Univerzita Pardubice

Fakulta elektrotechniky a informatiky

INFORMAČNÍ SYSTÉM ZÁSILKOVÉ SLUŽBY POMOCÍ ORACLE ADF

Bc. Josef Šmíd

Diplomová práce

2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Josef ŠMÍD**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Informační systém zásilkové služby pomocí Oracle ADF**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

V první části práce je nutné provést přehled technologií, které se používají pro vytváření robustních databázových aplikací (architektura Java Enterprise Edition, Oracle RDBMS, JSF, JPA, Oracle ADF Business (Faces) Components, Ajax, WebServices atd.) a popsat jejich vzájemnou spolupráci. Ve druhé části práce popsat princip činnosti návrhového vzoru MVC (Model-View-Controller) a blíže zhodnotit technologii Oracle ADF (Application Development Framework). Součástí práce bude celková analýza informačního systému a vytvoření UML diagramů. Hlavním cílem diplomové práce je vytvoření aplikace ?Informační systém zásilkové služby? s využitím výše popsaných principů a technologií. Aplikace bude implementovat struktury a funkce pro evidenci zákazníků, stavy a přehledy o jejich zakázkách; dále zaměstnance a vozidla firmy, jejich polohu a využití; evidenci zásilek, stav a jejich polohu. V práci budou implementovány nezbytné validace pro bezproblémový chod aplikace.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Oracle ADF Faces Components [online]. Oracle, [2007]. Dostupný z WWW: http://www.oracle.com/technology/products/adf/adffaces/index.html 2. Java EE at a Glance [online]. Sun Microsystems, Inc., c1994-2009. Dostupný z WWW: http://java.sun.com/javaee/**

Vedoucí diplomové práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 10. listopadu 2009

Prohlášení

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně Univerzity Pardubice.

V Pardubicích dne 17. 5. 2010

……..

Bc. Josef Šmíd

Poděkování

Děkuji tímto svému vedoucímu práce, panu Ing. Zdeňku Šilarovi za vedení, rady a podnětné připomínky při tvorbě této práce.

Anotace

Tato diplomová práce se zabývá aplikačním frameworkem Oracle ADF. V první řadě popisuje a seznamuje čtenáře s technologiemi, které daný framework používá. Dále je zmíněn a vysvětlen princip model-view-controller a jeho aplikace právě pomocí Oracle ADF. Samotný framework, práce s ním a jeho možnosti jsou popsány v další kapitole. Pro tento účel byla naprogramována aplikace „Informační systém zásilkové služby“, na které jsou možnosti frameworku využity. Tato kapitola je doplněna podrobným průvodcem – tutoriálem, který krok za krokem popisuje tvorbu aplikace. Zároveň je představen nástroj pro tvorbu aplikací pomocí Oracle ADF, kterým je vývojové prostředí Oracle JDeveloper.

Klíčová slova

Oracle ADF; framework; Java EE; informační systém; JDeveloper

Title

Information System of the Delivery Service using Oracle ADF

Annotation

This diploma thesis deals with Oracle ADF application framework. First of all, it describes and presents technologies used by the framework. Model-view-controller principle and its application with Oracle ADF help is mentioned and explained next. The framework itself, the work with it and its possibilities are described in the next chapter. For this purpose, "Mail Order Information System" application, on which the framework possibilities are applied, was programmed. This chapter contains a detailed guide – tutorial, which describes the application creation step by step. At the same time, application creation tool JDeveloper using Oracle ADF is introduced.

Keywords

Oracle ADF; framework; Java EE; information system; JDeveloper

Obsah

1. Úvod

Jsou různé cesty, jakými je možné se vydat při vývoji složitější a komplexnější aplikace. Začíná to výběrem programovacího jazyka, případně databázového systému. Je třeba zhodnotit možnosti, které daný jazyk nabízí, také obtížnost jazyka nebo úroveň dokumentace. Totéž platí pro databázový systém. Dalším krokem, který stojí za rozmýšlení, je výběr vhodného vývojového prostředí.

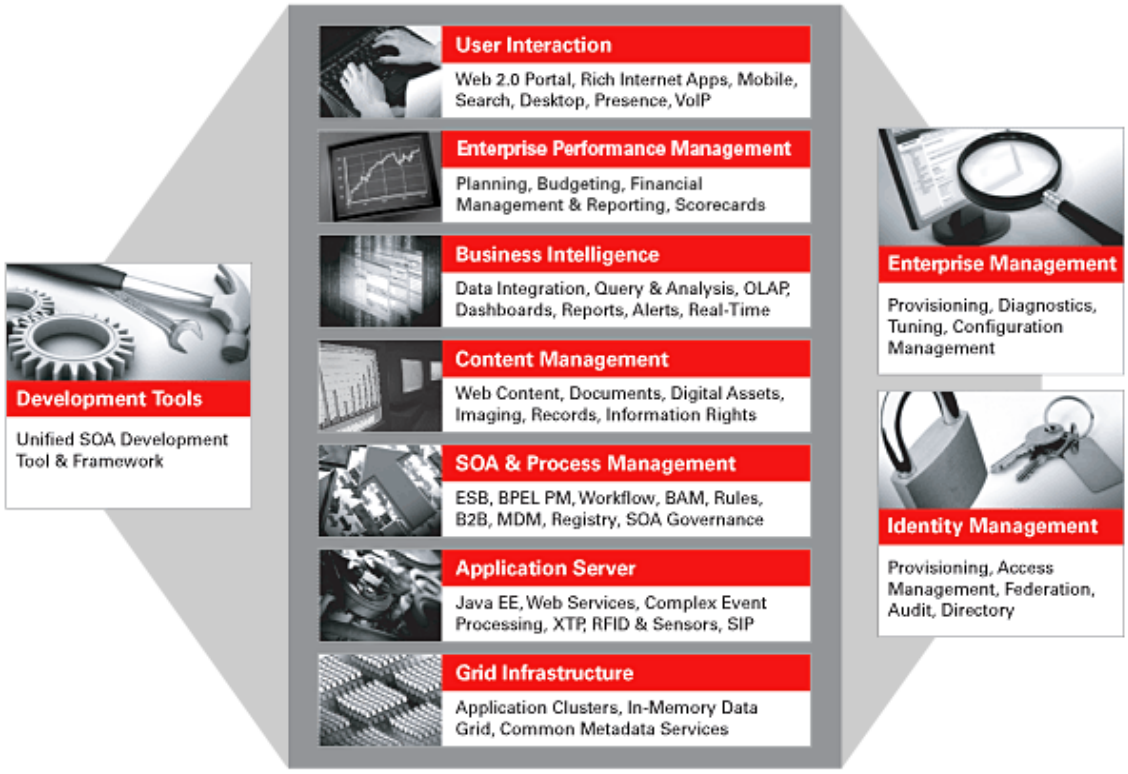
S takovýmto výběrem se musí potkat každý vývojář stojící před novým projektem. Někde jsou možnosti výběru dané požadavkem zadavatele, někde preferencemi samotného vývojáře. Nicméně tento prvotní výběr je velice důležitý, protože určuje a ohraničuje možnosti aplikace i vývojáře.

Jednou z možností, o které v porovnání s ostatními nelze najít tolik informací (a v českém jazyce to platí dvojnásob), je řešení společnosti Oracle. Jedná se o produkt Oracle ADF (Application Development Framework). Jak je již z názvu patrné, jde o framework pro vývoj aplikací. Konkrétně se jedná o kompletní sadu nástrojů a technologií integrovaných v rozsáhlém vývojovém prostředí. Pomocí takového vývojového prostředí pak lze vytvořit celou aplikaci bez toho, abychom museli například pro databázi použít jiný nástroj, jiný pro kompilaci apod.

Oracle ADF je vývojovým nástrojem pro tzv. Oracle Fusion Middleware (OFM) – kompletní rodinu produktů pro vývoj tzv. Enterprise aplikací. OFM obsahuje součásti zajišťující uživatelské rozhraní a komunikaci s uživatelem, tzv. Business Inteligence zapouzdřující práci s daty a jejich integraci do jednotlivých komponent a součástí, nástroje pro správu obsahu, tzv. Process Management a také například aplikační server. Tyto technologie budou popsány v jedné z následujících kapitol.

Hlavní síla Oracle ADF spočívá v jednoduchosti a připravenosti základních nástrojů, komponent a postupů používaných při tvorbě tzv. Fusion aplikací, tedy aplikací založených na principu OFM. Spolu s vývojovým prostředím JDeveloper, které všechny tyto prvky umožňuje použít, tak má vývojář velikou spoustu činností buď přímo vyřešených, nebo alespoň velice zjednodušených. Vše je založeno na vizualizaci jednotlivých postupů. Pomocí různých průvodců lze danou operaci či komponentu snadno sestavit a nakonfigurovat. Jedná se tedy o trochu jiný styl práce než klasické psaní zdrojo-

vého kódu, ale pokud vývojář do těchto specifik pronikne, může velice rychle postupovat ve vývoji dané aplikace.



Obrázek 1 – Schéma a součásti Oracle Fusion Middleware
Zdroj: www.oracle.com

V následujících kapitolách je nutné vymezit základní pojmy používané dále v práci a představit základní technologie, které jsou využity v jednotlivých částech OFM. Stranou nesmí zůstat ani architektura model-view-controller, která bude také představena. Praktická část je hlavně z důvodu nedostatku informací o ADF v českém jazyce doplněna rozsáhlým tutoriálem, podle kterého bude moci čtenář a zájemce podobnou aplikaci vytvořit sám.

1.1. Vymezení pojmů

1.1.1. Informační systém

Informační systém [1] je systém pro sběr, udržování, zpracování a poskytování dat. Obecně se jedná o účelově definovaný soubor komponent, mezi kterými existují určité vztahy a které splňují nějaký cíl. Informační systém může být v papírové podobě nebo elektronický, automatizovaný pomocí počítačů.

V kontextu této práce je pod pojmem informační systém míněn právě elektronický informační systém. Pro zjednodušení textu může být v této práci pro pojem informační systém použita zkratka IS.

1.1.2. Aplikace

Aplikace [2] je programové vybavení počítače, které je určeno pro přímou nebo i částečně nepřímou interakci s uživatelem. Je možno se také setkat s pojmem aplikační software nebo častěji jen software. Ovšem ne každý software musí nutně být v interakci s uživatelem. Například software síťových zařízení nebo telefonní ústředny může pracovat samostatně bez zásahu uživatele.

V této práci je pojem aplikace používán hlavně pro pojmenování výsledku práce programátora. Častěji je pojmem aplikace míněna webová aplikace, což je aplikace zpřístupněná uživatelům prostřednictvím webového serveru a počítačové sítě (Internet nebo intranet). Pod pojmem aplikace si tedy můžeme představit i konkrétní implementaci informačního systému. V tomto smyslu bude také pojem aplikace v této práci používán, hlavně v její druhé polovině.

1.1.3. Framework

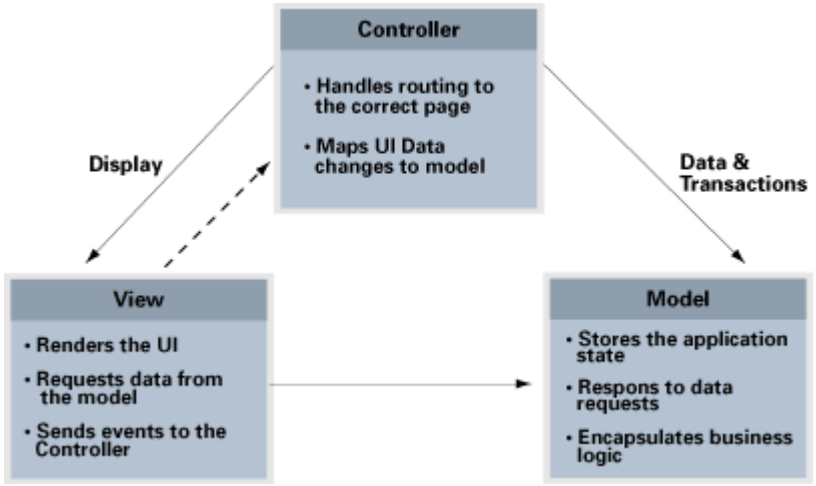
Framework [3] je softwarová struktura, která slouží jako podpora při programování a vývoji aplikací. Jeho cílem je usnadnit a předem vyřešit typické problémy dané oblasti vývoje. Vývojáři se pak mohou věnovat hlavně svému zadání a nemusí řešit základní a nutné operace, které nejsou samy o sobě složité, ale jejich implementace stojí drahocenný čas.

V Oracle ADF budiž jedním příkladem za všechny editační formuláře a tabulky. V tomto frameworku lze například tabulku pro editaci záznamu z databázové tabulky vytvořit jednoduše použitím tzv. „Data Control“ komponenty. Ve vizuálním editoru stránky se vytvoří kompletní formulář včetně zvolených polí, jejich popisků a navigačních tlačítek. Celá tvorba takového formuláře je záležitostí jednoho průvodce o několika krocích a vývojář tak nemusí zdlouhavě vytvářet jednotlivá pole pro jednotlivé řádky záznamu, navazovat je na tlačítka atd.

2. Architektura Model-view-controller

Aplikaci jako takovou je možné vyvinout a naprogramovat více způsoby. Většinou to záleží na zkušenostech a schopnostech vývojáře. Každý vývojář má své postupy a metody, jak dojít k určitému cíli. Pokud nějakou problematiku řeší poprvé, většinou dojde i k vyhledávání postupů, které použili jiní vývojáři a publikovali je pro ostatní. Postupem času se těchto publikací a návodů objevilo hodně a mnohdy byly velice podobné. Pro ty nejběžnější problémy při vývoji aplikací se tak průběžně vyrýsovaly nejvhodnější a nejefektivnější postupy – návrhy, jak ten konkrétní problém řešit. Zavedl se pojem návrhový vzor (design pattern).

Jedním z návrhových vzorů je také Model-view-controller (MVC). Mnohdy se také označuje jako architektura. Týká se totiž způsobu, jakým lze vystavět aplikaci, tedy její architekturou. Vznikl kolem roku 1978 a od té doby se velice často používá v různých frameworcích. Aplikaci lze totiž velice vhodně rozdělit do více vrstev, přičemž každá z vrstev je specificky zaměřená a stará se o jinou problematiku. Jedná se (jak je již z názvu jasné) od vrstvy model, view a controller.



Obrázek 2 – Architektura Model-view-controller
Zdroj: www.oracle.com

Princip funkce většiny informačních systémů (a obecně je možné říci i většiny aplikací) je následující [4]. Aplikace má v určité formě uložena data. Uživatel chce zobrazit, měnit nebo aktualizovat tato data. Vazba je tedy mezi uživatelem a daty. Bylo by tedy logické spojit tyto dvě části velice úzce tak, aby uživatel mohl snadno k datům přistupovat a pracovat s nimi. Je zde ovšem několik problémů, které tento zdánlivě vhodný přístup neřeší. Jednak se uživatelské rozhraní aplikací mění řádově častěji než datová

část (toto platí dvojnásob například u webových aplikací, kde dochází k redesignu často). Při změně uživatelského rozhraní by se tedy muselo zasáhnout i do části realizující práci s daty, což není vždy vhodné. Druhým problémem hlavně u rozsáhlejších aplikací, které jsou součástí větších systémů, je také to, že s částí aplikace starající se o práci s daty pracují i jiné systémy. Pak by změna této části kvůli změně uživatelského rozhraní znemožnila jiným systémům přistupovat k datům.

Pro doplnění je možné uvést i další případy, kdy se samo nabízí oddělení uživatelské části od datové. Je obvyklé, že jedno oddělení ve společnosti potřebuje stejná data zobrazit jinak než oddělení jiné. Například prodejce potřebuje kompletní informace o produktu, jeho popis a vlastnosti. Manažer ale potřebuje znát jen cenu, ale navíc třeba i prodejnost, případně další statistiky. Jiným příkladem může být i skutečnost, že uživatelské rozhraní může být technologicky odlišné pro různá zařízení nebo platformy. V takovém případě je lepší oddělit ho od logiky aplikace a vytvořit samostatné uživatelské rozhraní například pro klasické webové prohlížeče, samostatné pro mobilní zařízení a další pro desktopové aplikace zaměstnanců. Tato tři uživatelská rozhraní pak mohou využívat stejnou datovou část, která jim ve vhodné formě data poskytuje.

Architektura MVC odděluje vývoj aplikace do tří hlavních částí. Část nazvaná model představuje aplikační doménu. Je to nevizuální část, která obsahuje data a chování celé aplikace. Také se uvádí, že model zapouzdřuje tzv. business logiku, tedy opět způsob, jakým se aplikace chová a pracuje s daty. Část view je samotné uživatelské rozhraní. Patří sem vzhled jednotlivých stránek a formulářů a hlavně formátování zobrazovaných dat. Controller se pak stará o manipulaci s daty – reaguje na události a stará se o aktualizaci dat v části model nebo view.

Je důležité si uvědomit, že obě části – view i controller – jsou závislé na modelu (viz Obrázek 2). Naopak model nezávisí na žádné z ostatních dvou částí. Výhoda je zřejmá – business logiku aplikace lze vystavět a otestovat nezávisle na uživatelském rozhraní. Části view a controller bývají v mnoha vývojových prostředích poměrně slabě odděleny, někdy jsou i sloučeny do jedné části. Na druhou stranu například u webových aplikací je hranice mezi view a controller jasně definována. View je uživatelské rozhraní na straně klienta (webový prohlížeč), které se stará jen o zobrazení informací. Controller je pak serverová strana aplikace obsluhující požadavky klienta.

U každé aplikace je nutné zvážit, zda je svým zaměřením a rozsahem vhodná pro implementaci MVC. Je dobré zrekapitulovat výhody i nevýhody MVC a na základě toho se rozhodnout:

Výhody

- Část view je oddělená od zbytku aplikace. Lze tak aplikovat několik specifických částí view podle potřeby a účelu, například jeden view pro webový prohlížeč, další pro mobilní zařízení, jiný pro speciální aplikace atd.
- Část view lze libovolně změnit bez toho, aby bylo nutné zasáhnout do modelu. To je výhodou hlavně u aplikací, kde se vzhled a uživatelské rozhraní mění často, tedy typicky u webových aplikací.

Nevýhody

- U menších a jednodušších aplikací může použití MVC vést ke zvýšení složitosti a komplexnosti aplikace.
- Pokud je v aplikaci z její podstaty větší četnost aktualizací dat, může to vést ke zpomalení aplikace. Například může být náročné po každé aktualizaci přepočítávat různé grafy a statistiky. Proto by vývojář části model měl myslet i trochu na část view a implementovat techniky, které by tomuto zvýšení náročnosti zabránily nebo předešly.

O konkrétním využití architektury MVC v Oracle ADF bude pojednávat jedna z dalších kapitol.

3. Představení základních technologií

Každá aplikace vzniká v konkrétním vývojovém prostředí a za použití konkrétních technologií. Obojí je klíčové pro vývoj a určuje velké množství aspektů – vlastnosti aplikace, její chování a hlavně možnosti a technologie, které může programátor při vývoji aplikace využít. Nesprávně zvolená platforma nebo kombinace použitých technologií může vést ke komplikacím při vývoji. Může se jednat pouze o „maličkosti“, jako například zbytečná složitost implementace některých funkcí, i o zásadní problémy, například nemožnost implementace částí systému s požadovanou funkčností. Ono i ty takzvané maličkosti maličkostmi vlastně nejsou – systém by měl fungovat tak, jak je požadováno a každá (i když malá) takováto změna jen snižuje kvalitu a použitelnost systému.

Oracle ADF využívá pokročilé technologie založené především na jazyku Java. Stěžejní platformou je Java EE se všemi jejími součástmi, o kterých pojednávají další kapitoly. Další klíčovou technologií je XML, které je použito napříč celým frameworkem pro velkou spoustu dokumentů, ať už pro jednotlivé JSF fragmenty nebo pro Business komponenty. Velice úzce je svázán také s relační databází Oracle, hlavně přes vývojové prostředí JDeveloper, které má integrovanou součást SQL Developer pro vývoj a ladění databází.

3.1. Java EE

Java Platform, Enterprise Edition (dále jen Java EE) [5] je standardizovanou platformou pro vývoj tzv. enterprise aplikací (slovo „enterprise“ může být reprezentováno jako rozsáhlých, podnikových, webových aplikací) založených na programovacím jazyku Java. Je založena na platformě Java SE (Standard Edition) [6], která obsahuje nástroje pro vývoj a publikaci desktopových nebo serverových aplikací. Java EE navíc přidává zmíněnou „enterprise“ část, tedy knihovny a služby, které umožňují aplikaci vystavět jako škálovatelnou, přístupnou, bezpečnou a obecně takovou, která vyhovuje nárokům na rozsáhlé podnikové aplikace.

Java EE se poprvé objevila v roce 1999 s označením J2EE 1.2. Od té doby prošla několika dalšími vydáními a vylepšeními až po současnou verzi Java EE 6. Oproti první verzi byly přidány některé součásti zaměřené na robustnost vyvíjené aplikace ne-

bo webové služby. V poslední verzi byl také kladen důraz na zjednodušení procesu vývoje aplikace. Vzhledem k tomu, že ve většině aplikací není využívána každá dostupná součást platformy Java EE, podporuje nová verze profily. Je možné zvolit si pouze ty technologie a součásti, které vyvíjená aplikace bude používat. Součástí verze 6 je tzv. Web Profile, což je profil navržený pro vývoj současných webových aplikací. Obsahuje technologie pro transakční zpracování, zabezpečení aplikace, trvalost dat a další, které může vývojář webové aplikace potřebovat.

Součástí Java EE platformy jsou některé technologie, které jsou využívány i v Oracle ADF. Tyto budou popsány v následujících podkapitolách.

3.1.1. JSP

JSP (Java Server Pages) je technologie, která umožňuje vkládat části javovského kódu do statické HTML stránky. Jako vhodné přirovnání je možno uvést například technologie ASP nebo PHP.

JSP bylo navrženo pro vytváření webového obsahu, který má jak statické, tak dynamické části. V podstatě zpřístupňuje veškeré technologie jako při použití servletů, ale o hodně jednodušší je programování statického webového obsahu. To je výhodné při programování stránek, které neobsahují velké množství dynamicky generovaného obsahu. Zde je přehlednější, rychlejší a celkově efektivnější použít JSP místo servletu.

Velice silnou a užitečnou vlastností technologie JSP je možnost vytváření vlastních tagů (uživatelských značek) na bázi XML. Ve speciálním souboru označovaném jako Tag Library Descriptor definujeme kromě jiných parametrů tzv. manipulátory těchto uživatelských značek. To jsou třídy, které implementují programové chování těchto vlastních JSP značek. Tato vlastnost poskytuje prostor pro vývoj vícevrstvých aplikací, kde je programová logika oddělena od obsahu.

3.1.2. JSF

Java Server Faces ve verzi 2.0 jsou součástí platformy Java EE. Jedná se o framework běžící na straně serveru. Hlavní myšlenkou JSF je zjednodušit vývoj uživatelských rozhraní enterprise aplikací. Vývojář definuje uživatelské rozhraní pomocí speciálních XML tagů, kterým jsou předávána data ze standardních Java Beans. Tím dojde ke striktnímu oddělení vzhledu od logiky aplikace. Pokud tak na dané aplikaci

pracuje více vývojářů, každý se může věnovat jen své části – jeden řeší jen logiku aplikace a druhý řeší jen uživatelské rozhraní, přičemž pro zobrazení dat využívá výstupů právě z části logiky aplikace. Pokud uživatel přes webový prohlížeč pošle požadavek na zobrazení určité stránky, která využívá JSF, stane se následující:

Uživatel pošle požadavek na přístup k určité JSP stránce (její zobrazení). Tato stránka obsahuje JSF tagy. Při zpracování jsou místo JSF tagů doplněny komponenty uživatelského rozhraní, na které odkazují JSF tagy, jejich události, validátory a také hodnoty načtené a získané z Java Beans. Na výstupu je pak klasická HTML stránka, která je uživateli odeslána jako výsledek požadavku.

```
public class SalesTaxBean{
  // Two Attributes
  private double taxRate;
  private double amount;

  public SalesTaxBean(){ // Sample zero argument constructor
    setTaxRate("0");
    setAmount("0");
  }

  public double getTaxRate(){ // Tax Rate Getter
    return taxRate;
  }

  public void setTaxRate(String newTaxRate){ // Tax Rate Setter
    taxRate = Double.valueOf(newTaxRate).doubleValue()/100;
  }

  public double getAmount(){ // Getter for amount
    return amount;
  }

  public void setAmount(String newAmount){ // Setter for amount
    amount = Double.valueOf(newAmount).doubleValue();
  }

  public double getSalesTax(){ // Calculates the sales tax
    return amount * taxRate;
  }

  public double getTotal(){ // Calculates a total including tax
    return amount + (amount * taxRate);
  }
}
```

Obrázek 3 – Příklad jednoduché Java Bean
Zdroj: www.abbeyworkshop.com

3.1.3. Java Beans

V předchozí kapitole byl zmíněn pojem Java Bean. V této kapitole je vysvětleno, o co se vlastně jedná. Java Beans jsou klasické třídy zapsané v jazyce Java, ale mají určitá specifika. Většinou poskytují nějakou určitou funkčnost, například počítají nějaké hodnoty, případně je uchovávají či poskytují specifické služby (například odesílají e-maily). K proměnným takové třídy se přistupuje výhradně pomocí tzv. setget metod. U proměnné s názvem delka tedy musí být definovány metody setDelka(delka) a get-Delka(), které jsou automaticky v případě potřeby volány pro nastavení nebo načtení hodnoty proměnné délka.

Jednoduchý příklad Java Bean uvádí Obrázek 3. Je zde několik proměnných a zároveň jejich set a get metody. Metody getSalesTax() a getTotal() počítají potřebné hodnoty na základě těchto proměnných, zde není potřeba mít set metody.

3.1.1. EJB

Enterprise Java Beans (EJB) mají zdánlivě s Java Beans popisovanými v předchozí kapitole hodně společného. Ve skutečnosti však mají společné jen některé základní principy. Základní myšlenkou EJB je poskytnout framework pro tvorbu serverových komponent, které se „připojí“ k serveru, a tak rozšíří jeho funkčnost.

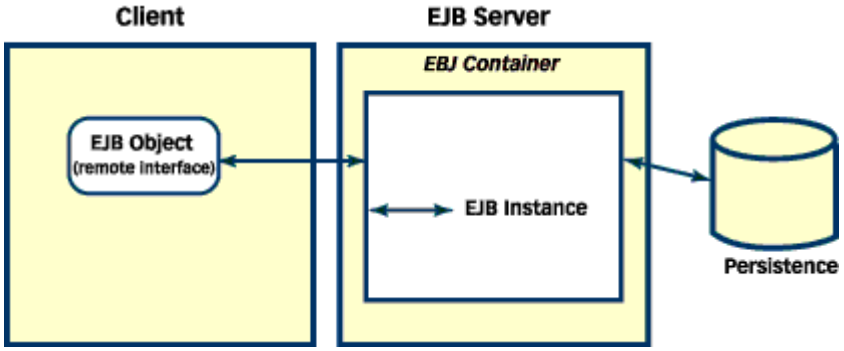
EJB je navrženo tak, aby usnadnilo vývojářům tvorbu aplikací. Můžou se soustředit na samotnou business logiku aplikace a nemusí řešit například obsluhu transakcí, vlákna apod. – toto všechno řeší framework za ně. Zároveň samozřejmě jsou EJB vlastně javovskými třídami, takže mohou využívat libovolné knihovny a jiná API, která jsou k dispozici.

Jak bylo zmíněno výše, EJB jsou serverovými komponentami. Aby bylo možné pochopit funkci klient-server řešení založeného na EJB, je nutné vědět, z jakých částí se EJB systém skládá. Jedná se o EJB komponenty, EJB kontejner a EJB objekt [7].

EJB komponenta je podobná klasické Java Bean. Běží uvnitř EJB kontejneru, který se spouští na EJB serveru. Je to javovská třída, která zapouzdřuje business logiku aplikace. Všechny ostatní třídy v EJB systému buď obsluhují přístup klienta ke komponentě, nebo pro ni zajišťují běžné a nutné služby.

EJB kontejner je místo, kde komponenta „žije“. Poskytuje jí základní služby jako obsluhu transakcí, zabezpečení, persistenci a jiné. Vývojář, který navrhuje komponentu, se tak může věnovat business logice a nemusí tyto základní (avšak nutné) záležitosti řešit. Pouze sdělí kontejneru, co chce provést a ten už je za provedení požadované operace zodpovědný. Ještě je nutné zmínit, že v jednom kontejneru může běžet více komponent a kontejner poskytuje dané služby všem komponentám.

EJB objekt je prostředek, pomocí kterého klient komunikuje se serverem. Implementuje rozhraní EJB komponenty a volá její metody (přes EJB kontejner). Na první pohled může objekt a komponenta vypadat stejně, protože implementují stejná rozhraní. Ve skutečnosti ale dělají různé činnosti. Komponenta běží na serveru a zapouzdřuje business logiku. Objekt běží na straně klienta a vzdáleně volá metody komponenty. Pro představu – EJB objekt si lze představit například jako obyčejný formulář. Celé schéma komunikace EJB systému popisuje Obrázek 4.



Obrázek 4 – Schéma EJB systému
Zdroj: www.javaworld.com

Existují dva typy Enterprise Java Bean – tzv. session bean a entity bean. První jmenovaná je svázána s jedním konkrétním klientem a existuje pouze uvnitř jedné session (tedy návštěvy uživatele na daném webu). Může například sledovat stránky, které uživatel navštívil. Po určitém čase uživatelovy neaktivity na webu zaniká. Entity bean reprezentuje nějaký objekt uložený v databázi. Může poskytovat data více uživatelům a existuje na serveru například i po jeho pádu – lze ji totiž rekonstruovat z dat v databázi. Příkladem může být objekt Zakázka, který reprezentuje záznam v databázové tabulce Zakázky.

3.2. XML

Jazyk XML (eXtensible Markup Language) zažívá posledních několik let velký rozmach. Vznikl z potřeby snadné výměny dat hlavně mezi vědeckými institucemi [8]. Tato data měla být nějakým nezneužitelným způsobem významově označena tak, aby byl druhou stranou pochopen logický význam každé jejich složky. Postupným vývojem došlo roku 1986 k definici a vzniku jazyka SGML (Standard Generalized Markup Language). Ten se pomalu blížil představám o takovém jazyce, ale jeho komplexnost bránila jeho dalšímu rozšíření.

Posledním „mezistupněm“ od SGML k XML je známý jazyk HTML, který byl právě v SGML definován. Stal se oblíbeným a používaným hlavně v souvislosti s internetem. Každopádně stále neřešil to, co bylo požadováno. Jednak HTML obsahuje značky (tagy), které slouží jak k významovému formátování textu, tak i k prezentaci (vzhledu). Nedávají tedy přesně informaci o tom, co se uvnitř tagu nachází za data. Další nevýhodou je omezená skupina předem definovaných tagů.

Nakonec jako podmnožina funkcí SGML vznikl právě jazyk XML. Jedná se o jazyk, který přímo informuje o významu dat v něm uložených. Obsahuje jen tagy informující o logické struktuře dokumentu, tedy žádné tagy prezentační. Velkou výhodou je to, že každý si může vytvořit vlastní sadu tagů přesně podle potřeb konkrétního použití. Aby nevznikaly kvůli této vlastnosti zmatky v tom, co který tag znamená, případně jaké má mít atributy, popisuje se struktura a význam tagů v samostatném souboru. V současné době existují dva způsoby, jak formálně popsat XML dokument. Prvním a straším (a stále méně využívaným) je DTD (Document Type Definition). Má mnoho nevýhod, které se stávají stále více nepřekonatelnými s tím, jak se jazyk XML rozšiřuje. Nepodporují jmenné prostory, datové typy (všechno je text, což dnes u všech XML dokumentů zdaleka neplatí), navíc syntaxe jazyka je taková, která se nikde jinde v XML nepoužívá.

Proto vzniklo XML Schema, což je stejně jako v případě DTD jazyk, který se používá k popisu XML dokumentu. Odstraňuje všechny výše zmíněné nevýhody DTD – podporuje jmenné prostory, datové typy, ale hlavně je sám XML dokumentem, takže s ním lze pracovat jako s kterýmkoli jiným XML souborem. Rozdíl mezi DTD a XML Schema souborem ukazuje Obrázek 5.

Využití XML je díky jeho vlastnostem opravdu široké – od různých ceníků přes dokumentace k programovacím jazykům a zdrojovým kódům nebo datové soubory aplikací až po například právě metadata objektů v Oracle ADF. Navíc díky DTD nebo XML schématu lze přidat k dokumentu jeho sémantický popis, aby i druhá strana znala význam a strukturu jednotlivých tagů dokumentu. Takové dokumenty pak lze libovolným způsobem zpracovávat nebo zobrazovat. Například pomocí XSL transformací je možné přiřadit jednotlivým tagům vzhled a přehledně je zobrazit.

DTD
```
<!ELEMENT zamestnanci (zamestnanec+)>
<!ELEMENT zamestnanec (jmeno, prijmeni, email+,
                       plat?, narozen)>
<!ELEMENT jmeno       (#PCDATA)>
<!ELEMENT prijmeni    (#PCDATA)>
<!ELEMENT email       (#PCDATA)>
<!ELEMENT plat        (#PCDATA)>
<!ELEMENT narozen     (#PCDATA)>
<!ATTLIST zamestnanec
          id     CDATA #REQUIRED>
```

WXS
```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="zamestnanci">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="zamestnanec"
                    maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="jmeno" type="xs:string"/>
              <xs:element name="prijmeni" type="xs:string"/>
              <xs:element name="email" type="xs:string"
                          maxOccurs="unbounded"/>
              <xs:element name="plat" type="xs:decimal"
```

Obrázek 5 – Ukázka části DTD a XML Schema dokumentu
Zdroj: www.kosek.cz

3.3. Ajax

Ajax (Asynchronous JavaScript and XML) je označení pro technologii vývoje interaktivních webových aplikací [9]. Základní myšlenkou je vylepšit jednu z nevýhod HTML stránek – nutnost při změně obsahu obnovit celou stránku. Stránka využívající AJAX pošle požadavek na server, který jí pošle zpět data, aniž by se musela obnovovat celá stránka. To vše pomocí JavaScriptu. Typickou a všem dobře známou realizací vyu-

živající AJAX jsou našeptávače při vyhledávání například v Googlu nebo Seznamu. Tato technologie má samozřejmě své nesporné výhody, ale i nevýhody.

Jednou z nevýhod je znemožnění funkce tlačítka Zpět v prohlížeči. Důvod je jednoduchý – tím, že se neobnoví celá stránka, nelze se nikam zpět vrátit. Prohlížeče takovou změnu jen části stránky nepodporují, a tak tlačítko Zpět ztrácí v tomto případě svou funkci. V některých případech (podle toho, jakým způsobem je aplikace napsaná) fungovat může, ale není zaručeno, že se aplikace vrátí do požadovaného stavu. Při takovýchto částečných změnách se také nemění URL v prohlížeči. Je tedy složité takto zobrazenou stránku například ukládat mezi záložky nebo poslat odkaz na ni e-mailem. Jako další nevýhodu lze brát fakt, že ne prohlížeči klienta musí fungovat JavaScript, aby byla zajištěna funkčnost Ajaxových komponent.

Nevýhod je relativně mnoho, ovšem zůstává jedna veliká výhoda – právě ta možnost načtení nebo obnovení jen části stránky. V dnešní době toho s oblibou využívá spousta webových aplikací (za všechny například Google nebo Facebook). AJAX je také základem komponent v Oracle ADF.

3.4. Oracle RDBMS

Oracle ADF je úzce spjatý s relační databází Oracle (RDBMS znamená Relational Database Management System) [10]. Framework jako takový sice spolupracuje i s jinými databázemi, ale použití databáze Oracle je v mnoha případech nejvhodnější.

Relační databáze je založena na modelu dvourozměrných tabulek (sloupce a řádky). Ty můžou nebo nemusí být propojeny (= vytvořena mezi nimi relace). Na rozdíl od hierarchického modelu, relační nepředpokládá žádné vazby mezi tabulkami. Uživatel nemusí rozumět fyzickému uspořádání dat v databázi, aby je mohl získat. Toto přispělo k velké popularitě a rozmachu relačních databází v 90. letech minulého století.

V současné době nabízí Oracle svůj databázový systém s označením Database 11g. Od své první verze prošel tento systém velkým počtem vylepšení a z jednoduché databáze podporující základní SQL se stal komplexní systém, který je využíván v těch největších společnostech.

Systém je nabízen v několika verzích rozdělených podle funkcí a určení. Pro soukromé či nekomerční nasazení v menších společnostech je k dispozici verze distribuovaná zdarma a určená pro obsluhu malých databází – Oracle Express Edition 10g (zde je vidět, že se jedná o předchozí verzi, nová verze 11g není zatím zdarma nabízena). Pro komerční a náročnější nasazení je nabízena jedna z pokročilejších verzi umožňujících obsluhu rozsáhlejších databází. Obrázek 6 ukazuje přehled v současné době nabízených verzí databáze Oracle.

Key Feature Summary	Express Edition 10g	Standard Edition One	Standard Edition	Enterprise Edition
Maximum	1 CPU	2 Sockets	4 Sockets	No Limit
RAM	1GB	OS Max	OS Max	OS Max
Database Size	4GB	No Limit	No Limit	No Limit
Windows	●	●	●	●
Linux	●	●	●	●
Unix		●	●	●
64 Bit Support		●	●	●

Obrázek 6 – Verze Oracle RDBMS
Zdroj: www.oracle.com

Je vidět, že se jednotlivé nabízené verze liší hlavně počtem procesorů a velikostí paměti serveru. Je tedy jasné, že pro malou databázi stačí zdarma distribuovaná verze Expres Edition, která může bez problému běžet na jednoprocesorovém serveru s menším množstvím paměti RAM. Na druhou stranu velká a rozsáhlá databáze musí využívat některou z vyšších verzí, která podporuje více procesorů a větší množství paměti.

3.4.1. Funkce a vlastnosti systému

Databázový systém musí splňovat mnoho předpokladů pro bezproblémovou činnost a vysoké zatížení. Musí být schopen obsloužit velké množství uživatelů a požadavků v jednom okamžiku, aniž by vznikl konflikt. Taktéž při chybě (ať už chybě serveru nebo uživatele) musí systém zachovat data v nezměněné formě. Tato kapitola popisuje některé základní vlastnosti databázového systému Oracle.

Hlavním problémem víceuživatelské databáze je kontrola souběžných přístupů ke stejným datům různými uživateli. V Oracle databázi je toto řešeno různými druhy „zámků“ založených na transakcích.

Tato vlastnost zabezpečuje, že data se v průběhu vykonávání dotazu nezmění. Dále zajišťuje, že požadavek na čtení dat nebude čekat, až se dokončí zápis a naopak. Jediný případ, kdy bude transakce čekat, je, pokud přijde požadavek na aktualizaci dat, která jsou právě aktualizována. Implementace dat v Oracle spočívá v tom, že každý uživatel pracuje s jakousi „soukromou databází“, kterou nemění jiný uživatel.

Oracle podporuje systém „zámků“, který umožňuje kontrolovat vícenásobné požadavky na data a zabezpečuje integritu dat. Systém funguje tak, že při aktualizaci dat jsou uzamčeny právě aktualizované řádky, dokud není transakce dokončena nebo potvrzena. Oracle umí použít neomezené množství řádků.

K dispozici je Oracle Enterprise Manager, což je nástroj pro kompletní správu DBMS. Obsahuje množství nástrojů, které umožňují skrze grafické uživatelské rozhraní i konzoli spravovat a vykonávat množství příkazů nad databází, například spravovat kompletní prostředí Oracle – databáze, aplikace, služby, modifikovat a ladit databáze, monitorovat stav databáze přes síťové rozhraní, plánovat spouštění úloh nad databází v určitém čase a další.

Dalším z nástrojů zajišťujících správu databáze je SQL* Plus. Tento nástroj umožňuje zadávání „ad-hoc“ SQL dotazů a PL/SQL kódu. Jeho rozšířením je systém iSQL* Plus, což je webové rozhraní, které poskytuje stejné služby v přehledné a graficky přívětivé formě webových stránek. Lze v něm tedy odkudkoliv z Internetu zadávat dotazy do databáze nebo modifikovat data. Uživatel má možnost zobrazit historii dotazů a také si prostředí nastavit podle svých požadavků.

Co se týče správy obsahu, Oracle nabízí podporu pro všechny běžné formáty dat – text, audio, video, obrázky a také XML (eXtensible Markup Language). Oracle zachází s XML daty jako s běžným datovým typem. Nabízí několik možností, jak například z dotazu rovnou vytvořit XML dokument.

Samozřejmostí u moderní databáze jsou integritní omezení (například NOT NULL nebo různé klíčové hodnoty – UNIQUE KEY, PRIMARY KEY, FOREIGN KEY). Podporovány jsou triggery, sekvence, pohledy, materializované pohledy nebo procedury a funkce v jazyce PL/SQL.

4. Oracle ADF architektura

Postavený na základě výše uvedených technologií, poskytuje Oracle ADF (dále také jen ADF) vývojářům komplexní nástroj pro tvorbu tzv. Fusion aplikací. Pojem Fusion aplikace je možné chápat jako pokročilou aplikaci integrující ty nejmodernější enterprise technologie. Fusion je vize společnosti Oracle pro podnikové technologie, aplikace a služby [11]. Slovní spojení „ve Fusion“ je tedy možné chápat jako „podle vize společnosti Oracle“. V této kapitole je nutné vysvětlit některé pojmy, principy a postupy při práci s ADF. Ty potom budou prakticky použity v dalších kapitolách.

ADF je založen na architektuře model-view-controller, která byla podrobněji popsána v jedné z předchozích kapitol [12]. Vrstva view zde poskytuje uživatelské rozhraní a vyvolává výjimky a události vrstvě controller. Základním prvkem této vrstvy je stránka. Vrstva controller řídí navigaci mezi stránkami v aplikaci. Zpracuje uživatelský vstup, ošetří chyby a rozhodne o další navigaci v aplikaci. Základním prvkem zde je tzv. task flow (bude vysvětleno dále). Vrstva model je potom zodpovědná za přístup k datům a business logiku. Tato vrstva je ve Fusion rozdělena na dvě podvrstvy. První z nich se stará o navázání modelu na view – jedná se o tzv. data control. Druhá je spojena s pojmem business components a stará se o samotné zpřístupnění dat.

4.1. Vrstva view

Jak bylo zmíněno výše, základním prvkem vrstvy view je stránka. Jedná se o klasickou stránku, která je zobrazena uživateli, s různými uživatelskými prvky – formuláři, obrázky, tlačítky a textem. Většinou jde o klasickou JSF stránku s příponou .jspx. Každopádně stránka je ve Fusion většinou použita jen pro základní (indexový) soubor a aplikace dále využívá hlavně dále zmíněné prvky, které zvyšují znovupoužitelnost a přístupnost celé aplikace.

Často využívaným prvkem je tzv. Page Fragment (dále fragment stránky). Na první pohled je velice malý rozdíl mezi stránkou a fragmentem – obě se vytvářejí podobným způsobem, mohou obsahovat stejné komponenty atd. Fragment ale není samostatnou stránkou. Je pouze možné ho vložit do části stránky, například do indexové stránky, která obsahuje menu, titulek a nadpis, vložíme fragment s obsahem. Soubor fragmentu stránky má příponu .jsff.

V souvislosti s těmito pojmy je nutné vysvětlit ještě jeden pojem, který bude dále hodně používán – facet. Facet je nějaká část buď stránky, ale hlavně některých ADF komponent, do které je možno vložit libovolný obsah. Analogií by mohlo být slovo „část“. Například komponenta Panel Splitter (bude představena v jedné z pozdějších kapitol) rozdělí plochu na dvě části (dva facety) – first a second.

Pro části aplikace společné všem stránkám se nabízí použití Page Template – šablony stránky. Aby nebylo nutné například hlavičku a patičku aplikace definovat v každé stránce, má vývojář možnost vytvořit šablonu, kterou pak využívají stránky. V šabloně je pak určen facet, ve kterém je vložena stránka a kde se mění obsah aplikace. Šablony můžou mít parametry, které jsou naplňovány hodnotami podle zobrazovaného obsahu. Příkladem může být titulek stránky.

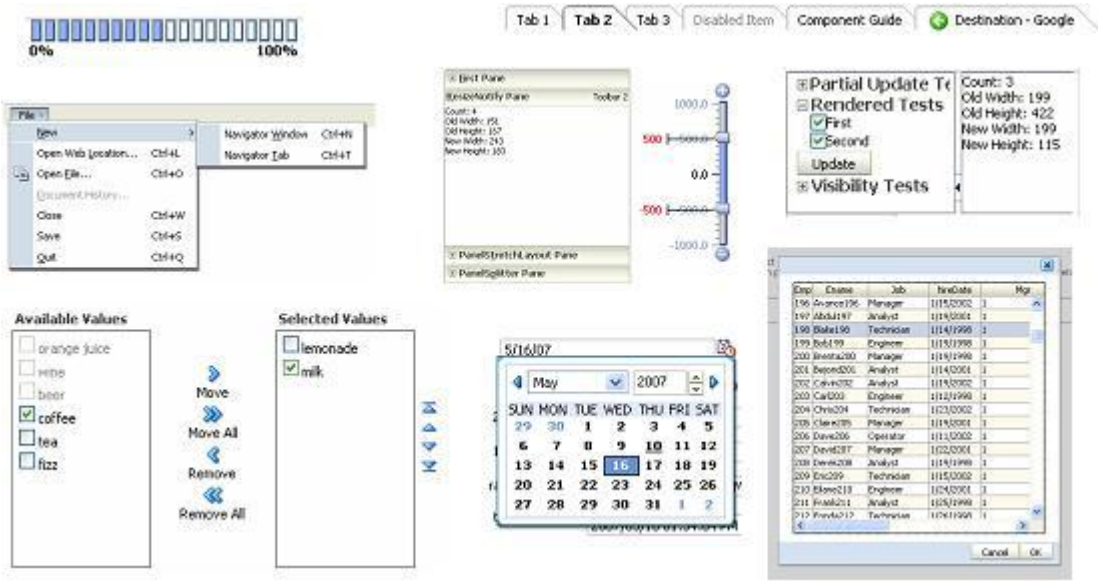
Důležitým pojmem ve view vrstvě Fusion aplikace je Region. Ten může být použit uvnitř stránky nebo fragmentu. Jak bude podrobněji popsáno dále, pro navigaci v aplikaci je vytvořen task flow. Ten může obsahovat několik stránek dané části aplikace a tyto stránky pak musí být někde zobrazeny. Ve vývojovém prostředí JDeveloper jednoduchým přetažením task flow do stránky nebo její části (do některého facetu) definujeme region, ve kterém se dané stránky z task flow zobrazí uživateli. Pro tento postup musí být použit tzv. Bounded Task Flow, který využívá fragmenty (vysvětleno níže).

4.1.1. ADF Rich Client komponenty

Jednou z hlavních předností uživatelské části ADF jsou vizuální komponenty, které vývojáři jistě znají z jiných vizuálních vývojových prostředí, například Borland Delphi nebo Microsoft Visual Studio. V ADF se jedná o komponenty založené na Ajaxu a určené pro web. Příklady takových komponent ukazuje Obrázek 7 a mnoho z nich je využito v praktické části této práce. Tyto komponenty se dají rozdělit do několika kategorií:

- komponenty členění stránky,
- tabulky a stromy,
- LOV komponenty („List of Values“),
- vstupní komponenty,

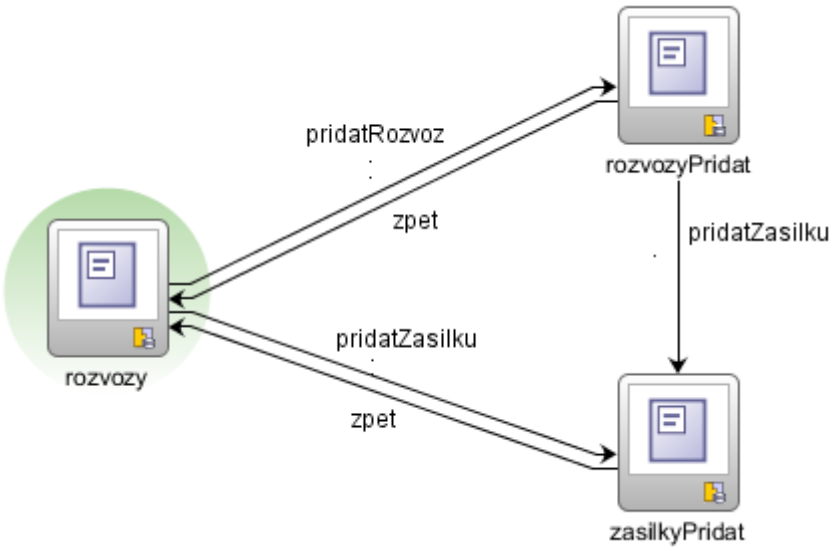
- navigační komponenty,
- výstupní komponenty,
- komponenty pro zobrazení dat.



Obrázek 7 – Ukázka některých ADF komponent
Zdroj: www.java.cz

4.2. *Vrstva controller*

Tato vrstva je nepostradatelná v každé Fusion aplikaci. Stará se o interakci s uživatelem a řídí navigaci mezi jednotlivými stránkami aplikace. Hlavním a klíčovým prvkem je zde tzv. Task Flow (viz Obrázek 8).



Obrázek 8 – Ukázka Task Flow
Zdroj: vlastní aplikace

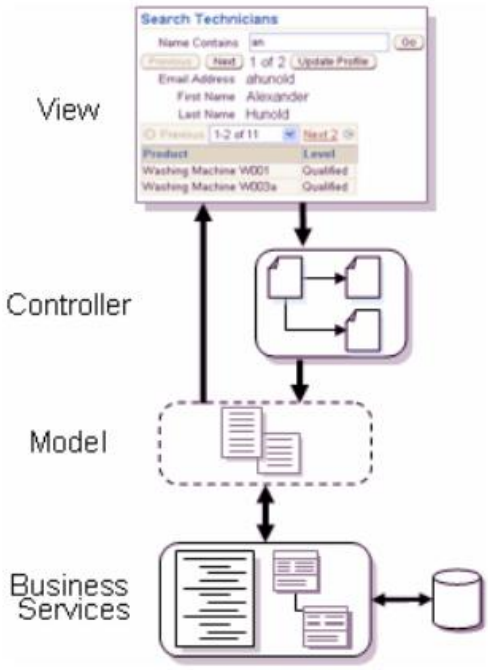
Task flow (TF) je nástroj, který umožňuje definovat jednotlivé úlohy aplikace a sestavit navigaci mezi nimi. To zahrnuje stránky a logiku, které uživatel využije při plnění takové úlohy. Při návrhu je základní komponentou View, která reprezentuje jednu stránku webu (nebo fragment). Pomocí Control Flow Case komponenty jsou jednotlivé view propojeny, a tím je definována navigace mezi stránkami. K dispozici jsou i další komponenty jako například Router, který simuluje CASE logiku, tedy rozhodování na základě podmínky, kterou stránku zobrazit.

Ve Fusion jsou dva typy task flow – bounded a unbounded. Bounded task flow podporuje navíc práci s transakcemi a znovupoužitelnost. Právě v bounded TF je možno používat fragmenty, v unbounded toto nelze. V naprosté většině případů bude v aplikaci použit bounded TF. Co se týče práce s transakcemi, lze velice snadno definovat počátek transakce a commit nebo rollback při jednotlivých výstupech z task flow. Může být definován jeden vstupní bod do task flow a žádný nebo více výstupních bodů. Bounded TF může mezi jeho částmi přenášet uživatelská data (tzv. pageFlowScope proměnné). Podporuje také vstup do TF s uživatelskými parametry a návratovou hodnotu při výstupu. Co se týče znuvupoužitelnosti, můžeme propojovat jednotlivé TF například pomocí Control Flow Case, a tím se dostat z jednoho TF do jiného (tedy z jedné části aplikace do jiné).

Unbounded TF tedy výše zmíněnou funkčnost nepodporuje. Každá aplikace může obsahovat jen jeden unbounded TF. Bývá většinou použit pro stránku, která obsahuje celé uživatelské rozhraní a dále je použit bounded TF pro navigaci mezi jednotlivými fragmenty.

4.3. Vrstva model

Tato vrstva se stará o přístup k datům a implementaci business logiky aplikace. Jak bylo zmíněno výše, ve Fusion je rozdělena do dvou částí – část Data Control, která se stará o navázání dat na view, a část Business Components, která zapouzdřuje konkrétní komponenty zpřístupňující data. Schéma Fusion aplikace ukazuje Obrázek 9. Zde je vidět právě ono rozdělení vrstvy model do dvou částí.



Obrázek 9 – Princip funkčnosti Fusion aplikace
Zdroj: www.oracle.com

4.3.1. Data Control

Data Controls zpřístupňují data získaná z databáze business komponentami do view vrstvy. Umožňují vytvořit tzv. Data Bound Pages, což jsou právě stránky, které zobrazují tato data. Může se jednat například o formulář nebo tabulku se záznamy z databáze. Ve vývojovém prostředí JDeveloper stačí pro vytvoření takové stránky přetáhnout příslušnou data control do stránky a pomocí průvodce vytvořit požadovanou strukturu (formulář, tabulku, graf, …).

4.3.2. Business Components

Business komponenty jsou druhou částí vrstvy model Fusion aplikace. Jsou hlavním zdrojem dat a business logiky aplikace. Komponent je několik druhů podle funkce, kterou zastávají.

Základem je tzv. Entity Object. Ten reprezentuje záznam v dané databázové tabulce se všemi jejími specifiky. Automaticky obsahuje business logiku dané tabulky včetně všech operací pro práci s tabulkou (operace INSERT, UPDATE, DELETE). Entity objekty můžou být propojeny tak, aby kopírovaly způsob propojení jednotlivých tabulek v databázi.

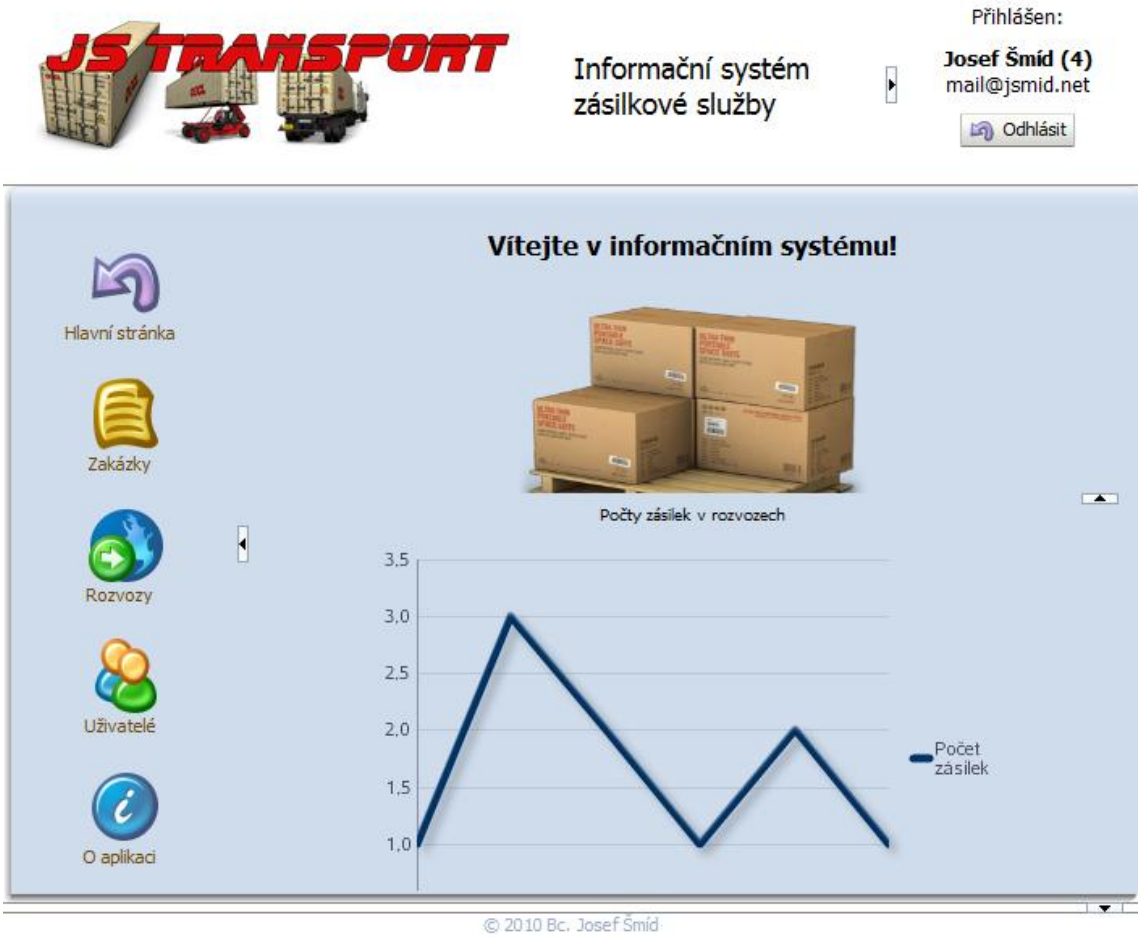
View Object je další z řady business komponent. Představuje vlastně SQL dotaz nad danou tabulkou nebo tabulkami a zapouzdřuje práci s jeho výsledky. Obsahuje funkčnost pro třídění, řazení a zobrazení dat v požadované formě. View objekty spolupracují s entity objekty – pokud uživatel změní zobrazená data, na základě této spolupráce jsou data validována a případně uložena do databáze. View objekty můžou být propojeny v hierarchii opět podle databáze, typickým použitím je tzv. master-detail vztah (na základě výběru konkrétního záznamu v master tabulce se načtou odpovídající data do detail tabulky).

Objekt Association představuje vztah mezi jednotlivými entity objekty. Definicí takového vztahu je možné propojit view objekty v požadované relaci. Takto definovaný vztah pak realizuje další business komponenta – View Link. Pomocí těchto komponent lze vytvořit libovolné relace od 1:N přes 0..1:0..1 až po M:N.

Poslední důležitou komponentou je Application Module. Jedná se o komponentu, která prostřednictvím view objektů poskytuje uživatelské rozhraní pro práci s daty. Podporuje transakce a je možné pomocí ní provádět typické operace pro práci s daty v databázových tabulkách. V případě dobře vytvořených relací mezi business komponentami zpřístupňuje v uživatelském rozhraní i tyto relace. Ve vývojovém prostředí JDeveloper je takovýto aplikační modul určený pro testování vytvořených business komponent.

5. Informační systém zásilkové služby

Úkolem praktické části této práce bylo vytvořit aplikaci s pracovním názvem „Informační systém zásilkové služby“ (viz Obrázek 10). Typickým uživatelem takového systému je fiktivní společnost zabývající se doručováním zásilek zákazníků. Tato společnost potřebuje pro snadnější organizaci a správu jednotlivých zásilek provozní informační systém, který umožní vést evidenci zákazníků, automobilů, vkládat a organizovat zakázky, plánovat rozvozy a přiřazovat jednotlivé zásilky do konkrétních rozvozů. Zároveň má samozřejmě více než jednoho zaměstnance a všichni zaměstnanci potřebují s aplikací pracovat, tím pádem je zde požadavek na multiuživatelský přístup (tedy přihlašování každého uživatele/zaměstnance svým jménem a heslem). Tím logicky vzniká i požadavek na zabezpečení dat – k údajům zákazníků, zakázek či rozvozů se může dostat jen pověřená osoba.



Obrázek 10 – Hlavní okno aplikace
Zdroj: vlastní aplikace

5.1. *Zjednodušení aplikace*

Pro účely této práce byla samozřejmě vzorová aplikace zjednodušena. Hlavním cílem má být ukázka postupů a principů práce ve vývojovém prostředí JDeveloper a také ukázka možností Oracle ADF. Proto některé skutečnosti, které by v reálném prostředí byly žádané, nebo dokonce nepostradatelné, nejsou v tomto vzorovém prostředí důležité.

Jedná se hlavně o abstrakci průběhu času. Aby bylo možné aplikaci jednoduše testovat, zanedbáme průběh času při vkládání dat. Bude tak možné například vložit rozvoz s datem starším než je aktuální nebo vložit do rozvozu zásilku, která byla vytvořena později, než proběhl samotný rozvoz. Tyto skutečnosti by v reálném systému nemohly existovat, pro naše testovací prostředí ale nejsou důležité.

Dále je aplikace zjednodušena o některé vazby a omezení mezi objekty nebo entitami. Například je možné vložit jednu konkrétní zásilku do několika rozvozů nebo je možné libovolně měnit stav zakázek a zásilek nezávisle na činnostech, které jsou s nimi prováděny.

5.2. *Analýza a případy užití*

Analýza problému sestávala v první řadě ze sepsání požadavků, které jsou na danou aplikaci kladeny. Na základě těchto požadavků vznikl diagram analytických tříd, který ukazuje Příloha A. Základem je objekt Uzivatel, který je předkem objektů Zakaznik a Zamestnanec. Zákazník může vlastnit libovolný počet zakázek. Každá zakázka je spravována zaměstnancem. Každá zakázka se skládá z několika zásilek. Zásilka je součástí rozvozu, rozvoz obsahuje několik zásilek. Stranou je objekt Automobily, který již podle svého jména představuje automobil.

Diagram případů užití zobrazuje Příloha B. Je zde jeden hlavní aktér – zaměstnanec. Ten obsluhuje většinu operací v systému. Spravuje databázi zákazníků, obsluhuje a zpracovává zakázky, plánuje rozvozy a přiřazuje zakázky do konkrétních rozvozů. Také mění údaje v databázi automobilů.

Vedlejším (ale nikoli nepodstatným) aktérem je administrátor. Ten se stará o databázi zaměstnanců, opravuje chyby v datech a stará se o technické zázemí systému.

5.3. *Datový model*

Informační systém používá jako datové úložiště databázový systém Oracle. Všechna data jsou uložena v tabulkách databáze. Použité databázové schéma je sestaveno kromě tabulek i ze dvou pohledů (VIEW), několika sekvencí (SEQUENCE) a triggerů pro obsluhu těchto sekvencí (TRIGGER). Kompletní databázové schéma je funkční tak, jak ukazuje Příloha C.

Základem je tabulka Uzivatele, která obsahuje společné informace jak o zaměstnancích, tak i o zákaznících. Jedná se hlavně o e-mailovou adresu (unikátní klíč), heslo, jméno a informace o přihlášení do systému. Doplňujícími tabulkami jsou Zakaznici a Zamestnanci. První jmenovaná tabulka doplňuje uživatele o fakturační údaje – jméno a adresu. Tabulka Zamestnanci pak přidává uživateli informace o jeho pracovní pozici.

Na základě těchto tří tabulek pracují dva pohledy – Zakaznici_view a Zamestnanci_view. Oba dva zobrazí shodně všechny atributy zákazníka nebo zaměstnance. Uživatel, který nemá odpovídající záznam buď v tabulce Zakaznici, nebo Zamestnanci, nebude v těchto pohledech vypsán.

Další klíčovou částí je tabulka Zakazky. Ta obsahuje informace o zakázkách, které jsou vkládány do systému. Každá zakázka se skládá z jedné nebo několika zásilek, které jsou uloženy v tabulce Zasilky. Zásilka pak má atributy jako hmotnost, speciální informace (křehké, rychle se kazící apod.) nebo adresu doručení.

Jednotlivé zásilky můžou být vkládány do rozvozů. Děje se tak přes tabulku Rozvozy_has_zasilky, která spojuje v M:N relaci záznamy tabulek Rozvozy a Zasilky.

Datový model doplňují tabulky Stavy, Mesta a Automobily, které obsahují informace odpovídající svému názvu – seznam stavů, které jsou přiřazovány zásilkám a zakázkám, seznam všech obcí a měst ČR a automobily, které společnost vlastní.

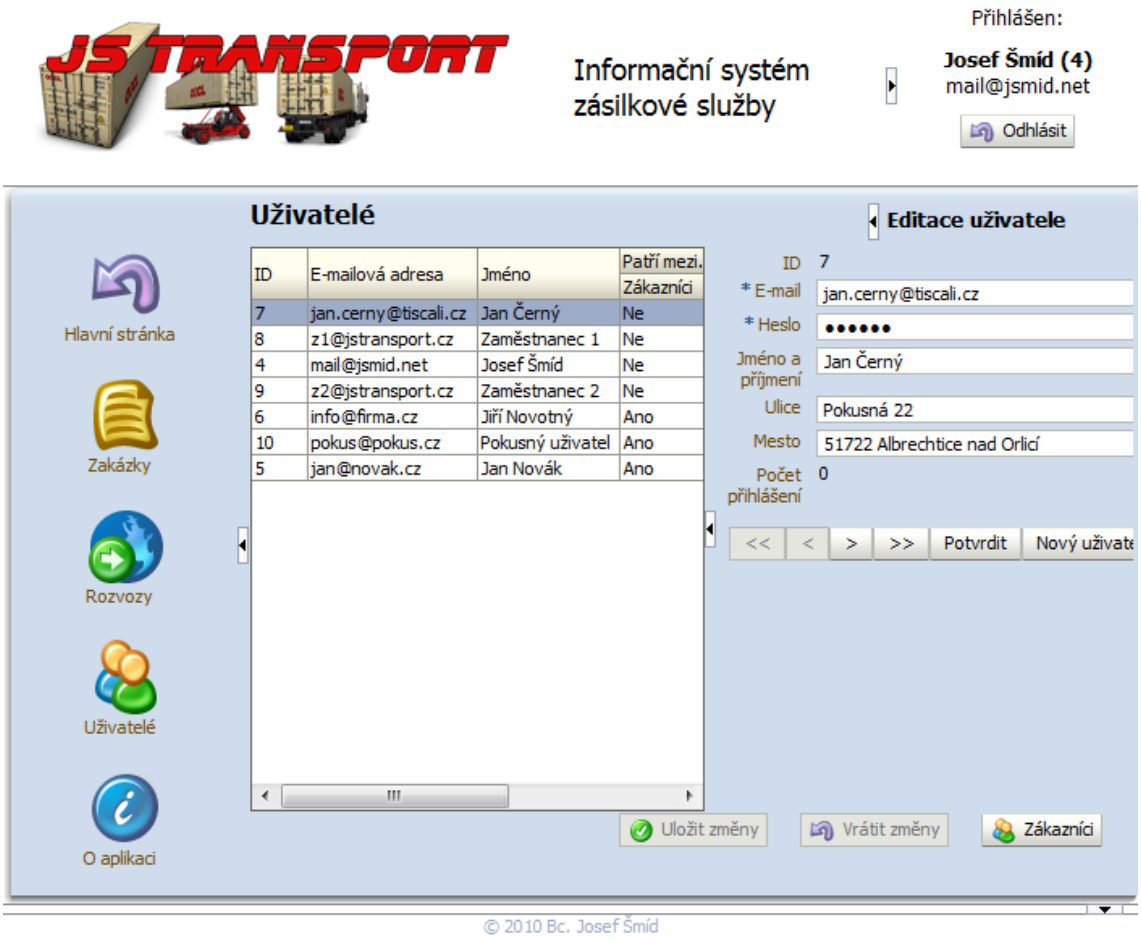
5.4. *Funkčnost aplikace*

Výsledkem praktické části této práce je fiktivní informační systém pro obsluhu procesů, které probíhají v zásilkové společnosti. Požadavkem pro vstup do systému je úspěšné přihlášení uživatelským jménem (zde e-mailová adresa zaměstnance) a heslem.

Pokud se uživatel pokusí zobrazit některou z částí systému, kam je vstup omezen, bude mu vypsána chybová zpráva a nabídnut odkaz na přihlašovací stránku.

Po úspěšném přihlášení se uživatel dostane do hlavního menu, kde bude moci vstoupit do jednotlivých částí systému. Na úvodní stránce je pouze ilustrační grafika a graf, který zobrazuje počty zakázek přiřazených do jednotlivých rozvozů (ukazuje vytíženost rozvozů). Ve všech částech systému je také zobrazeno jméno a identifikátor právě přihlášeného uživatele spolu s tlačítkem pro odhlášení.

První částí systému je menu Zakázky. Zde může zaměstnanec vyhledat požadovanou zakázku dle dostupných kritérií (jméno zakázky, stav zakázky nebo jméno zákazníka, kterému zakázka patří) a měnit její údaje. Po vstupu na editační stránku může také měnit stav zásilek dané zakázky a jejich údaje. Je možné přidávat nové zakázky včetně zásilek nebo mazat záznamy.



Obrázek 11 – Sekce Uživatelé
Zdroj: vlastní aplikace

V části Rozvozy jsou k dispozici informace o rozvozech a zásilkách, které jsou obsaženy v daném rozvozu. Zaměstnanec může naplánovat rozvoz a přidat do něj zásilky, editovat rozvozy nebo přiřazovat zásilky k jednotlivým rozvozům. Je zde také sekce pro správu automobilů, kde je možné přidávat, měnit nebo mazat automobily společnosti. Sekce automobily není provázaná se systémem, jedná se pouze o ukázku funkčnosti editačního formuláře a Task Flow.

Další částí systému jsou Uživatelé (viz Obrázek 11). Zde má zaměstnanec přehled o údajích jednotlivých uživatelů-zákazníků zásilkové společnosti. Může přidávat zákazníky nebo měnit jejich údaje.

Poslední položkou hlavního menu je O aplikaci, což je tzv. Popup okno, které zobrazí informace o dané aplikaci.

Ve všech částech systému je implementována obsluha transakcí, tzn. možnost uložit změny natrvalo nebo vrátit všechny změny do původního stavu. Tato funkčnost je založena na příkazech commit a rollback, které poskytuje databázový systém. V každé ze sekcí je implementován jeden ze dvou způsobů práce s transakcemi ve Fusion – buď přímo tlačítky navázanými na operace poskytované Data Controls, nebo přes Task Flow komponentu Return, která má nastavenu příslušnou hodnotu vlastnosti commit nebo rollback.

6. Implementace informačního systému pomocí Oracle ADF

V předchozí kapitole byl představen a podrobněji popsán výsledek praktické části práce. Následuje část, která je jednak doplněním předchozí kapitoly a jednak jakýmsi základním návodem, jak vytvořit podobnou Fusion aplikaci pomocí Oracle ADF. Jsou zde nastíněny hlavní kroky při tvorbě takové aplikace.

Jako součást této práce byl vytvořen podrobný tutoriál (návod), jak krok za krokem vytvořit stejnou aplikaci, která byla implementována v praktické části [13]. Tutoriál je doplněn tipy a obrázky, takže je možné podle něj postupovat a proniknout do možností Oracle ADF hlouběji na rozsáhlejším praktickém příkladu. Nemalou výhodou kromě rozsahu tutoriálu je i fakt, že je psán v českém jazyce, a to jako jediný nebo jeden z mála materiálů o Oracle ADF dostupných na Internetu (naprostá většina informací je v současné době v angličtině).

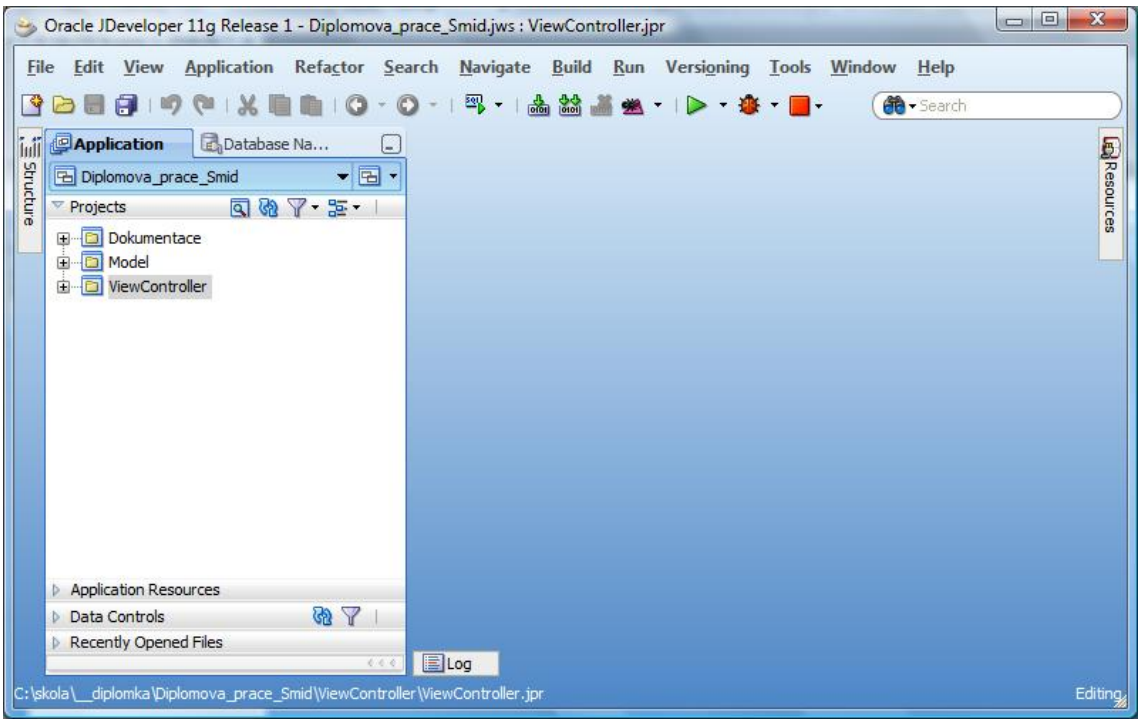
Pro vývoj aplikace je nutné mít nainstalován balík JDeveloper & ADF, který je ke stažení na webových stránkách společnosti Oracle pro platformy Windows nebo Unix. Instalace probíhá bez větších zásahů uživatele a zvládne ji i ten méně pokročilý.

Jelikož databáze využívá také produktu společnosti Oracle, je nutné nainstalovat nebo mít k dispozici tento databázový systém. Je možné využít verzi Oracle Database 10g Express Edition, která je nabízena zdarma. Ke stažení je také na webu Oracle. Po instalaci je vhodné zobrazit webové rozhraní databáze a vytvořit konkrétního uživatele pro tuto aplikaci (samostatné schéma).

Vývojové prostředí JDeveloper je součástí balíku Oracle Fusion Middleware. Jedná se o nástroj, který umožňuje kompletní vývoj Fusion aplikace – od prvotní analýzy, přes návrh, datový model, připojení k databázi, samotný vývoj, testování i tzv. deploying („publikaci“ aplikace pro produkční prostředí). Není tedy nutné používat pro některé fáze vývoje aplikace jiné nástroje, jak tomu bývá v některých případech u jiných vývojových prostředí.

Po spuštění JDeveloperu se zobrazí hlavní okno (viz Obrázek 12). To je rozděleno na několik důležitých částí. V horní části se nachází textové menu a nástrojová lišta s běžnými operacemi. V levé části je zobrazeno okno Application, kde je možné přepínat mezi jednotlivými otevřenými aplikacemi. Obsah aktuálně zvolené aplikace je pak

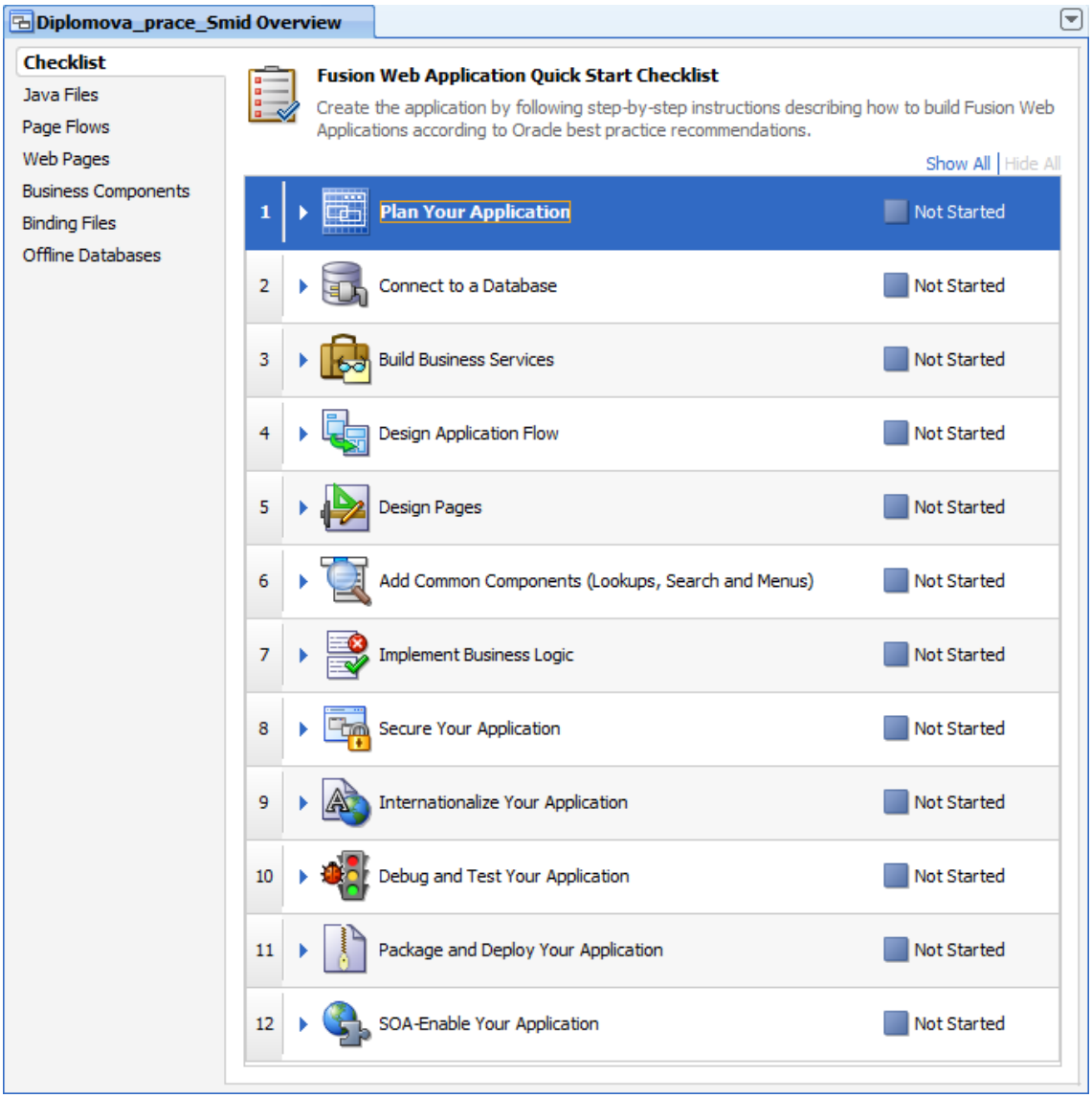
zobrazen také v této části. Po rozkliknutí seznamu aplikací je možné vytvořit novou nebo otevřít uloženou aplikaci. Druhým oknem skrytým pod záložkou je Database Navigator. Zde je možné vytvářet k jednotlivým aplikacím připojení do databáze a spravovat databázové objekty. Další okna a součásti vývojového prostředí se otevírají automaticky v případě potřeby, například okno Component Palette se zobrazí při otevření JSF stránky, JSF fragmentu nebo Task Flow a vždy obsahuje příslušné komponenty pro daný typ souboru.



Obrázek 12 – Hlavní okno vývojového prostředí JDeveloper
Zdroj: vlastní aplikace

Samotné implementaci Fusion aplikace musí předcházet založení aplikace v JDeveloperu. To lze provést ručně přidáním příslušných projektů a povolením vhodných technologií pro každý projekt (každý z projektů aplikace má specifický účel, a tedy používá jiné technologie). Druhá a lepší možnost je použití šablony. JDeveloper má připravenou šablonu s názvem „Fusion Web Application (ADF)“, která zajistí vytvoření a konfiguraci potřebných projektů a přípravu pracovního prostředí. Pomocí šablony jsou vytvořeny dva projekty – Model a ViewController. Pro lepší přehlednost je doporučeno vytvořit ještě třetí projekt Dokumentace, kam budou umístěny všechny soubory týkající se analýzy, návrhu a datového modelu, případně další soubory s poznámkami nebo odkazy.

Po úvodním kroku přípravy pracovního prostředí pro vývoj aplikace může začít samotný vývoj. JDeveloper obsahuje pro tento účel velice užitečnou funkci – tzv. Overview a hlavně její část Checklist (viz Obrázek 13). Tento nástroj se zobrazí automaticky po vytvoření Fusion aplikace nebo ho lze zobrazit či skrýt ručně v menu.



Obrázek 13 – Application Overview a Checklist
Zdroj: vlastní aplikace

První ze záložek funkce Overview je Checklist, který obsahuje 12 hlavních kroků, které by měl vývojář projít při tvorbě Fusion aplikace. U každého kroku lze pro lepší přehlednost nastavit fázi, v jaké se nachází – Not Started, In Process, Done nebo Will Not Do. Po rozbalení konkrétního kroku se zobrazí informace, jaké činnosti do něj patří a jakými kroky (Substeps) by měl vývojář pokračovat. Dále jsou zde také odkazy na nápovědu nebo rovnou některé průvodce.

Checklist samotný není nástrojem, který za vývojáře aplikaci vytvoří nebo ze špatné analýzy a návrhu udělá dobrý. Může ale vývojáři pomoci orientovat se v hodně rozsáhlé problematice Fusion architektury, sledovat s ním průběh implementace a pomoci mu navrhnout další postup a možnosti, jakým směrem při vývoji pokračovat. Následující kapitoly se věnují podrobněji jednotlivým krokům checklistu.

6.1. *Plan Your Application*

První krok checklistu je plánování aplikace. Tento krok zahrnuje úkony, které by měly být vykonány předtím, než se vývojář pustí do samotného psaní kódu a vytváření aplikace. Toto samozřejmě neplatí jen ve Fusion, mělo by se tímto krokem začínat vždy a před tvorbou jakékoli aplikace.

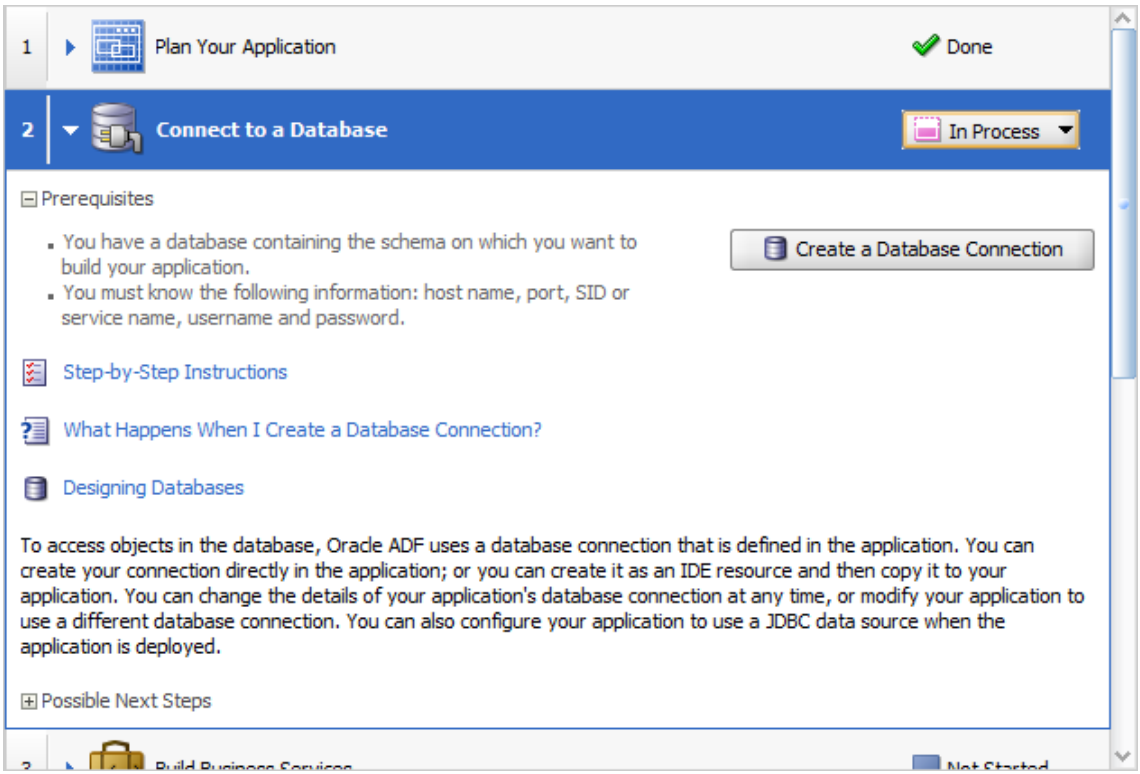
Předpokladem pro projití tohoto kroku je správně připravený pracovní prostor aplikace. JDeveloper obsahuje všechny důležité nástroje pro analýzu a návrh aplikace. V první řadě je nutné provést analýzu dané problematiky, zjistit a setřídit požadavky, sestavit případy užití a vytvořit analytický model tříd. Pro toto je k dispozici kompletní sada UML diagramů včetně všech vhodných komponent. Jasnou demonstrací jsou přílohy A, B a C této práce, které byly vytvořeny pomocí vhodných diagramů v JDeveloperu. Další možnosti zahrnují sekvenční diagram nebo diagram aktivit.

Dalším krokem, který následuje po analýze, je návrh – vytvoření datového modelu aplikace. Opět je k dispozici příslušná sada komponent a diagram. Specialitou je tzv. offline databáze, která umožňuje vytvořit jednotlivé databázové objekty (nejen tabulky, ale i sekvence, triggery a další) do zvláštního adresáře, a to i bez funkčního připojení k databázi. Po připojení k databázi je možné tyto objekty z offline databáze exportovat buď přímo do skutečné databáze, nebo do SQL skriptu. Funguje také obousměrné propojení s ER diagramem – z databázových objektů lze vytvořit příslušný diagram a naopak.

6.2. *Connect to a Database*

Druhým krokem checklistu je připojení ke skutečné databázi. Jedná se o nejrychlejší krok ze všech dvanácti. Předpokladem pro to je ale funkční a běžící databáze. Pak stačí jen nastavit parametry připojení a tento krok je splněn. Obrázek 14 uka-

zuje jeden z případů, jak checklist zjednodušuje práci vývojáři. Pod krokem připojení do databáze je rovnou k dispozici tlačítko, které spustí průvodce nakonfigurováním připojení. Není nutné ho hledat v menu, tato cesta je mnohem rychlejší.



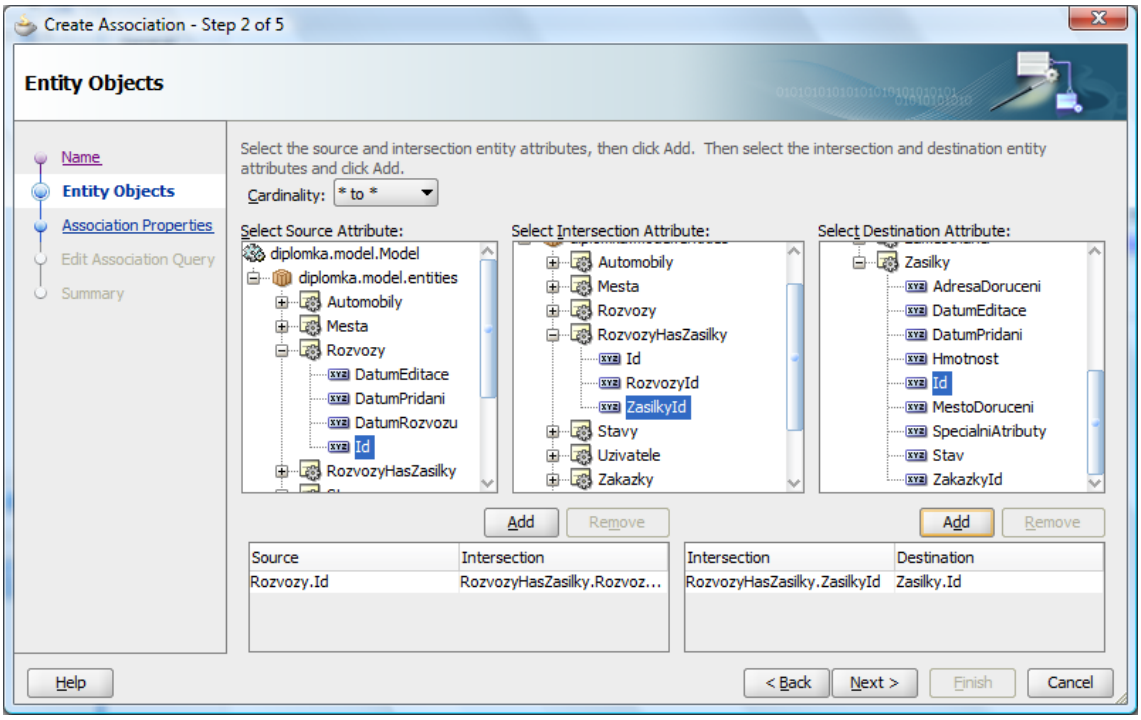
Obrázek 14 – Přehled nápovědy a možností ve druhém kroku checklistu
Zdroj: vlastní aplikace

6.3. *Build Business Services*

Třetím krokem v checklistu je vytvoření a konfigurace tzv. Business Components. Ty představují klíčové komponenty Fusion aplikace na vrstvě model. Obstarávají totiž přenos požadovaných dat z databáze směrem k uživateli a také naopak zajišťují aktualizaci dat v databázi. Business komponent je několik typů. Entity Objects představují jednotlivé řádky s daty v tabulkách. View Objects reprezentují SQL dotazy. Associations představují vztahy mezi entity objekty a View Links definují vztahy mezi view objekty.

Předpokladem pro absolvování tohoto kroku je hotová analýza a vytvořený datový model ve funkční databázi. Vytvoření základních business komponent je pak v prostředí JDeveloper velice snadné. Probíhá pomocí průvodce, který z databázových tabulek vytvoří příslušné entity objekty včetně asociací a view objekty včetně view linků. Jako další součást je vytvořen tzv. Application Module, což je modul, ve kterém je

jednak možné testovat správnou funkčnost business komponent, ale hlavně poskytuje tzv. Data Controls vrstvu (viz kapitola o Oracle ADF), pomocí které bude v jednom z dalších kroků možné vytvářet tabulky a formuláře pro práci s daty.



Obrázek 15 – Vytváření business komponenty Association pro relaci M:N
Zdroj: vlastní aplikace

Business komponenty je samozřejmě možné vytvářet i ručně. Je ale nutné znát dobře principy a účel jednotlivých komponent tak, aby došlo k jejich správné spolupráci. Průvodce, pomocí kterého lze z databázových tabulek vytvořit business komponenty automaticky, umí vytvořit pouze objekty k relacím 1:N, tedy tzv. master-detail vztahy. Například pro práci s M:N vztahy je nutné přidat příslušné business komponenty ručně. Ale i pro tento účel je JDeveloper vybaven průvodci, pomocí kterých je tvorba o mnoho snazší. Příklad takového průvodce přináší Obrázek 15.

6.4. Design Application Flow

Tento krok představuje posun od vrstvy model k vrstvě controller. Hlavním úkolem je navrhnout tzv. application flow, tedy způsob, jakým budou jednotlivé stránky aplikace propojeny. Na základě application flow jsou pak vytvářeny konkrétní stránky a nastavováno jejich propojení. Základním „stavebním kamenem“ je Task Flow (TF), který realizuje pomocí jednotlivých objektů navigaci příslušné části aplikace (příklad

TF ukazuje Obrázek 8). Celý application flow je tedy sestaven z jednoho nebo několika TF. Více se o TF zmiňuje jedna z předchozích kapitol.

Tento krok je sice až čtvrtý v pořadí, ale lze na něm začít pracovat o mnoho dříve. Předpokladem pro jeho řešení je totiž „pouze“ představa o navigaci mezi jednotlivými stránkami a částmi aplikace. Pokud pracuje na jedné Fusion aplikaci dva nebo více vývojářů, první může řešit vrstvu Model a business komponenty a druhý může současně začít navrhovat application flow.

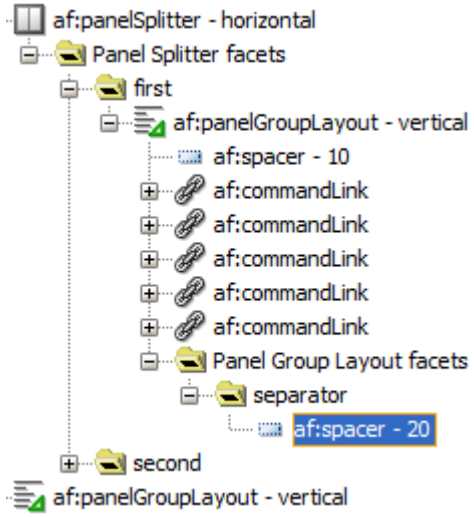
6.5. Design Pages

Pátým krokem nastává přesun do fáze počátku návrhu uživatelského rozhraní, tedy do vrstvy view. Krok s názvem „Design Pages“ checklistu obsahuje vytvoření šablony vzhledu aplikace, vytvoření hlavní JSF stránky a také vytvoření jednotlivých stránek aplikace. Tyto stránky jsou vytvářeny jako tzv. Page Fragments, tedy fragmenty, které se mění v určité, předem definované oblasti hlavní JSF stránky. Patří sem i trocha programování – práce s tzv. Backing Beans, což jsou známé javovské třídy, které poskytují požadovanou funkčnost. V praktické části této práce obsluhují tyto backing třídy hlavní menu aplikace a také přihlašování uživatelů.

Informační systém jako většina webových aplikací využívá záhlaví a zápatí společné pro všechny stránky. Také vzhled (skin) aplikace je na všech stránkách stejný. Proto se nabízí využití tzv. Page Template, tedy šablony, kterou budou všechny stránky webu využívat. Taková šablona je sama JSF stránkou, která obsahuje komponenty a prvky společné pro všechny stránky. V šabloně je definována oblast (tzv. facet), kde se zobrazí stránka, která šablonu využívá. Je také možné přidat volitelné parametry (například hlavní nadpis), jejichž hodnoty se do šablony dosadí až zobrazením příslušné stránky.

Konkrétní stránky a fragmenty je nejjednodušší vytvořit využitím navržených Task Flow. Přes jednotlivé objekty View konkrétního TF se v prostředí JDeveloper zobrazí přímo průvodce pro vytvoření nové stránky nebo fragmentu (v případě, že se jedná o Unbounded TF, je vytvořena JSF stránka; v případě Bounded TF je vytvořen JSF fragment).

Návrh stránky probíhá pomocí vizuálního editoru vkládáním jednotlivých komponent z okna Component Palette. Komponenty jsou rozděleny do tří skupin – Common Components, Layout a Operations. První jmenovaná obsahuje běžné vizuální komponenty, jako je například tlačítko, textové pole, obrázek atd. Layout obsahuje komponenty, které formátují stránku a umístění jednotlivých komponent do nich vložených. Ekvivalentem těchto „layout komponent“ v HTML by byl DIV s příslušnými CSS vlastnostmi. Protože vizuální programování není vždy přehledné, JDeveloper nabízí okno Structure, kde jsou hierarchicky zobrazeny jednotlivé komponenty vložené v editované stránce nebo fragmentu. Část okna ukazuje Obrázek 16.



Obrázek 16 – Hierarchická struktura komponent stránky v JDeveloperu
Zdroj: vlastní aplikace

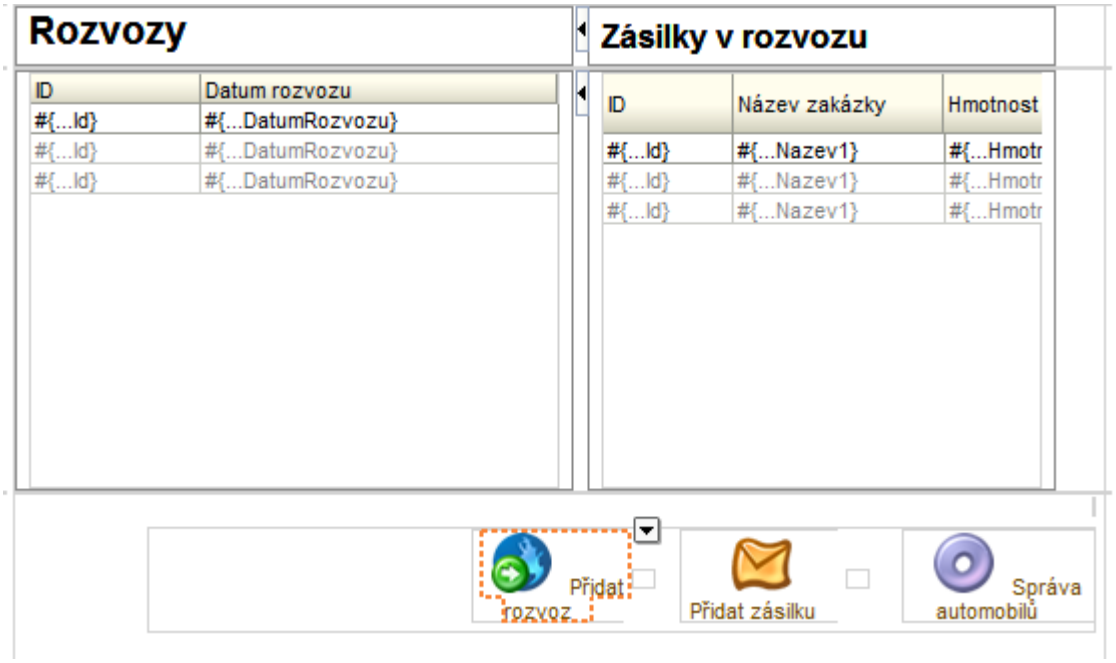
6.6. *Add Common Components*

Šestý krok checklistu je pokračováním kroku předchozího. Jedná se o velice důležitý krok, který posouvá možnosti vyvíjené aplikace o velký kus dále. Čím lépe jsou z předchozích kroků navržené a nakonfigurované business komponenty, tím jednodušší práci má vývojář zde.

Pomocí business komponent je možné vložit jednotlivé formuláře, tabulky, grafy a jiné komponenty pro práci s daty v databázi. Důležitá je zde část Data Controls (umístěná v JDeveloperu v okně Application Navigator), která obsahuje business logiku vytvořených komponent a umožní nám vytvořit formuláře a tabulky.

Takto vytvořené komponenty poskytují bohaté možnosti konfigurace – od vzhledu přes formát až po chování. Je možné propojit více komponent tak, že při změně

obsahu jedné se aktualizuje obsah druhé apod. Typickým propojením je tzv. master-detail vztah mezi tabulkou a editačním formulářem. Výběrem konkrétního řádku v tabulce se do formuláře načtou údaje vybraného řádku, a je tak možné záznam editovat. Časté je také použití dvou tabulek zobrazujících vztah 1:N. Výběrem řádku v jedné tabulce se do druhé načtou příslušné řádky z propojené business komponenty (příklad uvádí Obrázek 17 – rozvozy a zásilky ve vybraném rozvozu).



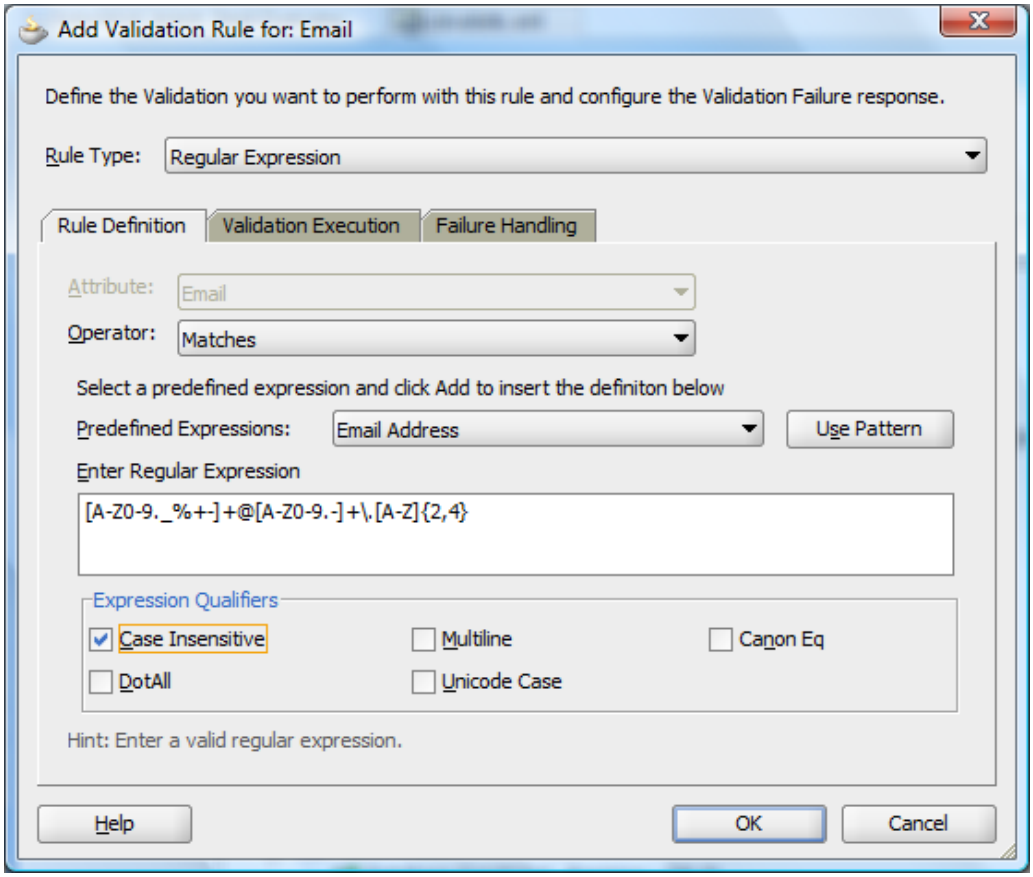
Obrázek 17 – Stránka editovaná v JDeveloperu obsahující dvě komponenty Table
Zdroj: vlastní aplikace

Formuláře standardně poskytují možnost vytvoření včetně navigačních tlačítek pro pohyb mezi záznamy databázové tabulky a tlačítko pro potvrzení změn. Přes Data Controls lze přidávat i další tlačítka poskytující další možnosti práce s daty – přidávání nových záznamů, mazání a již zmiňovaný commit a rollback.

6.7. Implement Business Logic

Před započetím tohoto kroku je nutné mít splněné všechny předchozí, tedy mít v podstatě navrženou aplikaci včetně všech business komponent, Task Flows a stránek. Hlavní úlohou tohoto kroku je již podle názvu implementace business logiky. V první řadě je nutné říci, že se jedná o finální úpravu vytvořených business komponent. To znamená návrat k vrstvě model.

Zmíněná business logika představuje tzv. validátory. Různá pole ve formulářích totiž můžou mít různá omezení. Typickým příkladem je minimální délka hesla, správný tvar e-mailové adresy a mnoho dalších. Při porušení tohoto omezení musí být uživatel o takové skutečnosti informován. K tomu slouží právě validátory, které se nastavují pro jednotlivé atributy business komponent, konkrétně entity objektů. Příklad vytvoření validátoru pro kontrolu správného tvaru e-mailové adresy ukazuje Obrázek 18.



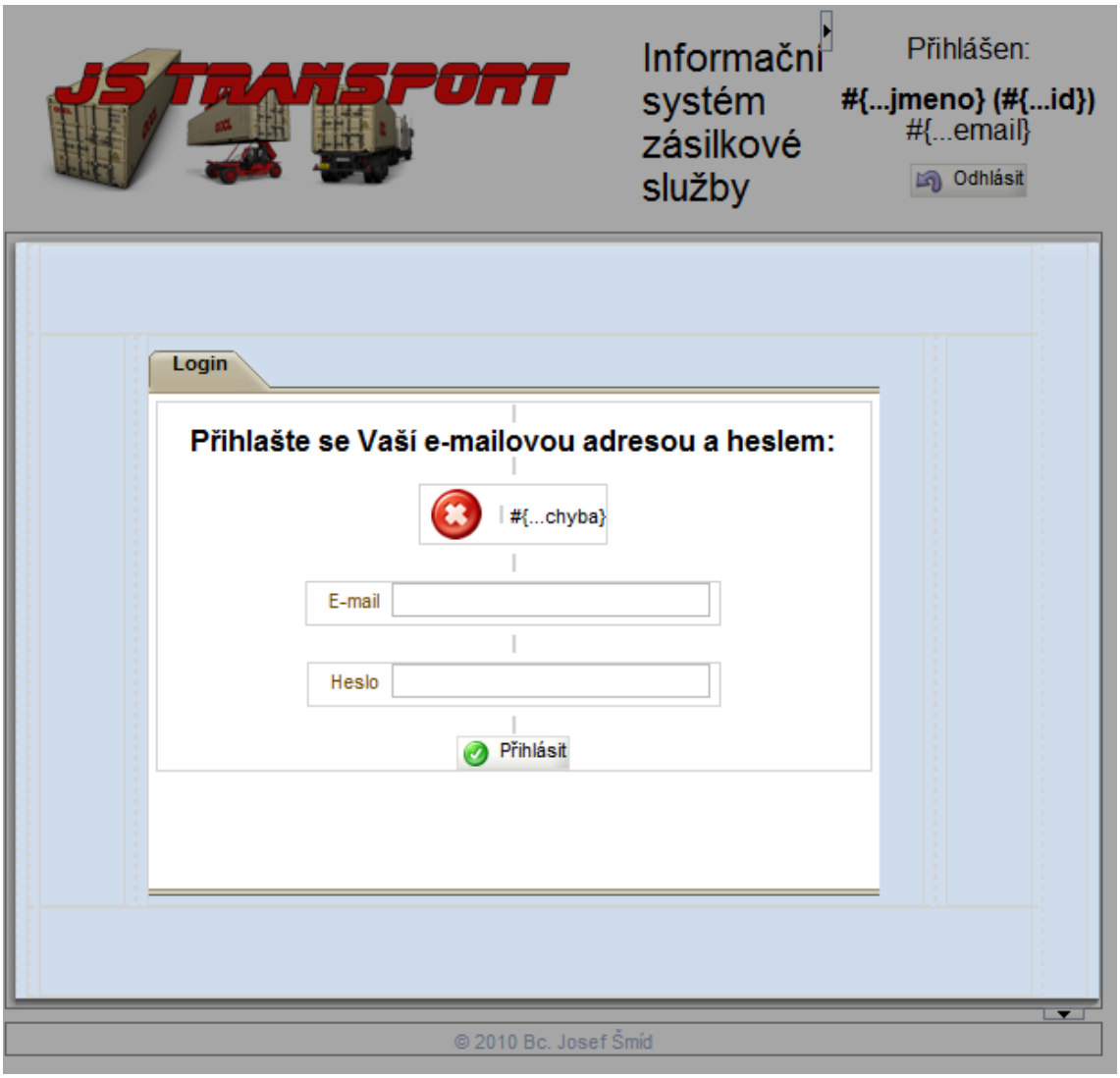
Obrázek 18 – Dialog pro vytvoření nového validátoru
Zdroj: vlastní aplikace

Některé validátory jsou již v příslušných business komponentách obsaženy. Byly nastaveny při samotném vytváření komponent, a to na základě integritních omezení databázových tabulek. Běžným příkladem je validátor založený na omezení NOT NULL nebo PRIMARY KEY – hodnota atributu nesmí být nulová, je tedy povinná. Dalším příkladem je maximální délka hodnoty atributu vycházejícího z databázového sloupce s datovým typem VARCHAR2 omezené délky.

6.8. *Secure Your Application*

Oracle ADF standardně poskytuje nástroje pro zabezpečení Fusion aplikace. Jsou založeny na vytvoření uživatelských rolí a konkrétních uživatelů WebLogic serveru a přiřazení oprávnění jednotlivým uživatelům (nebo skupinám) ke konkrétním Task Flow. Tím je umožněn přístup k aplikaci nebo jejím částem pouze oprávněným uživatelům. Vše opět probíhá pomocí dialogů a průvodců.

Ne vždy je ale takovéto zabezpečení vhodné. Je totiž nutné předem znát produkční prostředí, ve kterém vyvíjená aplikace poběží. ADF Security je totiž přizpůsobena pro webový server WebLogic, který je součástí instalačního balíku OFM. V případě použití tohoto webserveru i v ostrém nasazení je možné snadno nadefinovat jednotlivé uživatelské role a skupiny včetně oprávnění k jednotlivým stránkám aplikace.



Obrázek 19 – Návrh stránky login.jspx
Zdroj: vlastní aplikace

V případě praktické části této práce – informačního systému – není tato funkčnost využita. Místo toho je implementována autorizace a autentizace na základě údajů v databázové tabulce UZIVATELE a ZAMESTNANCI přes pohled ZAMESTNANCI_VIEW. Vše obsluhuje Backing Bean zmíněná v jedné z předchozích kapitol. Na základě této funkčnosti jsou také zabezpečeny jednotlivé stránky proti přístupu nepřihlášených uživatelů. Návrh stránky login.jspx představuje Obrázek 19.

6.9. *Internationalize Your Application*

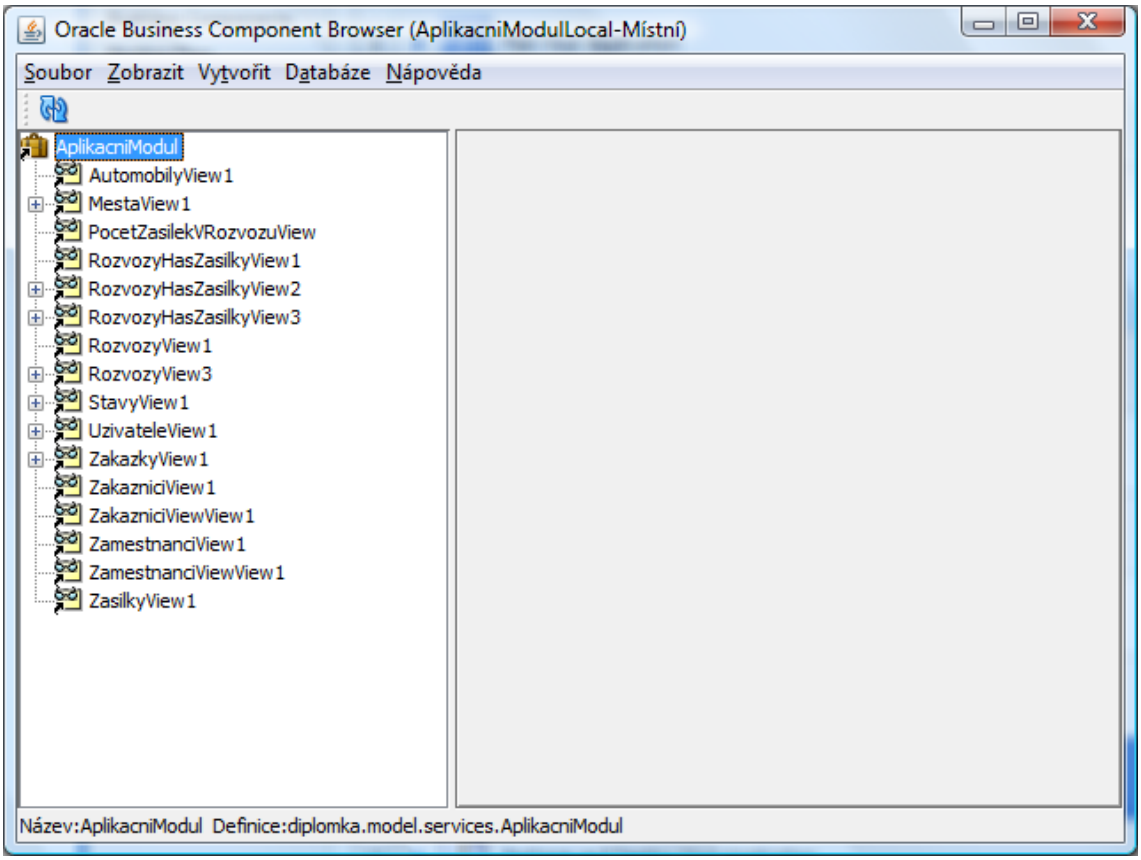
Oracle ADF poskytuje nástroj pro „internacionalizaci“ vyvíjené aplikace, tedy pro konfiguraci vhodnou pro překlad do cizích jazyků. Pro realizaci této funkce je nutné nakonfigurovat tzv. ResourceBundle, což je objekt obsahující všechny texty, které budeme chtít překládat do jiných jazyků. Pomocí Backing Bean pak probíhá načítání správných hodnot pro jednotlivé texty.

Jedná se o nepovinný krok – ne ve všech aplikacích bude požadován překlad do jiných jazyků. Ve vzorové aplikaci Informační systém zásilkové služby není tato funkčnost realizována.

6.10. *Debug and Test Your Application*

Desátým krokem checklistu je velice důležitá část vývoje aplikace – testování. Tento krok vlastně prolíná celou tvorbu aplikace počínající tvorbou databáze. Je velice vhodné vyvíjenou aplikaci průběžně testovat, a tak předejít pozdějším a zdánlivě záhadným chybám nebo chování, které by vzniklo v důsledku skrytí drobné chyby někde uvnitř aplikační logiky.

Ve Fusion je k dispozici již dříve zmíněný aplikační modul, který nabízí vývojářům možnost testování business komponent ještě před jejich použitím v konkrétních stránkách nebo fragmentech. Automaticky vytvářené business komponenty jsou do něj vkládány rovnou, ale je možné je kdykoli jakkoli nastavit či přidat ručně vytvořené komponenty. Obrázek 20 ukazuje příklad aplikačního modulu se seznamem vytvořených business komponent (view objektů).



Obrázek 20 – Aplikační modul sloužící k testování business komponent
Zdroj: vlastní aplikace

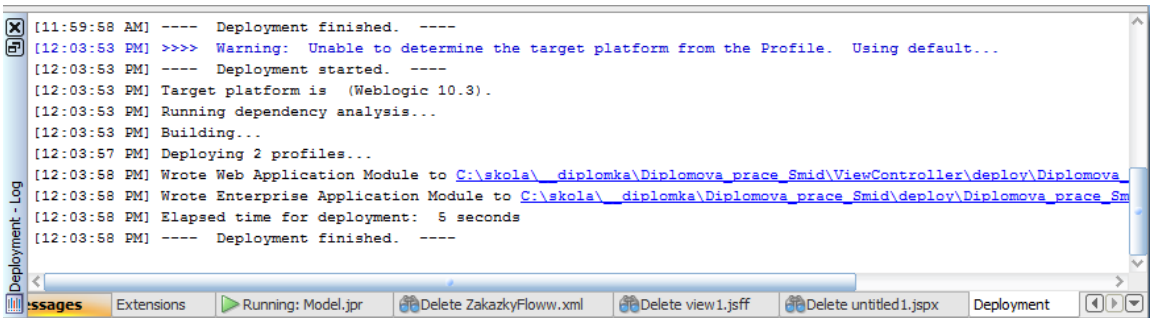
Dalším a klasickým způsobem testování je spuštění rozpracované aplikace a testování jednotlivých stránek, komponent a formulářů. Pro tuto činnost nabízí JDeveloper vestavěný server WebLogic, který je přímo nakonfigurován pro testovací spouštění vyvíjených aplikací. Kromě toho nabízí sadu nástrojů pro debugování (tedy hledání a odstraňování chyb) aplikace.

6.11. Package and Deploy Your Application

Závěrečným krokem vytvořené aplikace je její kompilace a případné vytvoření instalačního balíku nebo rovnou přenesení do produkčního prostředí. V Javě se tento krok nazývá deploying. Jedná se tedy o „publikování“ vytvořené aplikace do produkčního prostředí. Důležitým předpokladem je pochopitelně vytvořená, odladěná a funkční Fusion aplikace.

JDeveloper nabízí dvě možnosti, jak provést deploying aplikace. První z nich je vytvoření instance webového serveru WebLogic a deploying aplikace rovnou do tohoto serveru. Probíhá pomocí průvodce, ve kterém je vše potřebné nastaveno. Druhou mož-

ností deployingu je vytvoření EAR archivu, což je soubor obsahující vše potřebné pro instalaci a běh aplikace v jiném prostředí. Obrázek 21 ukazuje výsledek deployingu do EAR archivu.



Obrázek 21 – Log s informacemi o úspěšném deployingu aplikace do EAR archivu
Zdroj: vlastní aplikace

6.12. SOA-Enable Your Application

Fusion aplikace může volitelně pracovat v tzv. Service-oriented prostředí, tedy v prostředí využívajícím funkčnosti webových služeb. Business komponenty mohou být nastaveny tak, že volají externí procesy. Aplikační modul může poskytovat rozhraní pro využívání webových služeb. Také je možné volat vzdálené webové služby pomocí aplikačního modulu.

To vše je možné nakonfigurovat v prostředí JDeveloper. Aplikace pracující jako webové služby nebo využívající webových služeb jsou specifickými případy, a proto je tento krok nepovinný. V praktické části této práce – informačním systému zásilkové služby – nebyl využit.

7. Závěr

Oracle Application Development Framework je mocným a komplexním nástrojem, který přináší odlišný přístup a nové možnosti do vývoje enterprise aplikací. Stejně jako každý jiný kvalitní framework poskytuje mnoho připravených a předdefinovaných nástrojů a postupů, které vývojářům usnadňují základní činnosti při tvorbě aplikací, se kterými by jinak ztratili mnoho drahocenného času. Oracle ADF jde ještě dále a nabízí pokročilé průvodce, pomocí kterých lze relativně jednoduše vystavět rozsáhlou a moderní Fusion aplikaci.

Pro vývojáře zvyklého na klasický způsob vývoje aplikací formou psaní zdrojového kódu bude průběh vytváření Fusion aplikace ve vývojovém prostředí JDeveloper asi zpočátku velkou změnou – velká část vývoje totiž probíhá formou dialogových a konfiguračních oken a samotného psaní zdrojového kódu je minimum. Každopádně u ADF platí heslo „za vším hledej XML“ – na tomto jazyku je založena většina komponent a konfiguračních souborů. Vždyť i samotné JSF stránky jsou založeny na XML. Proto lze vždy přepnout na záložku zdrojového kódu dané stránky nebo konfiguračního souboru a ručně změnit či přidat požadovanou hodnotu. Pokud ale vývojář pronikne do základních principů a postupů tvorby Fusion aplikace, bude do XML kódu potřebovat zasáhnout jen minimálně a bude při vývoji postupovat až neobvykle rychle právě pomocí standardních nástrojů ADF.

Tato práce měla za cíl přednést základní informace o ADF architektuře a také o vizi „Fusion“, tedy tvorbě moderních enterprise aplikací podle představy společnosti Oracle. Jak teoretická, tak i praktická část ověřily, že vývoj aplikace pomocí Oracle ADF má svá specifika, ale při pochopení základních procesů a postupů jde vývoj poměrně snadno. Všemu napomáhá i komplexní vývojové prostředí JDeveloper, díky kterému lze celý vývoj realizovat a není nutné hledat externí nástroje pro určité činnosti.

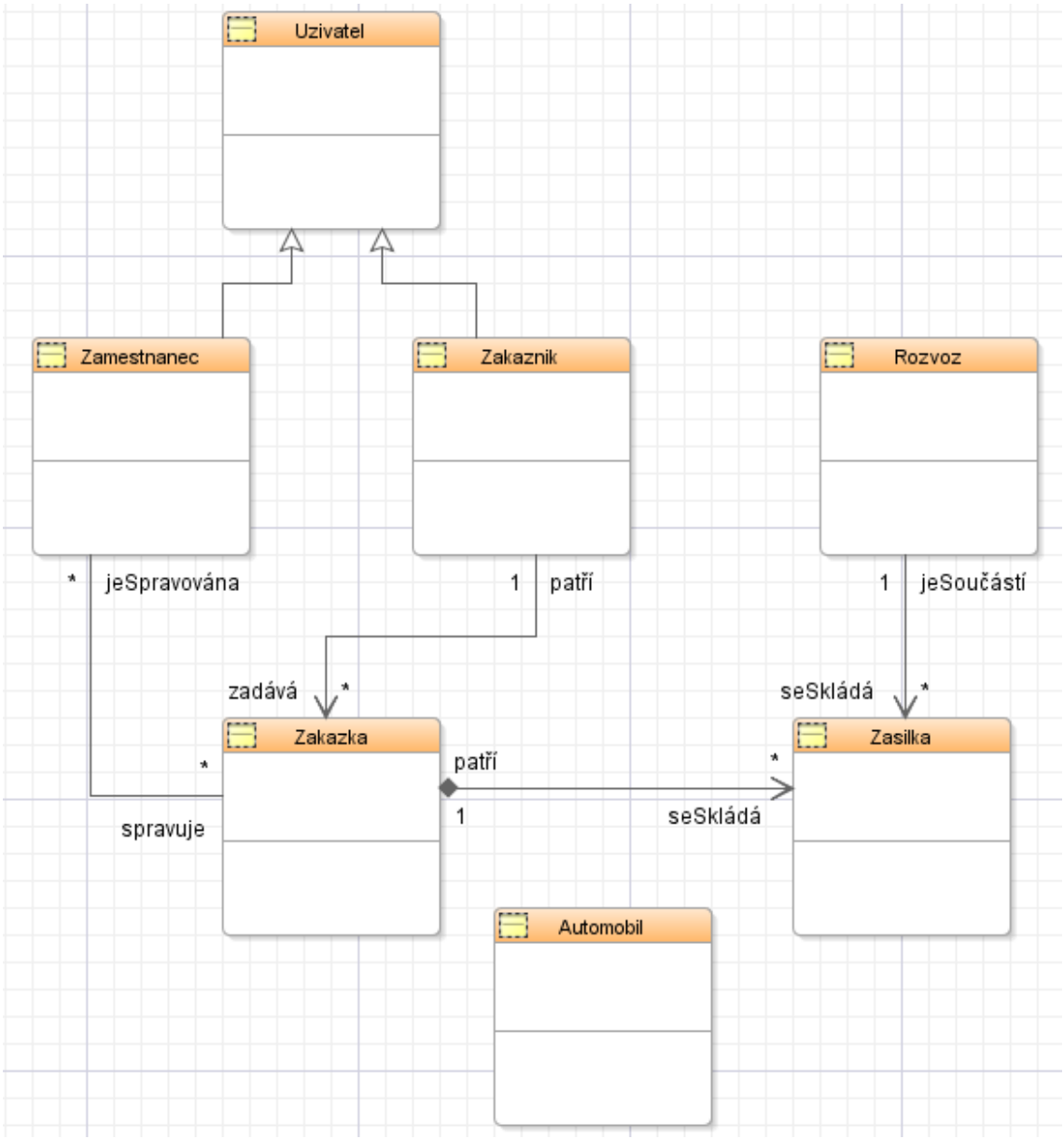
Hlavně praktická část spolu s podrobným tutoriálem pro vytvoření takové aplikace snad pomůže zájemcům seznámit se s možnostmi a postupy v ADF a nasměruje je k dalším samostatným pokrokům ve vývoji Fusion aplikací.

Zdroje informací

1. *Wikipedie, otevřená encyklopedie* [online]. 2010 [cit. 2010-05-14]. Informační systém. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Informační_systém>.
2. *Wikipedie, otevřená encyklopedie* [online]. 2010 [cit. 2010-05-14]. Aplikační software. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Aplikační_software>.
3. *Wikipedie, otevřená encyklopedie* [online]. 2010 [cit. 2010-05-14]. Framework. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Framework>.
4. *MSDN: Microsoft Development* [online]. c2010 [cit. 2010-05-14]. Model-View-Controller. Dostupné z WWW: <http://msdn.microsoft.com/en-us/library/ms978748.aspx>.
5. *Java EE at a Glance* [online]. c2010 [cit. 2010-05-14]. Dostupné z WWW: <http://java.sun.com/javaee/>.
6. *Java SE at a Glance* [online]. c2010 [cit. 2010-05-14]. Dostupné z WWW: <http://java.sun.com/javase/>.
7. JOHNSON, Mark. *JavaWorld* [online]. 10. 1. 1998 [cit. 2010-05-14]. A beginner's guide to Enterprise JavaBeans. Dostupné z WWW: <http://www.javaworld.com/javaworld/jw-10-1998/jw-10-beans.html?page=1>.
8. KOSEK, Jiří. *Domovská stránka Jirky Koska* [online]. c1999 [cit. 2010-05-14]. XML. Dostupné z WWW: <http://www.kosek.cz/clanky/xml/xml-uvod.html>.
9. SNÍŽEK, Martin. *Snizekweb.cz* [online]. 13. 9. 2005 [cit. 2010-05-14]. AJAX - kde jsou hranice?. Dostupné z WWW: <http://www.snizekweb.cz/clanky/ajax-kde-jsou-hranice/>.
10. ŠMÍD, Josef. *Školní informační systém*. Pardubice, 2007. 53 s. Bakalářská práce. Univerzita Pardubice.
11. *Oracle* [online]. c2010 [cit. 2010-05-14]. Oracle Application Development Framework. Dostupné z WWW: <http://www.oracle.com/technology/products/adf/index.html>.
12. AKEL, Laura. *Oracle ADF 11g Primer* [online]. Redwood Shores, USA : Oracle, Duben 2007 [cit. 2010-05-14]. Dostupné z WWW: <http://www.oracle.com/technology/products/jdev/11/how-tos/oracle adf 11g primer.pdf>.
13. ŠMÍD, Josef. *Oracle ADF 11g : Tvorba ukázkové Fusion aplikace* [online]. Pardubice : Josef Šmíd, Květen 2010 [cit. 2010-05-14]. Dostupné z WWW: <http://diplomka.jsmid.net/tutorial/Oracle ADF 11g TUTORIÁL.pdf>.
14. MILLS, Duncan; KOLETZKE, Peter; ROY-FADERMAN, Avrom. *Oracle JDeveloper 11g Handbook*. USA : McGraw-Hill, 2010. 864 s. ISBN 978-0-07-160238-9.

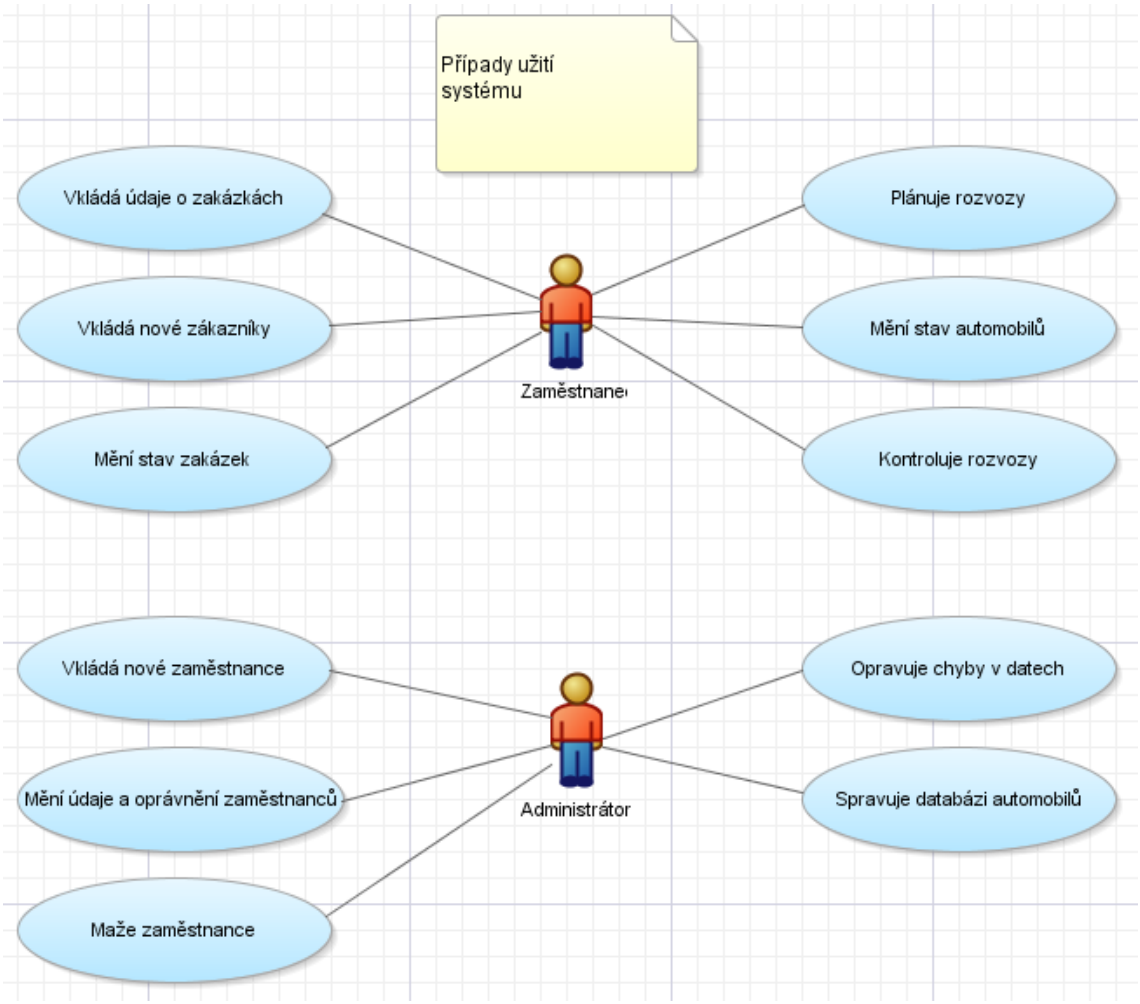
Seznam obrázků a příloh

Příloha A



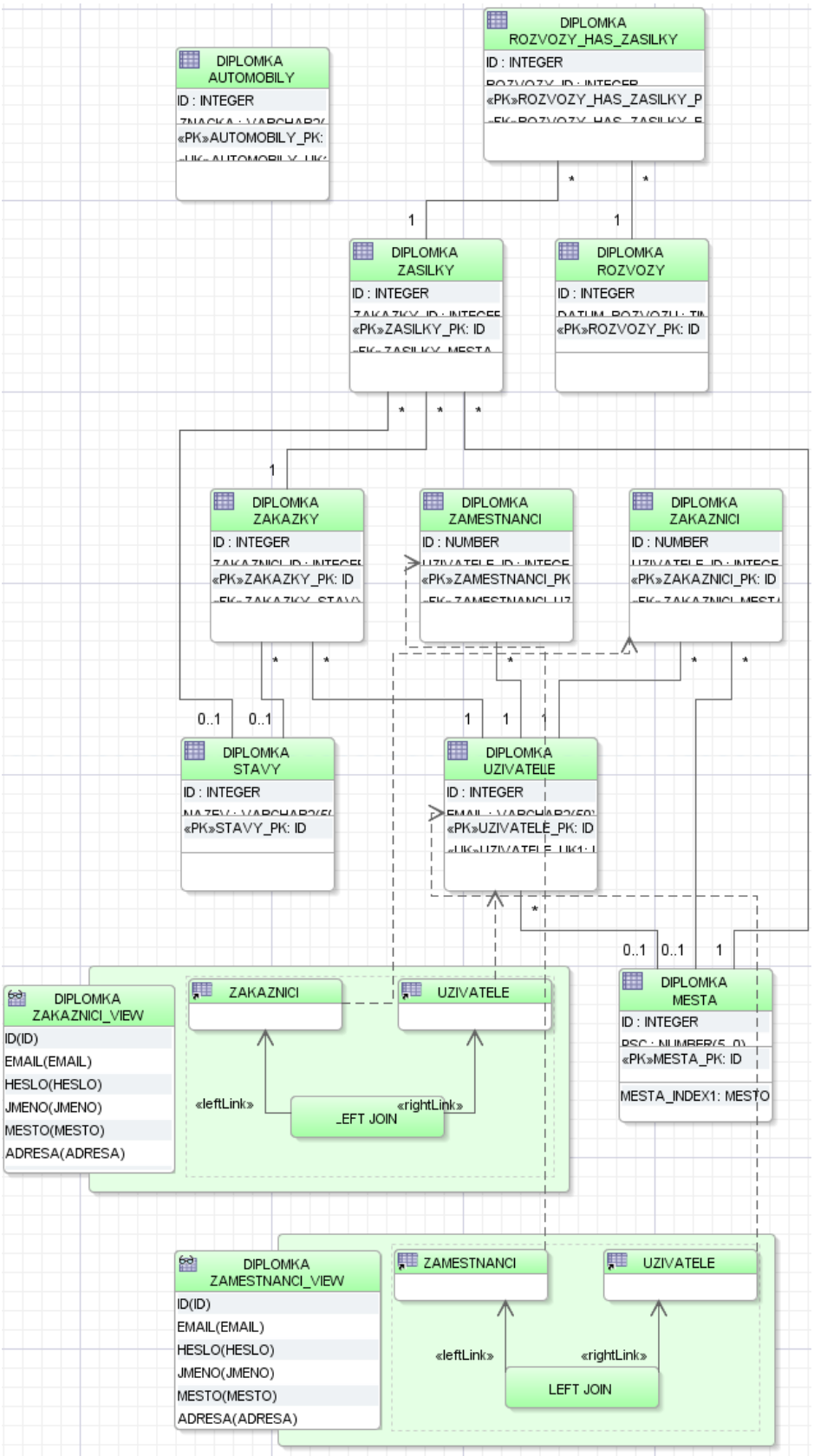
Příloha A – Analytické třídy

Příloha B



Příloha B – Případy užití

Příloha C



Příloha C – Datový model

Údaje pro knihovnickou databázi

Název práce	Informační systém zásilkové služby pomocí Oracle ADF
Autor práce	Bc. Josef Šmíd
Obor	Informační technologie
Rok obhajoby	2010
Vedoucí práce	Ing. Zdeněk Šilar
Anotace	Tato diplomová práce se zabývá aplikačním frameworkem Oracle ADF. V první řadě popisuje a seznamuje čtenáře s technologiemi, které daný framework používá. Dále je zmíněn a vysvětlen princip model-view-controller a jeho aplikace právě pomocí Oracle ADF. Samotný framework, práce s ním a jeho možnosti jsou popsány v další kapitole. Pro tento účel byla naprogramována aplikace „Informační systém zásilkové služby“, na které jsou možnosti frameworku využity. Tato kapitola je doplněna podrobným průvodcem – tutoriálem, který krok za krokem popisuje tvorbu aplikace. Zároveň je představen nástroj pro tvorbu aplikací pomocí Oracle ADF, kterým je vývojové prostředí Oracle JDeveloper.
Klíčová slova	Šmíd, Josef, informační systém, informační systém zásilkové služby, programovací jazyk Java, jazyk Java, Java, databázový systém Oracle, Oracle, Oracle Database, Java platforma, Java EE, Oracle ADF, Application Development Framework, AJAX, architektura MVC, Business Components, EJB, JSP, JSF, Task Flow, tutoriál, průvodce, návod