. Podľa [1] eneterprise bean obsahuje nasledujúce zložky:

- *Trieda enterprise beanu* – implementuje metódy špecifikované v aplikačnom (business) rozhraní beanu. Pre rôzne typy beanov je implementácia tried odlišná. V session beanoch, ich implementujúca trieda(y) obsahuje hlavne aplikačnú logiku, u entity beanov je to logika spojená s manipuláciou dát a v message-driven beanoch sú to metódy pre prijímanie a odosielanie správ a vyvolávanie metód session beanov, ktoré vedia tieto požiadavky obslúžiť [11].
- *Business rozhranie* - definuje metódy, ktoré sú implementované triedou enterprise beanu.
- *Pomocné triedy* – ostatné triedy používané v enterprise beane (pomocné triedy, výnimky, ...).

Tak ako pri webových archívoch aj EJB komponenty musia mať špecifikovaný deployment descriptor v jazyku XML. V tomto dokumente sa nachádzajú informácie pre EJB kontajner definujúce správu životného cyklu, kontrolu transakčnosti, nastavenie zabezpečenia daného beanu alebo u entity beanov informáciu či bean spravuje svoju perzistentnosť sám alebo necháva túto činnosť na EJB kontajneri [11].

EJB kontajner vytvorí zväčša jedinú inštanciu enterprise beanu. Pre jej použitie poskytuje kontajner takzvanú injektáž závislostí (Dependency Injection - DI). Pri DI nie je potrebné predávať si inštancie objektov cez konštruktory, ale ak objekt obsahuje odkaz na iný objekt (napríklad enterprise bean), tak pri štarte kontajnera sú oba vytvorené a v prvom objekte bude inicializovaný odkaz na ten druhý. Pre funkčnosť injektáže závislostí musí byť daný odkaz anotovaný anotáciou *@EJB* alebo musí byť táto závislosť definovaná v deployment descriptore [1]. Ak by sme chceli odkaz na inštanciu enterprise beanu inicializovať manuálne, musíme využiť metódu JNDI lookup. Táto metóda využíva objekt *InitialContext*, ktorý je podľa [11] vstupným bodom do JNDI stromu. Potom môžeme vytvoriť referenciu na daný enterprise bean pomocou lookup metódy vyvolanej z InitialContextu, ktorej predáme ako argument názov, pod ktorým je enterprise bean zaregistrovaný. Pri vyvolaní metódy injektovaného enterprise beanu, EJB kontajner vyvolá príslušnú metódu v EJB objekte na serveri. Hoci sa zdá, že metóda beží lokálne, je spustená vzdialene.

Pre delegáciu enterprise beanu sú všetky jeho súčasti zbalené do *ejb-jar* balíka, ktorý je následne možné spolu s webovými komponentmi zabaliť do *.ear archívu a pripojiť do kontajnera ako samostatnú aplikáciu alebo používať osamotene v Java EE aplikáciách.

2.2.3.2 JavaMail

JavaMail, ako už názov tejto technológie napovedá, je aplikačným rámcom, ktorý poskytuje súbor tried a rozhraní tvoriacich e-mailový systém. Tieto triedy implementujú internetové e-mailové protokoly a dodržujú štandardy RFC822 a RFC2045. Architektúra tohto aplikačného rámca sa skladá z týchto častí:

- *Abstraktná vrstva* - obsahuje abstraktné triedy a rozhrania, ktoré zapuzdrujú funkcionalitu podporovanú všetkými e-mailovými systémami.
- *Internetová vrstva* – implementuje časť abstraktnej vrstvy používajúc internetové štandardy RFC822 a MIME.

Základnými funkciami poskytovanými JavaMail aplikačným rámcom sú:

- Vytvorenie správy skladajúcej sa z hlavičky a dátového bloku.
- Vytvorenie *Session* objektu, ktorý sa stará o autentifikáciu užívateľa, riadi prenos a kontroluje prístup ku úložisku správ.
- Posielanie a prijímanie správ.

Dvoma základnými komponentmi JavaMail API sú triedy *Message* a *Session*. Abstraktná trieda Message definuje sadu atribútov tvoriacich e-mailovú správu. Jedná sa napríklad o informácie o prijímateľovi, definícia typu dát v tele správy a dáta, obalené objektom *DataHandler*. Trieda

Message implementuje rozhranie *Part*, ktoré definuje metódy pre špecifikáciu dát a ich formátovanie. Samotné dáta sú reprezentované kolekciou bytov alebo odkazom na ne, pričom objekt Message s nimi manipuluje len za pomoci *JavaBeans Activation Frameworku* (kapitola 2.2.3.3). Toto oddelenie umožňuje spracovať a prenášať dáta hocijakého typu pomocou volania tých istých metód. Klient vytvorí správu inštanciovaním potomka triedy Message, nastaví atribúty ako adresa prijímateľa, predmet a vloží obsah správy. Správa je poslaná volaním metódy *send* triedy *Transport*, ktorá predstavuje transportného agenta, ktorý dopraví správu ku prijímateľovi.

Správy sú uložené v objektoch triedy *Folder*, ktoré môžu vytvárať stromovú štruktúru podadresárov a správ a obsahujú metódy na ich získanie, editáciu, kopírovanie a mazanie. Trieda *Store* definuje databázu, v ktorej je hierarchicky udržiavaná adresárová štruktúra a v nej obsiahnuté správy. Táto trieda špecifikuje aj prístupový protokol (IMAP4, POP3, ...) pre sťahovanie správ a tak isto poskytuje metódy pre nadviazanie a prerušenie spojenia s databázou.

JavaMail API podporuje aj objekty triedy Message zložené z viacerých častí, pričom každá z nich definuje svoju vlastnú sadu atribútov a obsah. Podkapitola prebraná z [13].

2.2.3.3 JavaBeans Activation Framework

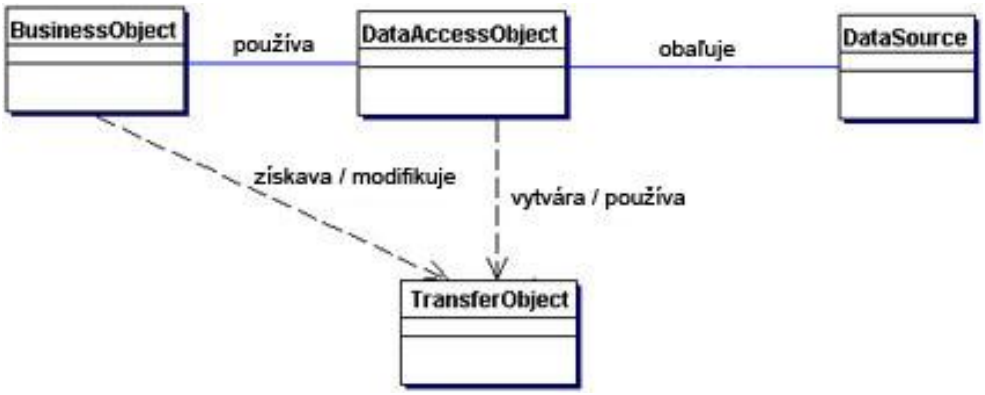
JavaBeans Activation Framework (JAF) je súčasťou Java EE špecifikácie lebo ho používa JavaMail. JAF určuje typ sprístupňovaných dát, zapuzdruje prístup k nim, poskytuje operácie závislé na rôznych typoch dát a inštanciuje softwarové komponenty (JavaBean-y), ktoré vykonávajú dané operácie nad danými typmi dát [14].

Zdroj [14] opisuje JAF, jeho funkcie a základnú architektúru na jednoduchom príklade prehliadača súborov podobnému Windows Explorer. V aplikácii dochádza ku trom diskrétnym operáciám. Prvou je inicializácia, v ktorej je pre každý súbor vytvorený *DataSource* objekt, ktorý obaľuje dáta a poskytuje prístup k nim (napr. k súborovému systému, URL, IMAP serverom, ...), potom je pre súbor vytvorený *DataHandler* objekt, ktorému je v konštruktore predaný DataSource. DataHandler poskytuje prístup ku *CommandMap* objektu (mape príkazov), ktorý obsahuje súbor operácií, možných prevádzať nad daným typom dát. Druhou fázou je získanie a zobrazenie zoznamu operácií (príkazov) nad daným súborom. Po kliknutí na súbor sa otvorí zoznam možných operácií v popup okne. Tieto operácie sú získané požiadavkou na DataHandler objekt, ktorý získa MIME typ dát z DataSource objektu a zobrazí operácie prislúchajúce danému typu dát udržiavané v CommandMap. Poslednou fázou je vykonanie operácie nad daným súborom. Po kliknutí na danú operáciu z ponuky v popupe, aplikácia získa JavaBean zodpovedný za jej spracovanie z *CommandInfo* objektu a predá mu dané dáta, ktoré JavaBean spracuje a vykoná nad nimi požadovanú operáciu.

Tento JavaBean by mal implementovať rozhranie *CommandObject*, ktoré poskytuje metódy pre naviazanie akcie na tento bean. Metóda *setCommandContext* previaže bean s príslušným DataHandler-om, z ktorého môžeme vyvolať metódu *getContent*, ktorá vracia príslušný objekt, na ktorý bolo kliknuté.

2.2.4 Technológie integračnej vrstvy

V úvode by som chcel spomenúť návrhový vzor *DAO - Data Access Object* – objekt pre prístup k dátam. Jedná sa o programovacie paradigma, ktoré vytvára vrstvu logiky pre prístup k dátam medzi databázovým systémom a vrstvou implementujúcou aplikačnú logiku. Tento návrhový vzor prispieva ku čistejšiemu OO návrhu aplikácie. Táto vrstva implementuje pre prístup k databáze jednu z ORM technológií, napr. Hibernate, využíva implementáciu klasických Java rozhraní a riadenie transakcií a bezpečnosti prenecháva business vrstve. Pre každú entitu je vytvorené DAO rozhranie, ktoré obsahuje metódy pre prístup k tabuľke reprezentujúcej túto entitu [9].



Obrázok 2.7: Architektúra využívajúca DAO. Obrázok prebraný z [17].

2.2.4.1 JavaMessage Service API

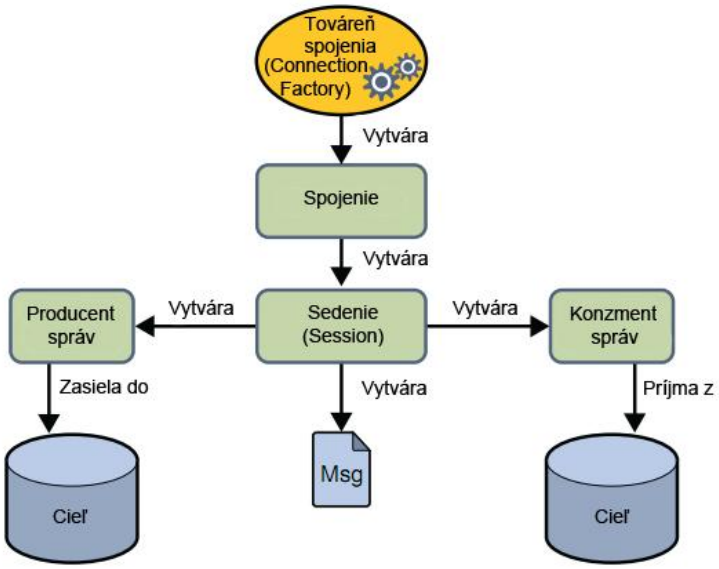
JMS (JavaMessage Service API) je aplikačné rozhranie, ktoré umožňuje Java EE aplikáciám vytvárať, príjmať, odosielať a čítať správy. Správy sú asynchrónnym spôsobom komunikácie medzi aplikáciami alebo ich komponentmi. Operácie so správami ako prenášanie, vytváranie alebo čítanie sú poskytované takzvaným agentom/poskytovateľom správ (provider). O komunikácii pomocou správ hovoríme, že je voľne previazaná, čo znamená, že odosielateľ a prijímateľ nemusia byť pri komunikácii prítomní v rovnakom čase, ba dokonca nemusia vedieť nič o druhej strane okrem typu správy a adrese odosielateľa/prijímateľa.

Okrem voľnej previazanosti môžeme o JMS tvrdiť, že je asynchrónnou a spoľahlivou technológiou. Asynchrónnosť zabezpečuje, že správa nemusí byť klientovi doručená na

požiadanie (požiadavka - odpoveď) ale hocikedy a spoľahlivosť zabezpečuje, že správa je doručená práve raz. Vďaka týmto vlastnostiam môže byť JMS použité napríklad v nasledujúcich prípadoch:

- Aplikačné komponenty nezávisia na informáciách o rozhraniach ostatných komponentov, takže môžu byť jednoducho nahradené.
- Aplikácia musí byť v prevádzke aj za podmienky, že nie sú všetky komponenty online alebo nie sú spustené súbežne.
- Aplikačný model dovoľuje komponentám zaslať správy a pracovať ďalej, bez potreby prijatia okamžitej odpovede.

Systém správ môže fungovať na dvoch princípoch. Prvým je *point-to-point*, ktorý je založený na frontách správ, odosielateľoch a prijímateľoch. Každá správa má jediného príjmateľa a je adresovaná do špeciálnej fronty, v ktorej je držaná, pokiaľ ju klient neprevezme alebo pokiaľ jej neskončí životnosť. Point-to-point je používané v prípadoch, keď každá správa musí byť spracovaná práve jedným príjmateľom. Druhým princípom je *publish/subscribe* (publikovať/odoberať). Jedná sa o systém podobný diskusným skupinám, kde zasielatelia (publishers) a odoberatelia sú väčšinou anonymní. Zasielatelia zašlú správu na „fórum“ a systém sa postará o doručenie tejto správy všetkým odoberateľom daného „fóra“. Tento princíp je používaný v prípade, že každá správa môže byť spracovaná 0 – N prijímateľmi. JMS pracuje implicitne asynchrónne, čiže klient (komponenta, aplikácia) má zaregistrovaný *message listener* (mechanizmus čakania na správu), ktorý ju po prijatí poskytovateľom spracuje. Je však možné prijímať správy aj synchrónne, keď prijímateľ explicitne žiada o správu a spojenie je blokované pokiaľ správa nedorazí.



Obrázok 2.8: Programovací model JMS systému.

Programovací model JMS pozostáva z nasledujúcich súčastí:

- *Administratívne spravované objekty* – ciele a továrne spojenia sú objekty, ktoré sú lepšie spravované administrátorsky a nie programovo pretože technológia, na ktorej sú založené je podstatne odlišná od technológie implementujúcej zvyšok JMS systému. Tieto objekty sú sprístupnené pomocou prenosných rozhraní. Továreň spojenia je objekt zabezpečujúci vytvorenie spojenia s poskytovateľom správ, cieľ je objekt popisujúci zdroj správ ak prijímame a cieľ správ ak odosielame. Ciele sú v point-to-point systémoch fronty a v publish/subscribe systémoch takzvané témy (fóra).
- *Spojenie* – obaľuje virtuálne spojenie s JMS poskytovateľom. Môže byť reprezentované otvoreným TCP / IP socketom a využíva sa pre vytvorenie sedenia (session).
- *Sedenie (Session)* – je jednovláknový kontext využívaný na vytváranie a prijímanie správ.
- *Producent správ* – je objekt vytváraný sedením a slúžiaci na zasielanie správ cieľu.
- *Konzument správ* – je objekt tak isto vytváraný sedením a slúžiaci na prijímanie správ. Je možné vytvoriť konzumenta pre cieľový objekt, frontu alebo tému. Pre objekt konzumenta môžme zaregistrovať message listener, v ktorého *onMessage* metóde je špecifikované, čo sa má stať po prijatí správy.

JMS správy dodržujú základný formát, ktorý sa zhoduje s formátom správ používaných v aplikáciách, ktoré neimplementujú JMS API. Správa je zložená z troch základných častí a to hlavička, vlastnosti (properties) a telo, z ktorých len hlavička je povinná. Sú v nej špecifikované

informácie napríklad o cieli správy, ID správy, čas expirácie, a pod. Vo vlastnostiach správy je možné nadefinovať špeciálne informácie, pre ktoré v hlavičke neexistujú preddefinované polia. Telo správy obsahuje samotné prenášané dáta a JMS definuje päť typov správ podľa typu dát. Sú nimi textová, mapová, bytová, prúdová (stream) a objektová správa. Podkapitola prebraná z [1].

2.2.4.2 JavaPersistence API

JavaPersistence API – JPA je ORM nástroj (object-relational mapping – nástroj pre mapovanie objektov na relačné databázy), ktorý slúži na správu a prístup k dátam z Java aplikácií. Obrazom relačných tabuliek v Java EE aplikácii sú entity. Jedná sa o objekty, ktoré obsahujú buď perzistentné polia alebo atribúty, sú anotované *javax.persistence.Entity* anotáciou a dodržiavajú JavaBeans špecifikáciu (private alebo protected atribúty a prístup k nim len cez get/set metódy). Entity musia mať public bezparamterický konštruktor a nesmú byť final.

Čo sa týka typov atribútov, JPA podporuje všetky primitívne a ich obaľovacie objekové Java typy ako aj String, BigDecimal, Date, atď. Každý atribút alebo jeho get metóda, ktoré nie sú označené kľúčovým slovom *transient* alebo anotáciou *javax.persistence.Transient* je zperzistentnený do databázového priestoru a predstavuje stĺpec v tabuľke. Názov stĺpca je automaticky odvodený z mena atribútu, môžme však použiť anotáciu *javax.persistence.Column* pre špecifikovanie jeho vlastností (napríklad meno).

Každá entita musí byť jednoznačne identifikovateľná pomocou primárneho kľúča. Primárny kľúč môže byť primitívny typ, jeho objektová obaľovacia trieda, String alebo dátum (java.util.Date/java.sql.Date) a jeho get metóda alebo jeho deklarácia musí byť anotovaná *javax.persistence.Id* anotáciou.

JPA podporuje kolekcie, pre ktorých implementáciu je možné použiť jedno z rozhraní java.util.List, java.util.Set, java.util.Map alebo java.util.Collection. Vzťahy medzi entitami sú buď jednosmerné (len jedna strana má odkaz na druhú) alebo obojsmerné (obe strany majú odkaz na opačnú). Mnohopočetné vzťahy je možné vyjadriť štyrmi spôsobmi.

- *One-to-one* – je prípad, že každá inštancia entity je prepojená s práve jednou inštanciou inej entity. Na mapovanie danej relácie používame *javax.persistence.OneToOne* anotáciu.
- *One-to-many* – je prípad, keď je inštancia entity prepojená s viacerými inštanciami inej entity. V prípade obojsmernej relácie je najčastejším scenárom, že strana *many* drží cudzí kľúč na stranu *one*, v prípade jednosmernej relácie je potrebné použiť asociačnú tabuľku [15]. Kolekciu označujeme anotáciou *javax.persistence.OneToMany*.
- *Many-to-one* – je opakom ku one-to-many. V prípade jednosmernej (z pohľadu many) aj obojsmernej relácie je najčastejšie použitý cudzí kľúč na strane *many* [15].

- *Many-to-many* – je prípad, keď sú viaceré inštancie jednej entity prepojené s viacerými inštanciami druhej entity a na jeho reprezentáciu je vždy nutné použiť asociačnú tabuľku [15].

Pri mapovaní dedičnosti poskytuje JPA viacero možností. Prvou je použitie takzvanej *MappedSuperclass*, ktorá nie je braná ako entita (nie je dekorovaná ako @Entity). Princípom tohto typu mapovania je, že všetky tabuľky, vytvorené z entít dediacich od tohto predka, budú obsahovať aj stĺpce atribútov rodičovskej triedy. Ďalším spôsobom je použitie klasickej rodičovskej triedy, ktorá je už entitiou a zvolením jednej z troch možných stratégií mapovania dedičnosti:

- *SINGLE_TABLE* – je stratégia, pri ktorej sú všetky triedy dediace od rodičovskej triedy mapované do jedinej tabuľky. Jednotlivé inštancie sú rozlíšené za pomoci takzvaného *discriminator column* (rozlišovací stĺpec), ktorý obsahuje špecifické hodnoty identifikujúce jednotlivé triedy.
- *TABLE_PER_CLASS* – pri tejto stratégii, je každá dediaca trieda mapovaná do vlastnej tabuľky spolu s atribútmi zdedenými z rodičovskej triedy.
- *JOINED* – v tomto prípade sú rodičovská trieda tak ako aj jej dedičné triedy reprezentované vlastnými tabuľkami a obsahujú primárne kľúče, ktoré sú zároveň cudzími kľúčmi do rodičovskej tabuľky.

Pre správu entít a prístup k databázovým tabuľkám definuje JPA rozhranie *javax.persistence.EntityManager* avšak veľmi rozšírené je podľa [9] aj použitie DAO. Tieto dva prístupy sa však nevylučujú.

JPA poskytuje pre vytváranie SQL výrazov svoj vlastný výrazový jazyk (query language - EJBQL). Tento jazyk je založený na SQL a je v ňom možné použiť všetky SQL klauzuly. Medzi jeho výhody patrí napríklad používanie pomenovaných parametrov, ktoré sa vo výraze vyskytujú v tvare „*:meno*“, kde meno je názov identifikátora. Hodnota tohto identifikátora sa nastaví po zostavení výrazu metódou *setParameter*.

2.2.4.3 JavaTransaction API

JavaTransaction API (JTA), je technológia zabezpečujúca integritu dát pri databázových operáciách a zabraňuje aktualizácii dát viacerými aplikáciami súčasne. Transakcia je postupnosť udalostí, ktoré musia prebehnúť všetky alebo žiadna z nich. Z toho vyplýva, že transakcia môže byť ukončená v prípade úspechu odoslaním dát (*commit*) alebo vrátením systému do stavu pred transakciou v prípade neúspechu (*rollback*). Pri rollback-u je zachovaná integrita dát a tie nie sú kompromitované čiastočnými dátami z nedokončenej transakcie.

V Java EE, konkrétne pri použití EJB technológie rozlišujeme dva druhy transakcií, riadené kontajnerom alebo riadené aplikačne. V prvom prípade sa kontajner stará o vyvolanie aj ukončenie transakcie a definuje jej oblasť. Oblasť transakcie je definovaná z dôvodu, aby bolo možné jednoznačne určiť, či metóda vyvolávaná z metódy bežiacej v transakcii bude patriť do tej istej transakcie, alebo bude pre ňu vytvorené nová. Oblasť transakcie, alebo inak povedané atribúty môžu byť nasledujúce:

- *Required – vyžadovaný*, vyvolávaná metóda bude bežať v tej istej transakcii.
- *RequiresNew – vyžaduje novú*, transakcia je pri volaní metódy pozastavená, pre vyvolávanú metódu je vytvorená nová transakcia, po ktorej ukončení je dokončená pôvodná.
- *Mandatory – povinný*, metóda vyvolávaná z prvej metódy beží v tej istej transakcii, ak prvá metóda nebeží v transakcii, je vyvolaná výnimka.
- *NotSupported – nepodporovaný*, podobný prípad ako RequiresNew, s tým rozdielom, že ak prvá metóda beží v transakcii táto transakcia sa pozastaví a druhá metóda beží mimo transakciu. Ak prvá metóda nebeží v transakcii, nie je vytvorená nová transakcia pre druhú metódu.
- *Supports – podporovaný*, ak prvá metóda beží v transakcii, z nej vyvolaná metóda beží v tej istej transakcii. Ak prvá metóda nebeží v transakcii nie je vytvorená transakcia ani pre druhú metódu.
- *Never – nikdy*. Ak metóda beží v transakcii a vyvolá inú metódu, je vyhodená výnimka, ak nebeží v transakcii, pre druhú metódu nie je vytvorená transakcia.

Pri aplikačne riadených transakciách je oblasť transakcie explicitne definovaná vývojárom. V tomto prípade je nutné použiť viac zdrojového kódu, avšak tento spôsob poskytuje jednu výhodu oproti kontajnerom riadeným transakciám, a to možnosť riadiť transakcie v rámci business metód. Je možné špecifikovať rôzne chovanie transakcie pri rôznych podmienkach. Kapitola prebraná z [1].

3 Postup inštalácie a príprava vývojového prostredia

Aby bolo možné vytvoriť aplikáciu na platforme Java EE, musia byť na danom počítači nainštalované vývojové komponenty, bez ktorých by bol vývoj značne obtiažny ba priam nemožný. Na priloženom optickom médiu sa nachádzajú plne nakonfigurované vývojové prostriedky použité v príkladoch, pričom táto kapitola je návodom ako pripraviť vývojové prostredie a prostredie počítača v prípade, že daný optický disk nebude dostupný. V tejto práci sú používané najnovšie technológie dostupné k danému dátumu [cit. 2010-03-25], pričom nie je zaručené, že návod bude funkčný aj pre iné verzie. Ukážkové projekty boli vyvíjané na operačnom systéme Windows XP, preto je aj tento návod orientovaný pre tento operačný systém.

3.1 Nastavenie vývojového prostredia

Najdôležitejšou komponentou pri vývoji na platforme Java EE je samotná platforma. Táto práca využíva platformu Java EE 5 SDK Update 8 (so vstavaným JDK 6 Update 18). Platforma je voľne dostupná na oficiálnych stránkach firmy SUN Microsystems (http://java.sun.com/javase/downloads/index.jsp), pričom je potrebné vybrať distribúciu obsahujúcu Java EE a nainštalovať ju. Ďalším krokom je stiahnutie vývojového prostredia. Príklady v tejto práci boli vyvíjané v IDE Eclipse 3.5 Galileo, ktoré môžeme stiahnuť z oficiálnych stránok projektu Eclipse IDE (http://www.eclipse.org/downloads). Pre potreby tejto práce je nutné zvoliť distribúciu Eclipse IDE for JavaEE Developers. Eclipse IDE nie je potrebné inštalovať, po stiahnutí .zip archívu ho stačí rozbaliť a môžeme začať s vývojom.

Pri prvom spustení sa nás Eclipse opýta na umiestnenie Workspace. Jedná sa o pracovný adresár, v ktorom sa budú nachádzať všetky zložky projektu a metadáta, ktoré potrebuje Eclipse pre svoj beh. Po zvolení adresára môžeme označiť možnosť aby sa primárne používal tento workspace a Eclipse sa na jeho umiestnenie už nebude pýtať pri ďalšom spustení.

3.2 Java EE server

Tak ako všetky webové aplikácie aj Java EE aplikácie bežia na špecializovaných serveroch. Existuje mnoho Java EE serverov od rôznych producentov, pre použite v tejto práci som sa aj na

základe [18] rozhodol pre JBoss aplikačný server verzie 5.1.0. Ďalšími alternatívami ku JBoss môžu byť napríklad Apache Geronimo, WebLogic od BEA Systems (od roku 2008 už Oracle), alebo WebSphere od IBM. Pri výbere Java EE serveru musíme prihliadať na viacero kritérií. Podľa [18] je to v prvom rade plná podpora pre platformu Java EE, čo znamená, že daný aplikačný server by mal obsahovať implementácie základných technológií Java EE špecifikácie, medzi nimi hlavne podpora pre EJB 3.0 (webový kontajner Tomcat ju napríklad sám o sebe neobsahuje), podpora JSP a Servletov, v neposledom rade podpora JSF a mnoho iných.

JBoss aplikačný server je možné stiahnuť v .zip archíve z http://www.jboss.org/jbossas/downloads/ (v našom prípade verzia 5.1.0). Následne je potrebné balík rozbaliť. Pri výbere umiestnenia JBoss musíme dbať na to, aby cesta k serveru neobsahovala žiadne medzery (nesmie to byť napríklad C:\Program Files\Jboss\...) pretože by dochádzalo ku problémom s deployom aplikácií na server. Pred integrovaním JBoss do Eclipse je potrebné manuálne upraviť ešte chybu v konfigurácii servra. Podľa [21] dochádza ku drobnej nekompatibilite JDK 1.6.0_18 a JBoss 5.1.0 a je potrebné ručne zeditovať súbory:

- \jboss-5.1.0.GA\server\all\conf\bootstrap\profile.xml
- \jboss-5.1.0.GA\server\default\conf\bootstrap\profile.xml
- \jboss-5.1.0.GA\server\minimal\conf\bootstrap\profile.xml
- \jboss-5.1.0.GA\server\standard\conf\bootstrap\profile.xml
- \jboss-5.1.0.GA\server\web\conf\bootstrap\profile.xml

V definícii beanu AttachmentStore je potrebné špecifikovať triedu parametra konštruktora, čiže zmeniť zápis

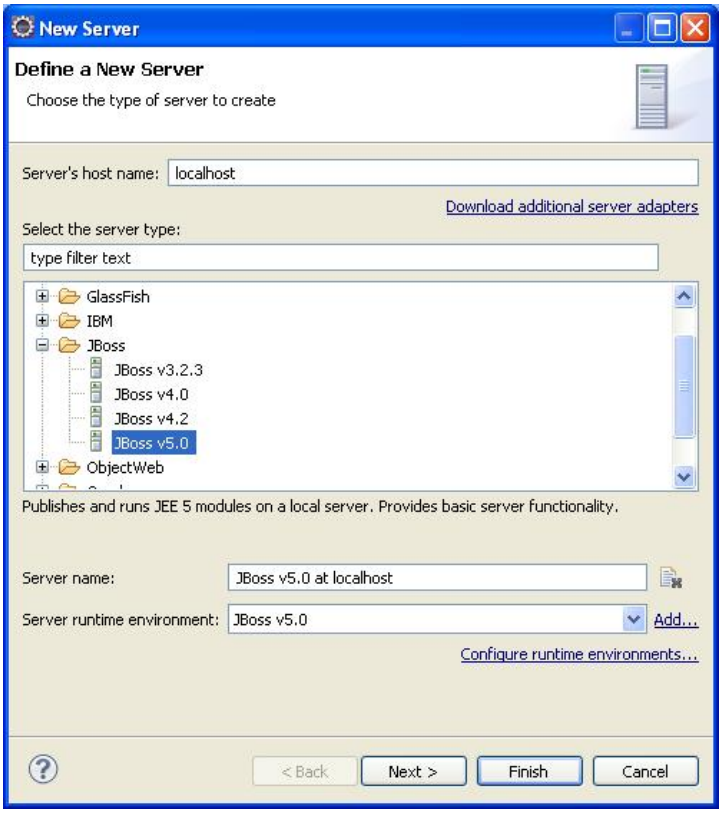
```
<constructor><parameter><inject bean="BootstrapProfileFactory"... na
<constructor><parameter class="java.io.File"><inject bean="...
```

Teraz môžeme integrovať aplikačný server do Eclipse. Klikneme pravým tlačidlom v dolnej časti užívateľského rozhrania (GUI) Eclipse do záložky Servers, zvolíme New server, kde v menu vyberieme našu inštaláciu JBoss (Obrázok 3.1). Môžeme špecifikovať konfiguráciu servera tým, že klikneme na odkaz "*Configure runtime enviroments…*" v spodnej časti okna, v prehľade konfigurácií serverov vyberieme JBoss v5.0, klikneme na "*Edit*" a v novo-otvorenom okne nastavíme JRE na nami nainštalované jdk1.6.0_18 (Obrázok 3.2). Ak sa toto v zozname nenachádza, tak klikneme na „*Installed JRE preferences*", pridáme novú Standard VM a v nasledujúcom okne zvolíme do JRE Home inštalačný adresár JDK. Naspäť na obrazovke, kde sme vyberali JRE zvolíme práve pridané JDK zo zoznamu a v poli pod ním zvolíme inštalačný adresár JBoss servera. Potvrdíme a na ďalšej obrazovke (Obrázok 3.3) je možné konfigurovať porty a adresy, na ktorých server načúva (doporučujem ponechať tak ako je)

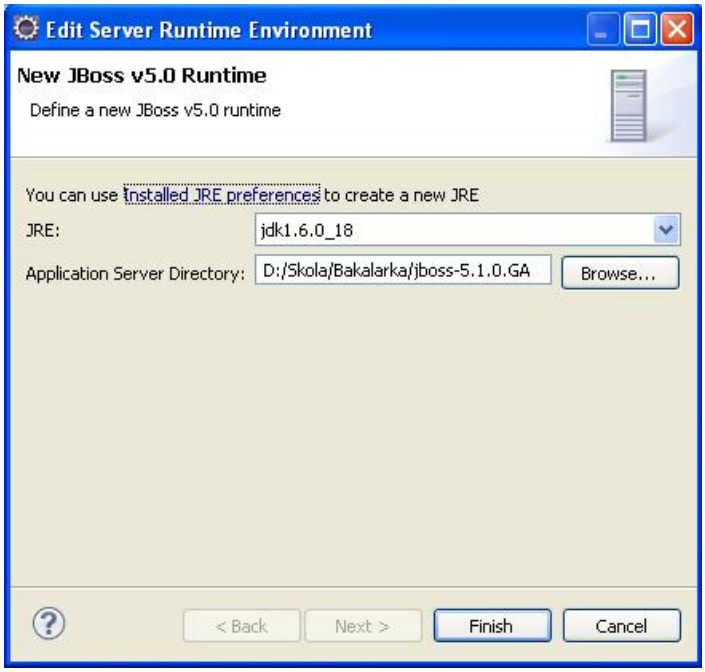
a klikneme Finish. Po uzavretí sprievodcu by sa nám v dolnej časti GUI v Eclipse mal zobraziť náš JBoss server. Podľa [19] je doporučené zvýšiť pamäť servera aj Eclipse aby sme predišli „java.lang.OutOfMemmoryError“ chybám. Toto nastavenie musíme prispôsobiť veľkosti fyzickej pamäti v počítači, na ktorom inštalujeme vývojové prostredie. Pamäť Eclipse zvýšime v súbore eclipse.ini v koreňovom adresári Eclipse IDE. Súbor môžeme modifkovať napríklad nasledovne (každá z premenných musí byť na samostatnom riadku):

- *-vmargs* (všetky parametre za týmto riadkom sú predávané priamo do virtuálneho stroja - JVM)
- *-Xms512m* (počiatočná hodnota Java heap – veľkosť pamäti JVM)
- *-Xmx768m* (maximálna hodnota Java heap – veľkosť pamäti JVM)
- *-XX:MaxPermSize=512m* (maximálna veľkosť pamäti, v ktorej sú držané objekty a triedy samotnej JVM [23])

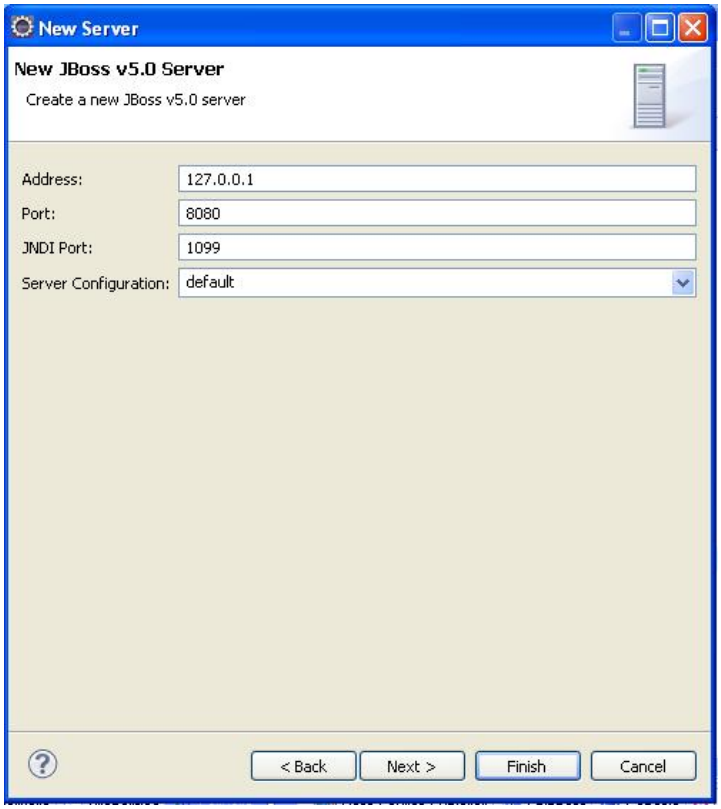
Pamäť serveru zvýšime tým, že klikneme 2x na server v záložke Servers dole v GUI, na obrazovke, ktorá sa nám otvorí, klikneme na odkaz „*Open launch configuration*“ a v záložke „*Arguments*“ pridáme do poľa VM arguments napríklad: „*-Xms512m –Xmx768m -XX:MaxPermSize=256m*“. Ďalej je vhodné zvýšiť čas spúšťania a zastavovania servera na obrazovke nastavenia servera v záložke „*Timeouts*“.



Obrázok 3.1: Definícia nového JBoss servera



Obrázok 3.2: Nastavenie behového prostredia nového JBoss servera



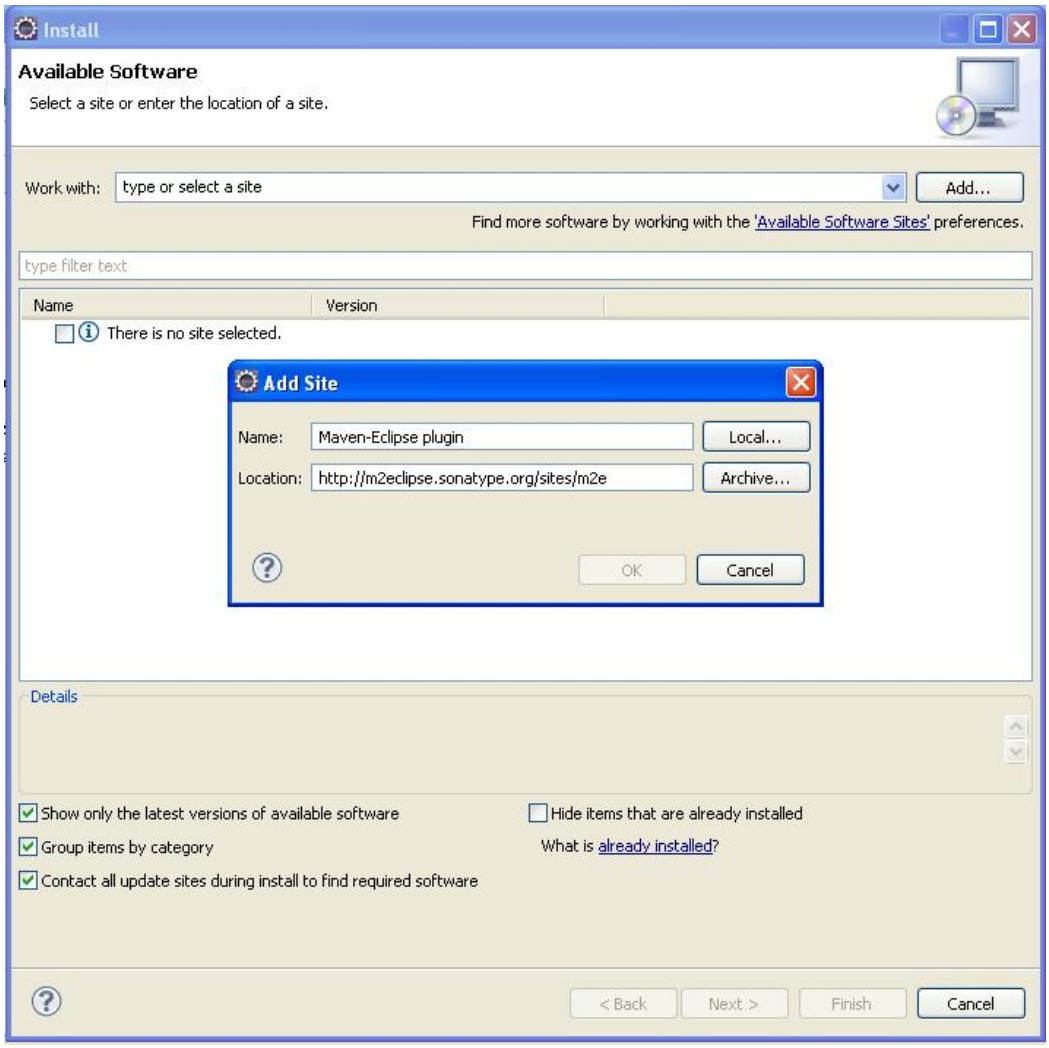
Obrázok 3.3: Konfigurácia portov a adresy nového JBoss servera

3.3 Nástroj pre zostavenie aplikácie

V tejto práci som sa rozhodol použiť nástroj Apache Maven 2.2.1, ktorý je na zásuvných moduloch (pluginoch) založená komponenta pre zostavovanie a manažment Java aplikácií [20]. Maven je možné voľne stiahnuť z http://maven.apache.org/download.html vo formáte .zip a jednoducho rozbaliť do cieľového adresára. Následne je potrebné nastaviť užívateľské premenné prostredia. Premennú M2_HOME, ktorá ukazuje do domovského adresára našej inštalácie Maven 2 a premennú M2, ktorá ukazuje do adresára Maven/bin. Do užívateľskej premennej PATH je potrebné doplniť premennú %M2%; aby bolo možné spúšťať mavenovské príkazy z príkazového riadku.

Aby sme mohli využiť zostavovaciu funkcionalitu Maven-u v Eclipse, musíme nainštalovať Eclipse-Maven plugin. Eclipse IDE poskytuje jednoduchý nástroj pre aktualizácie existujúceho a inštaláciu nového softwaru. Do menu inštalácie nového softwaru sa dostaneme cez menu Help -> Install New Software. V tomto menu pri vstupnom poli „Work with“ klikneme na Add a v okne, ktoré sa nám otvorilo zvolíme názov stránky (napr. Eclipse-Maven

plugin) a do Location zadáme http://m2eclipse.sonatype.org/sites/m2e (Obrázok 3.4). Potom vyčkáme kým sa načítajú informácie zo stránky a zvolíme Maven Integration For Eclipse a nainštalujeme.



Obrázok 3.4: Pridanie m2eclipse update site pre inštaláciu M2 plugin

4 Pridanie podpory pre technológie založené na Java EE a popis vývoja ukážkových aplikácií

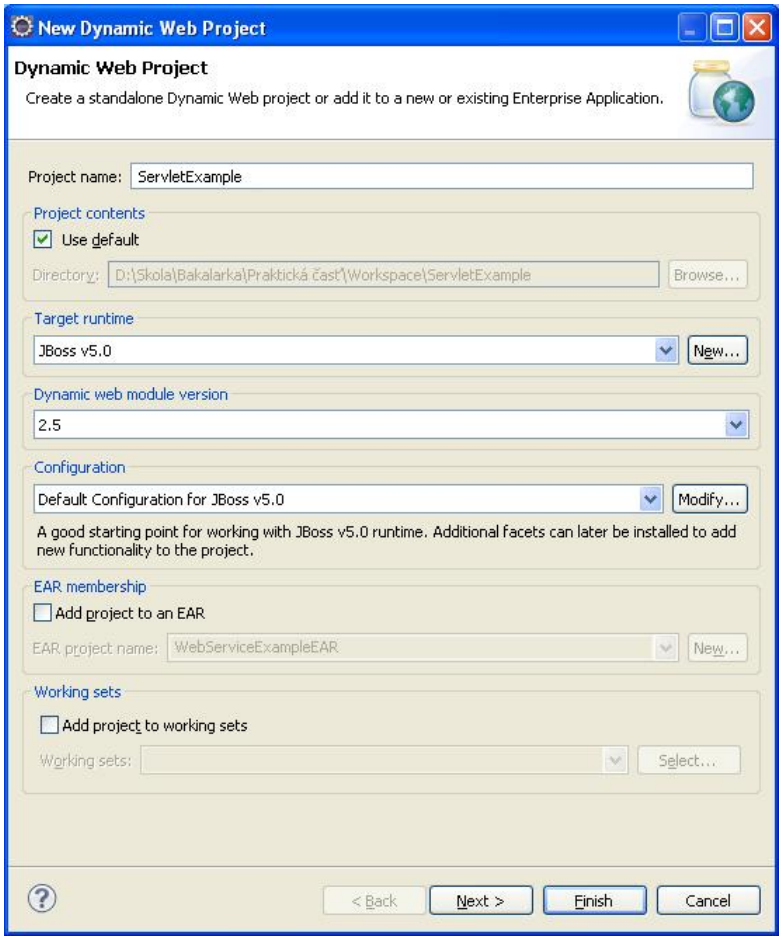
Táto kapitola popisuje ako vytvoriť projekty demonštrujúce základné technológie platformy Java EE a ako pridať podporu pre technológie, ktoré nie sú zahrnuté v špecifikácii Java EE.

4.1 Ukážková aplikácia demonštrujúca technológiu JavaServlet

Tento príklad demonštruje použite technológie JavaServlet a popisuje základný cyklus prijatia a spracovania HTTP požiadavky ako aj jej prechod filtrami. Jedná sa o úplne základné použitie tejto technológie kde je výsledný HTML kód generovaný manuálne do odpovede pre klienta. Projekt je pripravený vo workspace na priloženom optickom disku, jeho názov je ServletExample a zdrojové súbory sa nachádzajú v balíku cz.vutbr.fit.*.

Pri vývoji tohto projektu bolo potrebné prejsť nasledujúcimi fázami:

- V menu v Eclipse zvolíme File -> New -> Dynamic Web Project, na úvodnej obrazovke zvolíme meno projektu, target runtime vyberieme náš JBoss server (Obrázok 4.1), nasledujúcu obrazovku necháme tak ako je a na tretej obrazovke zvolíme context root, čo je názov pod ktorým budeme pristupovať ku aplikácii cez URL (napr: http://localhost:8080/ServletExample, kde ServletExample je context root), necháme zaškrtnuté „Generate web.xml deplyment descriptor“ a dáme Finish. Predchádzajúci postup nám vygeneruje adresárovú štruktúru projektu a deployment descriptor web.xml, ktorý sa nachádza v adresári WebContent/WEB-INF/.



Obrázok 4.1: Vytovrenie nového Dynamic Web Project

- Pre automatické zostavovanie aplikácie za pomoci Maven musíme zapnúť na projekte maven dependency management (maven správu závislostí). Klikneme pravým tlačidlom na projekt v Project Exploreri, a v záložke Maven zvolíme Enable Dependency Management. Zobrazí sa nám okno, v ktorom je možné špecifkovať základné atribúty pom.xml, čo je súbor, v ktorom udávame ako má Maven zostavovať aplikáciu. Po tom ako zvolíme meno, popis projektu (nie je povinné) a nastavíme atribút „*packaging*“ na hodnotu „*war*“, klikneme Next a na nasledujúcej obrazovke môžme definovať maven závislosti (externé knižnice, ktoré by mali byť pridané do classpath), čo však zatiaľ nepotrebujeme. Po dokončení sprievodcu vidíme, že nám Maven vygeneroval pom.xml a ikona projektu v Project Exploreri sa označila v ľavom hornom rohu modrým písmenom „M“, čo znamená, že sa o jeho zostavovanie teraz stará Maven, ktorý má na starosti aj všetky externé závislosti a knižnice. To znamená, že ak by sme potrebovali

využiť v projekte knižnicu tretej strany, nemusíme ju ručne sťahovať a includovať do classpath, ale definujeme ju v pom.xml a maven si ju automaticky stiahne do svojho lokálneho repozitára a prepojí ju s projektom. Maven vytvára svoj lokálny repozitár pod Windows v adresári C:\Documents And Settings\<meno užívateľa>\.m2\repository. Klikneme pravým tlačidlom myši na projekt v Project Exploreri, v menu Maven klikneme na záložku Add plugin a vo vyhľadávacom poli napíšeme „maven-war-plugin“. Maven nám ho vyhľadá v domovskej online repository a po kliknutí na Ok, pridá jeho špecifikáciu do pom.xml. Našou úlohou je teraz špecifikovať tomuto pluginu, kde sa nachádza deployment descriptor pre našu aplikáciu. Jeho konfiguráciu upravíme nasledovne:

```
<configuration>
      <webResources>
            <resource>
                  <directory>
                        WebContent
                  </directory>
                  <includes>
                        <include>WEB-INF/web.xml</include>
                  </includes>
            </resource>
      </webResources>
</configuration>
```

Ak chceme zbaliť aplikáciu do archívu klikneme pravým tlačidlom myši na projekt, v menu „Run as“ zvolíme „Maven package“ a Maven nám zabalí preložené .class súbory z adresára target/classes a všetky externé zdroje definované v tagu `<webResources></webResources>` do .war archívu, ktorý môžeme pripojiť do webového kontajnera.

- Po nastavení Maven správy závislostí musíme ešte špecifikovať Java Build Path, z toho dôvodu, že Maven odstránil adresár src z classpath.

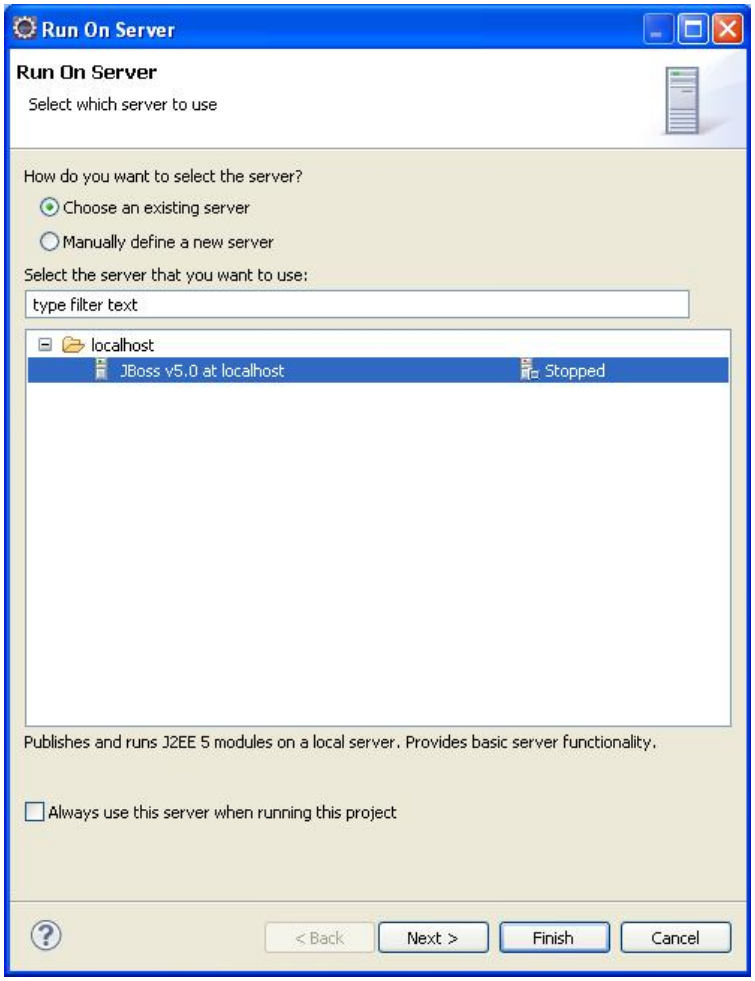
- V balíku src.java vytvoríme balíky a zdrojové súbory. Následne musíme dané servlety a filtre namapovať vo web.xml tak ako je to demonštrované v ukážkovej aplikácii. Pri mapovaní fitrov musíme mať na pamäti, že filtre sú v reťazi filtrov prechádzané v takom poradí, ako sú špecifikované vo web.xml. Pri vytváraní zdrojových súborov je vhodné používať na platforme nezávislé kódovanie (napr. UTF-8). Pre zmenu kódovania klikneme pravým tlačidlom myši na projekt a v Properties v záložke Resource zvolíme nami požadované kódovanie v poli Text file encoding.

- Aplikácia je po každom uložení zostavovaná automaticky, ak túto funkcionalitu nevypneme v záložke Project -> Build Automatically v hornom menu v Eclipse. Po dokončení zdrojových súborov klikneme pravým tlačidlom myši na projekt, v menu Run As zvolíme Run on server (Obrázok 4.2), vyberieme náš JBoss server a ak sa na nasledujúcej obrazovke nenachádza náš projekt vpravo tak ho tam presunieme. Klikneme na Finish a po chvíli sa server naštartuje. Po nabehnutí sa k našej aplikácii dostaneme cez URL http://localhost:8080/ServletExample/ , kde /ServletExample je context path našej aplikácie a „/“ je mapovanie servletu, ktorý obsluhuje danú URL. Toto mapovanie sme špecifikovali vo web.xml. Pri opätovnom spustení serveru stačí kliknúť na daný server pravým tlačidlom a zvoliť Run alebo Publish. Obe možnosti spustia server a nasadia naň našu aplikáciu.

- V tomto prípade je naša aplikácia jednoduchý webový projekt, ktorý je nasadený vo vstavanom webovom kontajneri JBoss serveru. Implementácia tohoto kontajnera je podľa [22] založená na webovom kontajneri Apache Tomcat. Webové kontajnery Tomcat poskytujú vlastnosť, ktorá sa nazýva Hot Code Replace, čiže nahradzovanie kódu za behu bez potreby reštartu a redeploy aplikácie. Ak meníme jednoduchý kód (zmeny v tele metód) tak sa tento kód po chvíli nahradí na serveri, čo nám signalizuje hláška v pravom dolnom rohu Eclipse „Publishing to JBoss on localhost“ a môžeme okamžite vidieť zmeny.

- Aplikáciu je možné debugovať (ladiť) za behu. Stačí spustiť JBoss v debug móde (kliknúť pravým tlačidlom myši na server a zvoliť možnosť Debug). Ak spúšťame server v debug móde môžeme pridať breakpoint na ľubovoľný riadok v zdrojovom súbore a ako náhle naň beh programu narazí, aplikácia sa zastaví a my môžeme zvyšok zdrojového kódu krokovať.

Vidíme, že generovanie odpovede priamo pre klienta v rámci Java kódu v servlete je pracné a náchylné na chyby. Preto by ďalším rozšírením takejto jednoduchej aplikácie mohlo byť pridanie jsp stránok, ktoré by obsahovali statické HTML elementy a servlet by pre nich generoval len ten obsah, ktorý sa mení a tento obsah by v obaľovacom objekte predával do jsp, čiže by sa jednalo o aplikáciu rámca MVC. Pri použití filtrov vidíme, že je zložité dostať sa ku samotnému obsahu odpovede (HTML kódu) a upravovať ho. Aj preto je typickou úlohou filtrov napríklad zmena hlavičky odpovede, autentifikácia a autorizácia užívateľov a podobne a nie modifikácia jej samotného obsahu.



Obrázok 4.2: Pripojenie aplikácie na server

4.2 Ukážková aplikácia demonštrujúca typické použitie filtrov

Táto aplikácia sa podobá predchádzajúcej, využíva rovnaké technológie, ale použitie filtrov je značne odlišné. V tomto prípade neupravujeme obsah odpovede, ale kontrolujeme, či užívateľ zadal správne meno do vstupného poľa vo formulári. Ako už bolo spomenuté, filtre sa často využívajú pre autorizáciu a autentifikáciu užívateľov. Jediný filter, ktorý sa nachádza v tejto aplikácii (AuthenticationFilter) kontroluje, či sa užívateľom zadané meno rovná menu, ktoré bolo vypísané na stránke a vložené do požiadavky pri odoslaní formulára za pomoci input hidden tagu. Ak áno, filter prepúšťa požiadavku ďalej do servletu, ak nie, požiadavka je

presmerovaná na chybovú stránku. Postup pri vytváraní aplikácie je totožný s postupom pri vytváraní predchádzajúcej.

- Vytvoríme nový Dynamic Web Project
- Zapneme na ňom Maven správu závislostí
- Upravíme Java Build Path
- Vytvoríme balíky a zdrojové súbory
- Ak už máme dole v záložke „Servers“ zobrazený náš JBoss server, klikneme naň pravým tlačidlom myši, zvolíme možnosť „Add“ a pridáme náš nový projekt zľava doprava. Ak sa tam server nenachádza postupujeme ako pri predchádzajúcej aplikácii.

Ukážková aplikácia beží na URL http://localhost:8080/ServletExample2.

4.3 Ukážková aplikácia demonštrujúca technológiu JavaServer Pages

Aj v tomto prípade sa bude jednať o Dynamic Web Project, takže postup je rovnaký ako u predchádzajúcich dvoch projektov. Jediný rozdiel je, že nebudeme volať servlet, ktorý bude generovať HTML pre klienta ale toto HTML špecifikujeme v jsp súbore index.jsp. Tento súbor umiestníme do adresára WebContent (pozor nie do WEB-INF, tieto adresáre sú vo väčšine Java EE serverov chránené). Pri prístupe na URL http://localhost:8080/JSPExample/index.jsp alebo http://localhost:8080/JSPExample sa nám zobrazí text s daného jsp súboru a aktuálny dátum a čas. Tento údaj je za každým refreshom stránky aktualizovaný, pretože sa získava za pomoci volania metódy podporného beanu (backing bean). Metóda podporného beanu, ktorú chceme vyvolávať z jsp stránky musí začínať prefixom *get-* a v našom prípade ju zavoláme pomocou volania podporného beanu a reťazca za týmto prefixom:

```
    <jsp:useBean id="backingBean" scope="application"
class="cz.vutbr.fit.beans.BackingBean"/>
    Práve teraz je: ${backingBean.formattedDate}
```

Názov metódy v Java triede je `public String getFormattedDate()`.

4.4 Ukážková aplikácia demonštrujúca technológiu JavaServerFaces

Opäť vytvoríme nový Dynamic Web Project a zapneme na ňom Maven správu závislostí. Pre správne fungovanie rámca JSF musíme vo web.xml definovať FacesServlet. Táto špecifikácia určí, že všetky jsp stránky namapované na tento servlet obsahujú JSF komponenty a majú byť podľa toho spracovávané. V adresári WebContent\WEB-INF vytvoríme súbor s názvom *faces-config.xml*, v ktorom sú okrem podporného beanu definované aj navigačné pravidlá medzi stránkami. V tomto súbore špecifikujeme aj umiestnenie .properties súboru, ktorý obsahuje informačné a chybové hlášky. Súčasťou aplikácie sú aj dve jsp stránky index.jsp a success.jsp. Prvá z nich obsahuje formulár, na ktorého polia sú naviazané vstavané JSF validátory, na druhú je užívateľ presmerovaný v prípade, že ním zadané dáta prešli validáciou. Aplikácia beží na URL http://localhost:8080/JSFExample/jsf/index.jsp.

4.5 Ukážková aplikácia demonštrujúca technológiu Enterprise JavaBeans

Súčasťou tejto aplikácie je projekt EJBExample, ktorý predstavuje Enterprise JavaBean (enterprise bean). Jedná sa o veľmi jednoduchú implementáciu, ktorá pri inštanciovaní triedy enterprise beanu v konštruktore vygeneruje náhodne číslo typu `Long`, ktoré predstavuje ID daného enterprise beanu. Táto trieda obsahuje metódu `public Long getId()`, ktorá vracia toto číslo. Klientom, ktorý volá tento enterprise bean je webová aplikácia EJBwebClientExample, ktorá je implementovaná pomocou technológie JSP. Táto aplikácia má za úlohu demonštrovať schoponosť EJB kontajnera obsluhovať veľké množstvo klientov súčasne. Pri zobrazení stránky index.jsp klientskej aplikácie, je zavolaná metóda podporného beanu, ktorá vytvorí a spustí desať vláken v jednom okamihu. Každé toto vlákno zavolá sto krát metódu `getId()` nášho ukážkového enterprise beanu a uloží vrátené ID do množiny spoločnej pre všetky vlákna. Výsledkom tejto operácie je výpis danej množiny do jsp stránky. V tomto výpise si môžeme všimnúť, že aj napriek tomu, že daný enterprise bean bol volaný dohromady tisíc krát bolo vytvorených podstatne menej jeho inštancií.

Pri vytváraní EJB projektu a webového klienta postupujeme nasledovne:

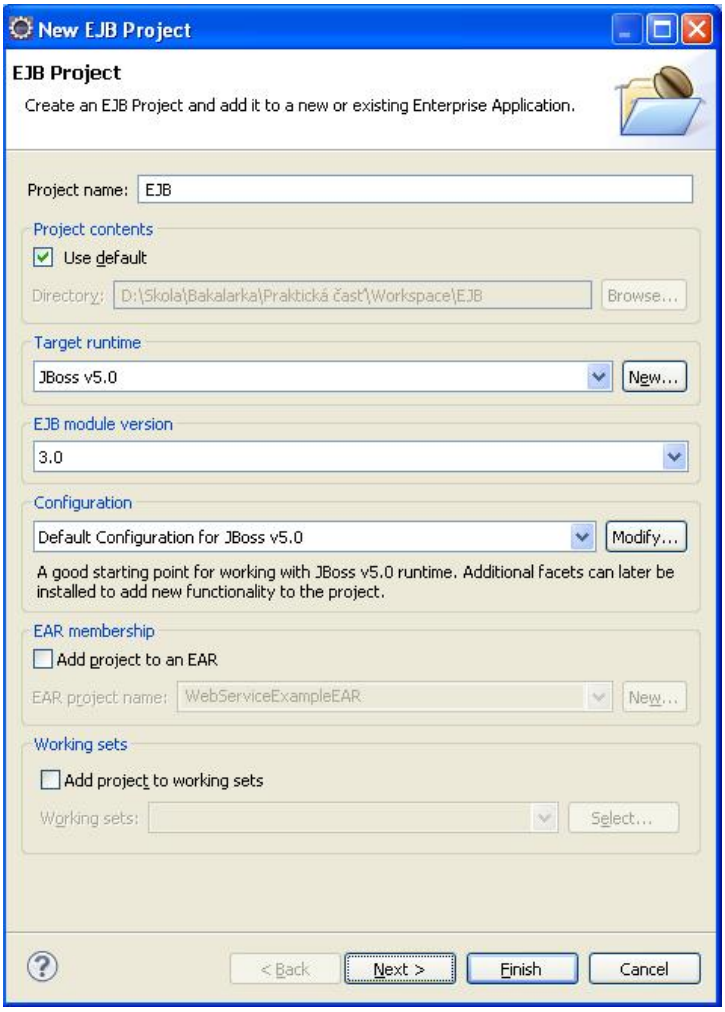
- V ľavej časti GUI v Project Exploreri klikenem pravým tlačidlom myši, zvolíme New->EJB Project. Na prvej obrazovke sprievodcu (Obrázok 4.3) zvolíme meno

projektu, *Target runtime* zvolíme JBoss v5.0, *EJB module version* ponecháme na verzii 3.0 a *Configuraton* necháme *Default configuration for JBoss v5.0*. Nasledujúcu obrazovku ponecháme tak ako je a na poslednej zaškrtneme *Generate ejb-jar.xml deplyment descriptor*.

- V zdrojovom balíku *ejbModule* vytvoríme triedy a balíky. Pri implementácii rozhrania enterprise beanu ho označíme anotáciou `@Remote` a implementujúcu triedu anotáciou `@Stateless(mappedName = "ExampleBean")`, čo namapuje náš enterprise bean pri deployi do EJB kontajnera na JNDI názov „ExampleBean“. Pod týmto názvom ho budeme za pomoci metódy `lookup()` vyhľadávať z webového klienta.
- Pri vytváraní webového klienta použijeme už niekoľko krát opakovaný postup. Nový Dynamic Web Project, vytvoríme podporný bean a jsp stránku. Ak by sme chceli zabaliť našu aplikáciu, môžeme zapnúť na projekte Maven správu závislostí.
- Vlákna implementujeme ako vnútornú triedu, ktorá implementuje rozhranie `Runnable`. Pri pridávaní ID enterprise beanov do množiny musíme dbať na to, že táto množina je zdieľaný zdroj, čiže prístup k nej musí byť synchronizovaný. Inštancie enterprise beanov získavame pomocou jednoduchej metódy (zobrazená nižšie), ktorá vyhľadá v JNDI strome náš EJB a prevedie dependency injection tohto enterprise beanu na náš lokálny odkaz.

```
private ExampleBean getExampleBean() {
      ExampleBean exampleBean = null;
      try {
            InitialContext ic = new InitialContext();
            exampleBean =
            (ExampleBean)ic.lookup("ExampleBean");
      } catch (NamingException e) {
            System.out.println("Nepodarilo sa nájsť
            ExampleBean!!!");
            e.printStackTrace();
      }
      return exampleBean;
}
```

Oba projekty musíme po dokončení pripojiť do nášho JBoss servera (kliknúť pravým tlačidlom na JBoss, v menu zvloiť Add a pridať oba projekty z ľavej časti okna do pravej). Webový klient využívajúci náš enterprise bean je dostupný na adrese http://localhost:8080/EJBwebClientExample/index.jsp.



Obrázok 4.3: Vytvorenie EJB projektu – základné nastavnia

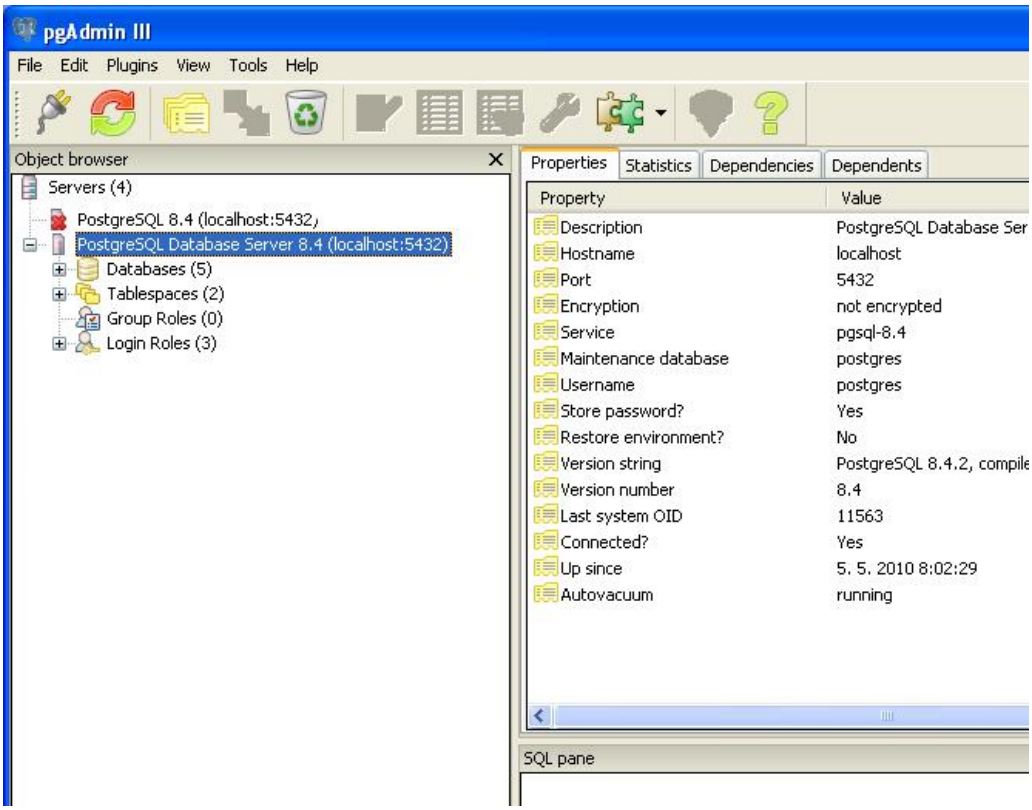
4.6 Ukážková aplikácia demonštrujúca technológiu Java Persistence API

Nevyhnutnou súčasťou aplikácie používajúcej technológiu JPA je databáza. Pre potreby tejto práce som zvolil na základe osobných kladných skúseností databázu PostgreSQL 8.4.3-1. Pre túto distribúciu som sa rozhodol pre jej jednoduchosť, rýchlosť a výborné administračné nástroje.

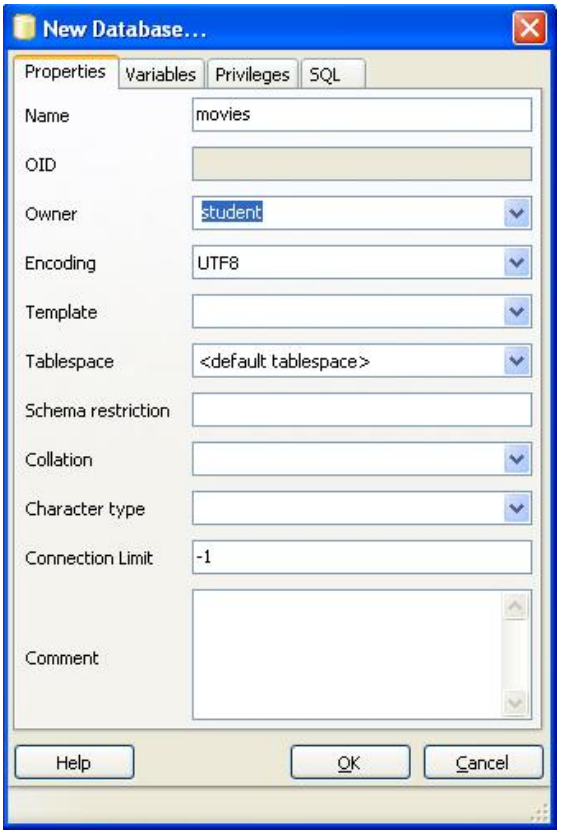
PostgreSQL stiahneme zo stránok http://www.enterprisedb.com/products/pgdownload.do a spustíme inštaláciu. PostgreSQL vytvorí pri inštalácii ďalšieho systémového užívateľa

s názvom „Postgres“, ktorý slúži ako superužívateľ, ktorý má všetky práva na všetky operácie pre celý databázový server.

Po ukončení inštalácie spustíme administračný nástroj pgAdmin III (Štart -> Všetky Programy -> PostgreSQL 8.4 -> pgAdmin III), kde v ľavom menu (Object browser) vidíme lokálny databázový server *PostgreSQL Database Server 8.4 (localhost:5432)* (Obrázok 4.4). Po dvojkliku na tento server a rozbalení ponuky klikneme pravým tlačidlom na *Login Roles* a vytvoríme nového užívateľa s menom a heslom „student“. Potom klikneme pravým tlačidlom na *Databases*, zvolíme *New Database* a vytvoríme novú databázu s názvom „movies“, kódovaním UTF-8 a vlastníkom „student“ (Obrázok 4.5). Vytvorenie databázovej schémy a vloženie dát prevedieme neskôr pomocou Maven zásuvných modulov.

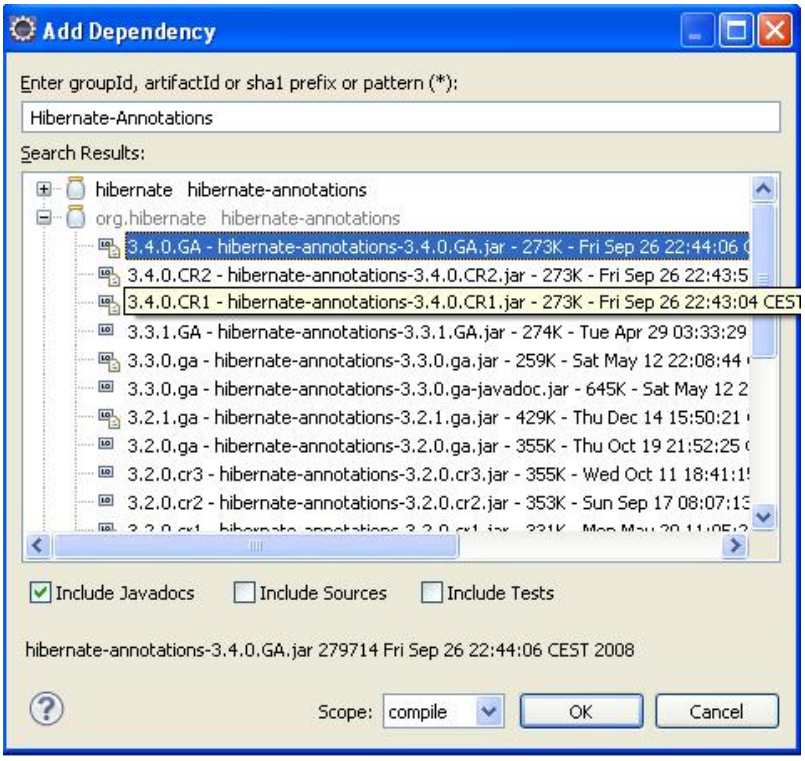


Obrázok 4.4: Administračný nástroj databázy PostgreSQL – pgAdmin III



Obrázok 4.5: Vytvorenie novej databázy v PostgreSQL

V Eclipse si opäť vytvoríme nový Dynamic Web Project a zapneme na ňom Maven správu závislostí. V tomto projekte som sa rozhodol použiť veľmi rozšírený ORM nástroj Hibernate, na ktorom si ukážeme ako spravovať závislosti pomocou Maven. V prvom rade musíme v našom projekte nadefinovať, ktoré externé knižnice budeme používať. Klikneme pravým tlačidlom myši na pom.xml, v menu Maven zvolíme Add dependency a do vstupného poľa napíšeme „javax.persistence“. Z možností, ktoré sa nám zobrazili vyberieme „*javax.persistence persistence-api*“ a potvrdíme. Vidíme, že nám maven vytvoril element <dependency> a vložil doň odkaz na nami požadovanú knižnicu. Tento postup zopakujeme s „*org.hibernate hibernate-annotations*“ (vyberieme verziu 3.4.0.GA – Obrázok 4.6) , „*org.hibernate hibernate-core*“ (verzia 3.3.2.GA) a nakoniec musíme definovať aj JDBC ovládač (driver) na PostgreSQL databázu „*postgresql postgresql*“ a zvolíme najnovšiu verziu - 8.4-701.jdbc4. Prvý názov (napr. *javax.persistence*), je takzvaný *groupId*, ktorý najčastejšie identifikuje producenta danej technológie a druhý (napr. *persistence-api*), je *artifactId*, ktorý identifikuje danú knižnicu.



Obrázok 4.6: Pridanie závislosti (Hibernate Annotations) do projektu za pomoci Maven

Po pridaní závislostí si vytvoríme v pom.xml nový profil na znovuvytvorenie databázy. Bude slúžiť na to, aby sa pri každom spustení maven tasku (napríklad package) nespúšťalo pregenerovanie databázovej schémy a naplnenie testovacími dátami. Na základe informácií z [24] a [25] vytvoríme za elementom `<dependencies>` nasledujúcu definíciu profilu a špecifikáciu zásuvných modulov:

```
<profiles>
  <profile>
    <id>recreateDB</id>
      <build>
        <plugins>
          <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>dbunit-maven-plugin</artifactId>
            <version>1.0-beta-3</version>
            <dependencies>
              <dependency>
                <groupId>postgresql</groupId>
                <artifactId>postgresql</artifactId>
                <version>8.4-701.jdbc4</version>
              </dependency>
            </dependencies>
            <configuration>
              <url>jdbc:postgresql://localhost:5432/movies</url>
              <driver>org.postgresql.Driver</driver>
```

```
            <username>student</username>
            <password>student</password>
            <type>INSERT</type>
            <src>src/database/insert.xml</src>
          </configuration>
        </plugin>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>sql-maven-plugin</artifactId>
          <version>1.2</version>
          <dependencies>
            <dependency>
              <groupId>postgresql</groupId>
              <artifactId>postgresql</artifactId>
              <version>8.4-701.jdbc4</version>
            </dependency>
          </dependencies>
          <configuration>
            <url>jdbc:postgresql://localhost:5432/movies</url>
            <driver>org.postgresql.Driver</driver>
            <username>student</username>
            <password>student</password>
            <onError>continue</onError>
            <autocommit>true</autocommit>
            <srcFiles>
              <srcFile>src/database/schema.sql</srcFile>
            </srcFiles>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

Vidíme, že oba zásuvné moduly sú závislé na JDBC ovládači a obom sme špecifikovali pripájacie údaje k našej databáze. Sql-maven-plugin je zásuvný modul, ktorý podľa [25] vykonáva SQL príkazy, či už priamo alebo zo súborov. My túto funkcionalitu využijeme na vytvorenie databázovej schémy z pripraveného súboru \src\database\schema.sql. Dbunit-maven-plugin slúži podľa [24] na prevádzanie databázových operácií ako INSERT, EXPORT a podobne pričom vytvára alebo používa dátové zdroje v dbunit formáte. Jedná sa o XML dokumenty, kde sú dáta popísané veľmi prehľadne a umožňujú tak ich jednoduchú editáciu. My využijeme task „*dbunit:operation*“, ktorý prevedie vloženie testovacích dát do databázy. Súbor s danými dátami je \src\database\insert.xml. Teraz môžme spustiť Maven task, ktorý nám z daného SQL súboru vygeneruje schému a vloží do databázy testovacie dáta. V root adresári našej aplikácie (tam kde máme pom.xml), otvoríme príkazový riadok a spustíme príkaz: „*mvn –P recreateDB sql:execute dbunit:operation –Drecreate.db.skip=false*“. Parameter –P a názov za ním hovorí Mavenu, ktorý profil sa má použiť, „sql:execute“ a „dbunit:operation“

spúšťajú tasky prídavných modulov popísaných vyššie a parameter „–Drecreate.db.skip=false“ hovorí Mavenu aby pregeneroval databázu.

V projekte vytvoríme balíky, triedy a jsp stránky, označíme anotáciou entitné objekty a vytvoríme DAO vrstvu, ktorá sa bude starať o databázové operácie. V tomto projekte je už jasne pozorovateľná 3-vrstvová architektúra aplikácie. Prezentačnú vrstvu tvoria jsp stránky, ktoré volajú metódy podporného ManagerBean (predstavuje business vrstvu), ktorý získava dáta z DAO vrstvy, ktorá komunikuje za pomoci Hibernate s databázou.

V našom projekte je Hibernate konfigurovaný za pomoci anotácií. Neexistujú žiadne entitne špecifické XML konfiguračné súbory a súbor hibernate.cfg.xml obsahuje len definíciu pripojenia k databáze a mapovanie entitných objektov. Všetko ostatné (definícia primárnych a cudzích kľúčov, stĺpcov v tabuľkách, sekvencií ID a podobne) je špecifikované anotáciami priamo v entitných objektoch.

Pred deployom projektu na server je ešte potrebné ručne nakopírovať JDBC ovládač (nájdeme ho v našej lokálnej Maven repository v adresári \postgresql\postgresql\8.4-701.jdbc4\postgresql-8.4-701.jdbc4.jar) do:

- \jboss-5.1.0.GA\server\all\lib\
- \jboss-5.1.0.GA\server\default\lib\
- \jboss-5.1.0.GA\server\standard\lib\

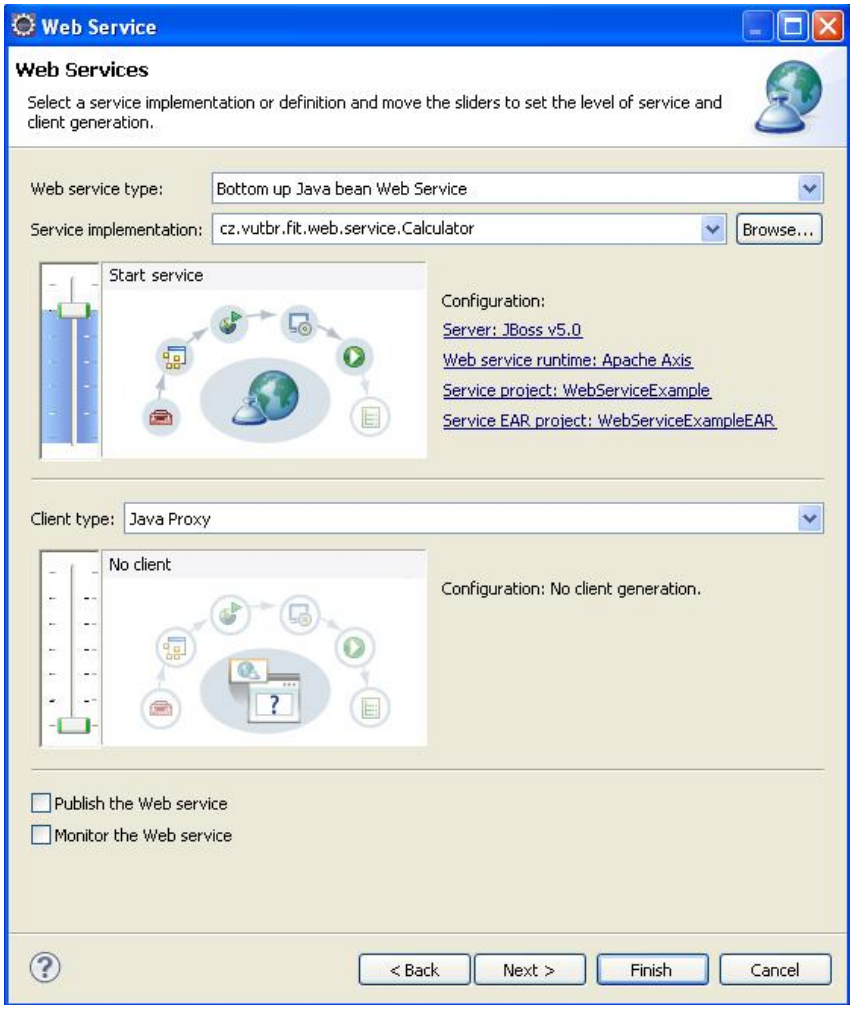
Musíme to podľa [26] vykonať z toho dôvodu, pretože distribúcia JBoss obsahuje HSQL databázu a primárne sa snaží pripojiť k nej. Ak chceme definovať pripojenie k inej databáze, musíme daný ovládač nakopírovať do vyššie spomenutých adresárov, čím nahradíme primárny dátový zdroj. URL aplikácie, ktorá predstavuje jednoduchú filmovú databázu, je http://localhost:8080/JPAExample/index.jsp a poskytuje príklady na takzvané CRUD (create/read/update/delete) operácie.

4.7 Ukážková aplikácia demonštrujúca webové služby

Táto aplikácia pozostáva podobne ako ukážková aplikácia pre EJB z dvoch častí. Prvou je samotná webová služba (WebServiceExample) a druhou je jednoduchý aplikačný klient, ktorý túto službu volá. Pri vytváraní webovej služby postupujeme spočiatku úplne rovnako ako pri vytváraní Dynamic Web Project. Po vytvorení príslušných balíkov v nich vytvoríme jednoduchú triedu, ktorú označíme anotáciou `@WebService`. Ukážková webová služba je jednoduchou

kalkulačkou, ktorá obsahuje jedinú metódu `public Integer calculate(Integer arg1, Integer arg2, String operation)`, ktorá vykoná špecifikovanú aritmetickú operáciu nad dvoma argumentmi, ktoré sú nej predané. Podporované oprácie sú sčítanie („+“), odčítanie („-“), celočíselné násobenie („x“) a celočíselné delenie („/“). Táto metóda je anotovaná ako `@WebMethod`, čo umožní jej zviditeľnenie pre klientov tejto webovej služby.

Po ukončení implementácie zdrojových súborov, klikneme v Project Exploreri pravým tlačidlom, zvolíme New->Other..-> tu v záložke Web Services vyberieme Web Service a zvolíme Next. Na ďalšej obrazovke (Obrázok 4.7) špecifikujeme implementáciu webovej služby (naša trieda Calculator) a ovládací prvok pri obrázku nastavíme do predposlednej polohy (Start service). Toto nastavenie zaručuje, že webová služba bude zostavená, pripojená na server, nainštalovaná a spustená. V časti „*Configuration*“ zvolíme server JBoss v5.0 (ak ešte nie je), „*WebServiceRuntime: Apache Axis*“, „*Service Project*“ poukážeme na náš projekt WebServiceExample a v okne „*Service EAR Project*“ zvolíme meno EAR projektu, ktorý nám Eclipse vygeneruje. Klientskú časť ponecháme zatiaľ tak ako je (vytvoríme neskôr samostatne). Na ďalšej obrazovke skontrolujeme, či máme zaškrtnutú našu anotovanú metódu. Po kliknutí na next nám Eclipse vygeneruje *wsdl* súbor, čo je popisný XML súbor, ktorý špecifikuje danú webovú službu. Po ukončení tejto operácie zvolíme na nasledujúcej obrazovke možnosť „*Launch the Web Service Explorer to publish this Web Service to a UDDI Registry*“ a klikneme Finish. Ak otvoríme súbor \WebContent\wsdl\Calculator.wsdl a zvolíme záložku Source tak na konci daného dokumentu vidíme adresu, na ktorej naša webová služba načúva. Ak pristúpime na adresu http://localhost:8080/WebServiceExample/services uvidíme všetky webové služby, ktoré sú v rámci nášho Java EE servera spustené a medzi nimi aj našu službu Calculator.



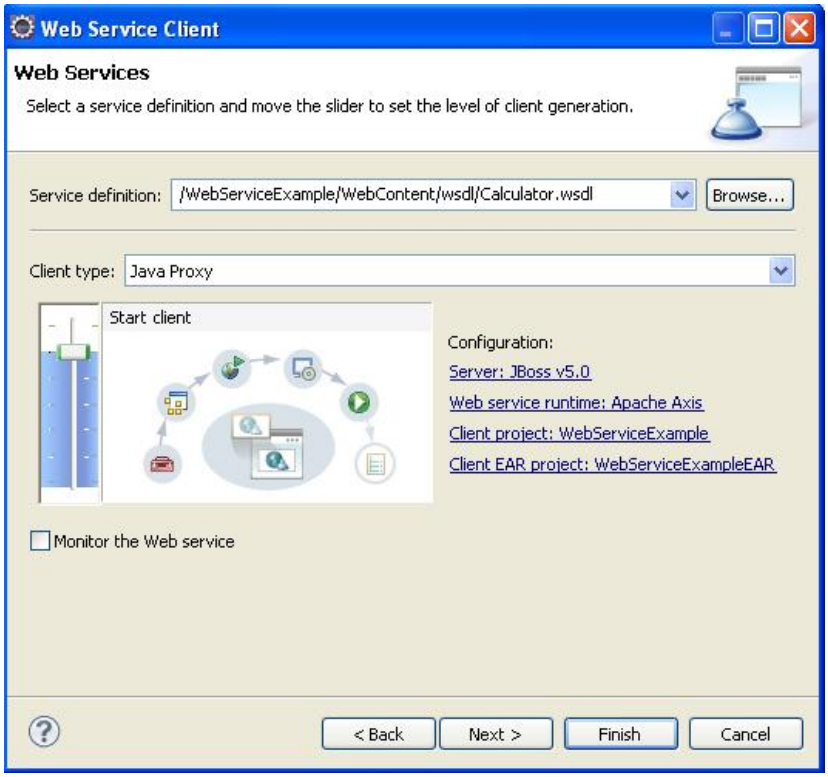
Obrázok 4.7: Vytvorenie webovej služby

Vytvorenie klienta webovej služby ponecháme na Eclipse. Klikneme opäť pravým tlačidlom myši do Project Explorera, zvolíme New->Other..-> a v záložke Web Services vyberieme Web Service Client. Na obrazovke, ktorá sa nám objavila (Obrázok 4.8) zvolíme v Service definition cestu ku nášmu vygenerovanému Calculator.wsdl súboru, pri obrázku nastavíme ovládacím prvkom možnosť Start client. V konfiguračnej časti musí byť Web service runtime rovnaký ako u webovej služby, ktorú sme predtým vytvorili, Client project nazveme napríklad WebServiceClientExample a Client EAR Project zvolíme už vygenerovaný WebServiceExampleEAR (Obrázok 4.9). Na nasledujúcej obrazovke zvolíme umiestnenie vygenerovaných zdrojových súborov a klikneme Finish. Po chvíli nám Eclipse vytvorí nový Dynamic Web Project a v ňom niekoľko tried a rozhraní, ktoré sa starajú o komunikácu a pripojenie k webovej službe (Obrázok 4.10). Našou jedinou úlohou je vytvoriť triedu

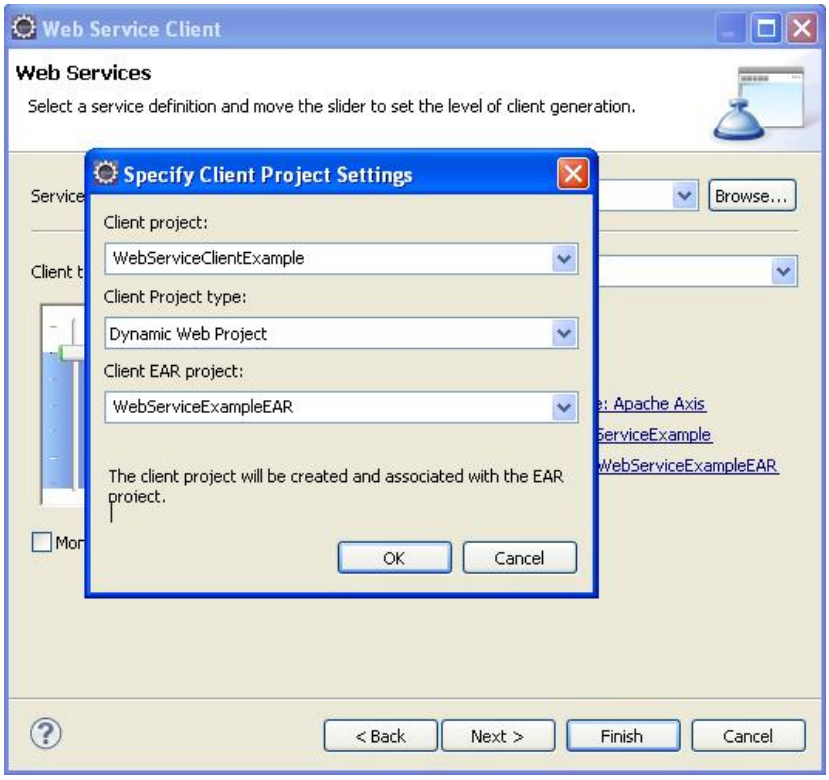
obsahujúcu metódu main a použiť v nej tieto predpripravené triedy a metódy pre volanie webovej služby. Webovú službu môžeme volať veľmi jednoducho za pomoci nasledujúceho kódu, ktorý vytvorí proxy objekt pre našu webovú službu:

```
CalculatorService service = new CalculatorServiceLocator();
Calculator calculator = service.getCalculator();
```

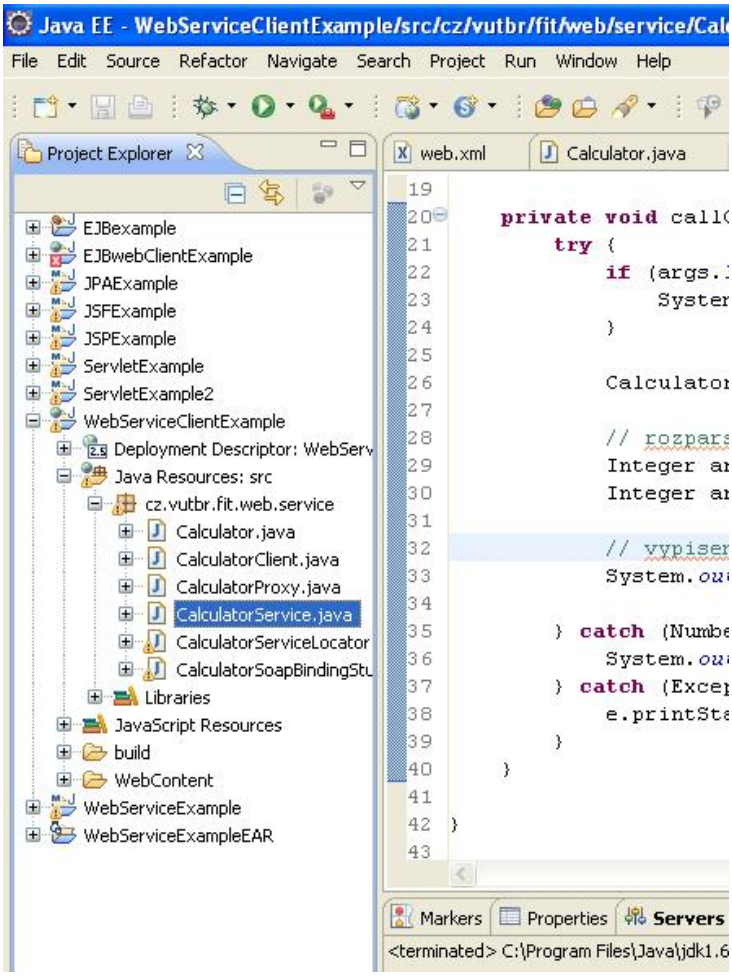
V tomto momente môžeme zavolať metódu `calculator.calculate(arg1, arg2, operation);` a tá nám za pomoci RPC volania zavolá požadovanú metódu a vráti výsledok.



Obrázok 4.8: Vytvorenie klienta webovej služby



Obrázok 4.9: Konfigurácia klienta webovej služby



Obrázok 4.10: Vygenerované triedy a rozhrania klienta webovej služby

Ak chceme spustiť klienta a zavolať webovú službu klikneme pravým tlačidlom v Project Exploreri na triedu obsahujúcu metódu main, v menu Run As vyberieme položku „*Run Configurations...*“, kde v záložke „*Arguments*“ vložíme do poľa „*Program arguments*“ argumenty pre metódu main (napríklad 3 + 4 oddelené medzerami). Zavrieme okno a opäť klikneme na danú triedu pravým tlačidlom, teraz však z menu Run As vyberieme Java Application. Eclipse spustí metódu main v tejto triede, ktorá rozparsuje vstupné argumenty, zavolá webovú službu a vráti výsledok aritmetickej operácie, ktorý vypíše do konzoly.

5 Záver

Cieľom tejto práce bolo oboznámiť čitateľa s platformou Java Enterprise Edition a so základnými rysmi technológií, ktoré sú súčasťou tejto platformy. Jej ďalšou úlohou bolo uviesť čitateľa do problematiky praktického vývoja aplikácií založených na týchto technológiách od získania vývojových prostriedkov, cez ich nastavenie, implementáciu aplikácií, ich zostavenie a nasadenie. Táto práca nemá za úlohu doslovne popísať používané technológie, ich kompletnú špecifikáciu je možné nájsť v literatúre, z ktorej som pri tvorbe práce čerpal.

Keďže existuje málo zdrojov popisujúcich platformu Java EE v slovenčine respektíve v češtine, považujem túto prácu prínosnú hlavne pre začínajúceho Java EE programátora, ktorému poskytuje dostatočné množstvo informácií pre zorientovanie sa v tejto problematike. Kapitola popisujúca samotnú platformu Java Enterprise Edition mu predá potrebné teoretické znalosti o popisovaných technológiách, ktoré čitateľ môže okamžite konfrontovať s praxou nahliadaním do priložených ukážkových aplikácií na DVD disku.

V prípade, že sa čitateľ rozhodne vyvíjať aplikácie na platforme Java EE, kapitoly 3 a 4 mu poskytnú praktické znalosti a rady ako pripraviť prostredie počítača, nainštalovať naň potrebný software a ako implementovať v praxi technológie špecifikácie Java EE.

Dúfam, že táto práca upúta čo najviac začínajúcich vývojárov, ukáže im nespochybniteľné výhody platformy Java Enterprise Edition a prípadne im dopomôže pri výbere technológie vhodnej pre ich projekt.

Literatúra

[1] Jendrock, E., Ball, J., Carson, D., Evans, I., Fordin, S., Haase, K. The Java EE 5 Tutorial [online]. Posledná modifikácia: 17.9.2007. [cit. 2009-10-25]. Dostupné na URL: <http://java.sun.com/javaee/5/docs/tutorial/doc/index.html>.

[2] Downey, T. Web Development With Java Using Hibernate, JSPs and Servlets. Springer, London, 2008. ISBN 978-1-84628-862-3.

[3] The O'Reilly Java Authors. Java Enterprise Best Practices. 1. vydanie. O'Reilly & Associates, Sebastopol, CA, USA, 2002. ISBN 978-0-596-00384-5.

[4] Hunter, J., Crawford, W. Java Servlet Programming. 1. vydanie. O'Reilly & Associates, Sebastopol, CA, USA, 1998. ISBN 1-56592-391-X.

[5] *Java BluePrints > Enterprise BluePrints* [online]. c2010, posledná modifikácia 16.2.2010 [cit. 2010-05-12]. Dostupné na URL: <http://java.sun.com/blueprints/enterprise/>.

[6] *Turbine Site – Turbine* [online]. c2000, posledná modifikácia 19.2.2009 [cit. 2010-05-12]. Dostupné na URL: <http://turbine.apache.org/>.

[7] Bergsten, H. JavaServer Pages. 2. vydanie. O'Reilly & Associates, Sebastopol, CA, USA, 2002. ISBN 0-596-00317-X.

[8] *Open Source JSP Tag Libraries* [online]. Posledná modifikácia 12.5.2010 [cit. 2010-05-12]. Dostupné na URL: <http://java-source.net/open-source/jsp-tag-libraries>.

[9] Johnson, R. Expert One-on-One J2EE Design and Development. Wiley Publishing, Indianapolis, IN, USA, 2003. ISBN 0-7645-4385-7.

[10] Chappell, D., Jewell, T. Java Web Services. 1. vydanie. O'Reilly & Associates, Sebastopol, CA, USA, 2002. ISBN 0-596-00269-6.

[11] Roman, E., Brose, G., Sriganesh, R. P. Mastering Enterprise JavaBeans. 3. vydanie. Wiley Publishing, Indianapolis, IN, USA, 2005. ISBN 0-7645-7682-8.

[12] Johnson, R., Hoeller, J. Expert One-on-One J2EE Development without EJB. Wiley Publishing, Indianapolis, IN, USA, 2004. ISBN 0-7645-5831-5.

[13] *JavaMail API Design Specification Version 1.4* [online]. Posledná modifikácia 17.4.2006. [cit. 12.1.2010]. Dostupné na URL: <http://java.sun.com/products/javamail/JavaMail-1.4.pdf>.

[14] Calder, B., Shannon, B. *JavaBeans Activation Framework Specification Version 1.1* [online]. Posledná modifikácia 17.4.2006. [cit. 12.1.2010]. Dostupné na URL: <http://java.sun.com/javase/technologies/desktop/javabeans/glasgow/JAF-1.1.pdf>.

[15] *Hibernate Annotations* [online]. c2004, posledná modifikácia 15.4.2010 [cit. 2010-05-12]. Dostupné na URL: <http://docs.jboss.org/hibernate/stable/annotations/reference/en/html_single>.

[16] *apdx_aa.gif (Obrázok GIF)* [online]. Dostupné na URL: <http://download.oracle.com/docs/cd/B15897_01/web.1012/b14361/graphics/apdx_aa.gif>.

[17] *figure09_01.jpg (Obrázok JPEG)* [online]. Posledná modifikácia 27.8.2008 [cit. 2010-05-12]. Dostupné na URL: <http://java.sun.com/blueprints/corej2eepatterns/Patterns/images09/figure09_01.jpg>.

[18] Campbell, J. *JBoss, Geronimo, or Tomcat? – JavaWorld* [online]. 12.11.2007, posledná modifikácia 12.5.2010 [cit. 2010-05-12]. Dostupné na URL: <http://www.javaworld.com/javaworld/jw-12-2007/jw-12-appservers.html?page=1>.

[19] *Apache Geronimo v2.2 Documentation: PLUGIN_RELEASE-NOTES-2.2.0.txt* [online]. c2003, posledná modifikácia 27.4.2010 [cit. 2010-05-12]. Dostupné na URL: <http://cwiki.apache.org/GMOxDOC22/pluginrelease-notes-220txt.html>.

[20] *Maven - What is Maven?* [online]. c2002, posledná modifikácia 12.5.2010 [cit. 2010-05-12]. Dostupné na URL: <http://maven.apache.org/what-is-maven.html>.

[21] *Problem installing 5.1GA on solaris... - JBoss Community* [online]. 27.5.2009, posledná modifikácia 12.5.2010 [cit. 2010-05-12]. Dostupné na URL: <http://community.jboss.org/message/7318#7318>.

[22] *Administration And Configuration Guide* [online]. november 2008, posledná modifikácia 5.12.2009 [cit. 2010-05-12]. Dostupné na URL: <http://www.jboss.org/file-access/default/members/jbossas/freezone/docs/Administration_And_Configuration_Guide/5/html_single/index.html>.

[23] *Garbage Collection - Frequently Asked Questions* [online]. c2003, posledná modifikácia 18.12.2007 [cit. 2010-05-12]. Dostupné na URL: <http://java.sun.com/docs/hotspot/gc1.4.2/faq.html>.

[24] *DbUnit Plugin - Maven 2 DbUnit Plugin – Introduction* [online]. c2006, posledná modifikácia 14.7.2009 [cit. 2010-05-12]. Dostupné na URL: <http://mojo.codehaus.org/dbunit-maven-plugin/>.

[25] *SQL Maven Plugin – Introduction* [online]. c2006, posledná modifikácia 15.2.2010 [cit. 2010-05-12]. Dostupné na URL: <http://mojo.codehaus.org/sql-maven-plugin/>.

[26] *Chapter 8. Using other Databases* [online]. Posledná modifikácia 18.5.2005 [cit. 2010-05-12]. Dostupné na URL: <http://docs.jboss.org/jbossas/getting_started/v4/html/db.html>.

[27] *SOA@WORK: IT job trends - Which technologies you should learn next* [online]. Posledná modifikácia 12.5.2010 [cit. 2010-05-12]. Dostupné na URL: <http://www.soa-at-work.com/2010/02/it-job-trends-which-technologies-you.html>.