VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

JAVA EE ORGANIZÉR – MODUL ÚKOLY

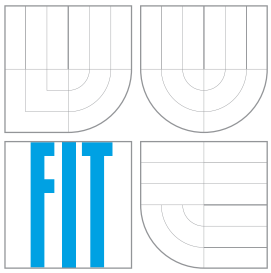
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR ČERNÝ

BRNO 2009

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

JAVA EE ORGANIZÉR – MODUL ÚKOLY

JAVA EE ORGANIZER – TASKS MODULE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR ČERNÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. RADEK KOČÍ, Ph.D.

BRNO 2009

Abstrakt

Tato práce se zabývá problematikou aplikací pro orgranizaci času a kontaktů. Obsahuje analýzu stávajících řešení, používaných architektur a stručnou charakteristiku konkrétních aplikací. Práce se také zabývá tvorbou aplikačního klienta pro Java EE aplikace a pojednává o základních technologiích, které se v této oblasti používají (Swing, platforma Java EE a její technologie). Praktická část obsahuje návrh a implementaci modulů pro správu kontaktů a úkolů, které jsou zakomponovány do demonstrační aplikace Java EE Organizér.

Abstract

This thesis deals with applications for organization of time and contacts. It includes analysis of existing solutions, used architectures and brief characteristics of existing applications. Thesis also deals with developing a client application for Java EE applications, and discusses the basic technologies that are used in this field (Swing, Java EE platform and its technologies). The practical part includes design and implementation of modules for managing contacts and tasks, which are incorporated into the demo application Java EE Organizer.

Klíčová slova

Java, Java EE, Java Enterprise Edition, síťová aplikace, databázová aplikace, organizér, modul úkoly

Keywords

Java, Java EE, Java Enterprise Edition, network application, database application, organizer, tasks module

Citace

Petr Černý: Java EE Organizér – modul úkoly, bakalářská práce, Brno, FIT VUT v Brně, 2009

Java EE Organizér – modul úkoly

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Radka Kočího, Ph.D. a že jsem uvedl všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Černý
20. května 2009

Obsah

Kapitola 1

Úvod

Tento dokument vznikl jako technická zpráva k bakalářské práci. Tato práce byla vypracována na téma návrhu a implementace síťové aplikace Java EE Organizér (síťová aplikace pro organizaci času a kontaktů). Jedná se o týmovou práci, kterou sdílím s kolegy Janem Kovářem[8] a Pavlem Palátem[9].

Ve druhé kapitole bude popsána analýza stávajících aplikací. Dozvíte se něco o základních architekturách těchto aplikací, jejich výhodách a nevýhodách. V závěru kapitoly bude zmíněna stručná charakteristika konkrétních, často využívaných, systémů.

Ve třetí kapitole se dočtete o základních technologiích, které byly použity pro vývoj a běh aplikace. Bude zde blíže popsán výčet základních technologií platformy Java EE (viz 3.2), popis Swing API (viz 3.5), charakteristika aplikačního serveru GlassFish a SŘBD Firebird.

Ve čtvrté kapitole naleznete základní návrh aplikace pro moduly kontaktů a úkolů. Zaměříme se především na návrh uživatelského prostředí prezentační vrstvy (viz 4.2).

V páté kapitole budou popsány postupy a nástroje použité pro implementaci aplikace. Opět se zaměříme především na tvorbu prezentační vrstvy. Budou zmíněny některé techniky a komponenty, které byly pro implementaci klíčové.

V závěrečné kapitole budou shrnuty zkušenosti, které jsem během práce získal. Také bude zmíněno, co jsem se během práce naučil a jaký byl můj celkový přínos.

1.1 Aplikace Java EE Organizér

Java EE Organizér je aplikace, napsaná v jazyce *Java*, sloužící k organizaci úkolů, událostí a kontaktů. Úkolem je činnost, kterou má uživatel v plánu provést. Událostí je myšlena činnost, která je svázána s nějakým časovým údajem (např. naplánovaná schůze). Taková událost pak trvá určitou dobu a může se opakovat. Java EE Organizér je tedy určitou formou diáře. Ten uživateli pomáhá, aby si nemusel pamatovat vše, co má v plánu. Tato aplikace pomáhá vhodným způsobem organizovat a prezentovat naplánované činnosti. Tyto činnosti jsou zařazovány do uživatelem předem vytvořených kategorií. Obsahuje mechanismy pro připomínání blížících se termínů. S touto aplikací je spojeno i jednoduché uživatelské prostředí, které uživateli usnadňuje práci.

Java EE Organizér je především nástrojem pro efektivní organizaci času. Zajišťuje pouze nezbytné funkce pro organizaci času a kontaktů, jelikož příliš robustní (např. groupware) systémy jsou, v mnoha případech, pro běžné použití zbytečně složité (a díky množství různých funkcí také nepřehledné).

Kapitola 2

Analýza stávajících řešení

V této kapitole se okrajově zmíníme o historii a současnosti stávajících řešení organizačních nástrojů pro správu úkolů a kontaktů. Budeme se zabývat analýzou existujících řešení, která si rozdělíme do několika kategorií. Na závěr kapitoly bude zmíněna stručná charakteristika některých z nejvýznamnějších představitelů počítačových organizačních nástrojů dneška.

2.1 Úvod

Těžko by se datovalo období, kdy si lidé začali zaznamenávat své naplánované úkoly na nějaké médium, které jim pomáhalo si na úkol vzpomenout. V tomto ohledu nejsou organizéry žádná novinka. To stejné platí o zaznamenávání osobních informací o ostatních osobách. V dnešní uspěchané době je však pro uživatele stále důležitější mít tyto informace neustále k dispozici a mít možnost tyto informace pravidelně (a pokud možno nenáročně) zálohovat (jelikož žádné běžné médium nevydrží „všechno“).

Značným krokem vpřed byl rozvoj mobilních elektronických zařízení (např. kdysi populární databanky), které tuto funkci splňovali lépe než papírové diáře a jiné tradičnější prostředky. První mobilní zařízení pro organizaci času však byly uživatelsky velmi nepřívětivé. V dnešní době se již aplikace splňující funkci organizéru nahrávají do všech mobilních zařízení, jako například telefonů, smartphonů, PDA apod. Také se výrazně zlepšila uživatelská přívětivost a funkcionalita těchto aplikací. Mezi nejtypičtější pokroky patří strukturování úkolů do kategorií, upozorňování na termíny úkolů nebo možnost synchronizace s počítačem, případně přímo s internetovým serverem.

V současné době, kdy je velmi znatelný rozmach internetu a s ním spojených technologií, se však nabízí nové řešení. Toto řešení je, z pohledu konzistence dat a přenositelnosti mezi zařízeními, lepší. V době, kdy má člověk možnost se téměř odkudkoli připojit k síti internet, se otvírá možnost využívání síťových aplikací, které neběží na klientském stroji, nýbrž na vzdáleném serveru. K tomuto serveru je poté přes síť připojena klientská aplikace, která běží na libovolném klientském stroji, který to umožňuje. Tato možnost má pro uživatele hned několik velkých výhod.

První velkou výhodou takto řešených síťových zařízení je zajištění konzistence ukládaných dat. Jelikož se všechna data odesílají přímo na server, který je ukládá do databáze, prakticky nemůže nastat situace, že by některé připojené zařízení nezobrazovalo aktuální data. Jediná výjimka může nastat v situaci, že se data ze serveru ještě neaktualizovala po změně dat z jiného zařízení. Tato výjimka může být znatelná především u webových rozhraní, kdy je potřeba obnovit webovou stránku, aby se změna na serveru projevila.

Další výhodou aplikací tohoto druhu je přenositelnost. Díky tomu, že hlavní aplikace běží na serveru a na klientském stroji běží pouze prezentace, může být uživatel k aplikaci, vzhledem k nevelkým HW nárokům, připojen z prakticky jakéhokoli mobilního zařízení (mobilní telefon, PDA, notebook, netbook, ...) stejně dobře, jako ze svého stolního počítače. Této výhody lze dosáhnout dvěma způsoby. První možností je zavedení komunikace prezentační aplikace se serverem přes některý standardizovaný protokol (nejčastěji *HTTP*). Poté jsou data posílána ve standardizovaných formátech (např. *HTML*), které je možné otevírat a prohlížet pomocí klientských aplikací, které jsou obvykle distribuovány společně se všemi očekávanými koncovými zařízeními (webový prohlížeč). Další možností je vytvořit specializovanou klientskou aplikaci, která bude fungovat nezávisle na cílové platformě[1] (např. aplikace napsané v jazyce Java). První řešení je tedy přenositelné již na úrovni komunikace, zatímco druhé řešení je přenositelné až na úrovni klientského programu. Vzhledem k tomu, jak bývá architektura takovýchto aplikací navržena, se ovšem nemusíme spokojit pouze s jednou z těchto variant, ale můžeme přistoupit i na kombinované řešení. Můžeme tedy přistupovat k serveru jak z aplikačního klienta, tak z webového rozhraní. Aplikační klient může poskytovat vyšší komfort při použití na běžných počítačích (má vzhled podobný aplikacím, na které je uživatel zvyklý). Naproti tomu k webovému rozhraní můžeme přistupovat z prakticky libovolného přístroje, který obsahuje webový prohlížeč (na použití aplikačního klienta nemusí mít zařízení dostatečný výkon).

2.2 Lokální aplikace bez možnosti synchronizace

Jedná se o aplikace spojené s konkrétním médiem, na kterém si uchovávají všechna svoje data. Aplikace tohoto druhu neumožňují synchronizaci a přenos dat s jinými zařízeními. Často také nepodporují zálohování dat. Vznikne-li potřeba data zálohovat, je nutné záznamy manuálně opisovat. V případě elektronických zařízení je většinou přítomna možnost zálohovat obsah na úrovni souborů. Vývoj softwarových aplikací tohoto druhu je velmi jednoduchý a rychlý, přičemž je výsledek, pro mnohá využití, postačující. V dnešní době se softwarové aplikace tohoto druhu příliš nevyužívají.

Příklad zástupců této kategorie:

- Tužka a papír, úkolníčky, tel. seznam, ...
- Starší elektronické databanky
- Aplikace vytvořené k osobnímu použití

2.3 Lokální aplikace s možností synchronizace

Jedná se o, v současné době, nejrozšířenější formu organizačních nástrojů. Tyto aplikace, oproti předchozí kategorii, umožňují synchronizaci dat mezi dvěma aplikacemi. Typicky se synchronizují dvě zařízení, která jsou propojená přes USB kabel nebo přes síť. Samotná synchronizace pak většinou probíhá pomocí oddělené aplikace, která synchronizaci zajišťuje. Problémem této metody je obtížnost určení, které zařízení má aktuální data. Další problém nastane v případech, kdy uživatel využívá více než dvě organizační aplikace. V takovém případě musí postupně synchronizovat všechny aplikace. Tyto problémy obstojně řeší varianta se synchronizačním serverem.

[1] To znamená, že poběží stejně například pod MS Windows, jako pod Linuxem

Aplikace této kategorie mají několik charakteristických vlastností, které vyplývají z jejich podstaty. Nejdůležitější z nich je jejich autonomie. Aplikace pro svůj běh nevyžaduje spojení z žádným zařízením ani jinou aplikací (aplikaci nemusíte nikdy synchronizovat, a přesto bude fungovat správně). V případě, že si data chcete zálohovat, můžete tak učinit právě pomocí synchronizace. Nevýhodou je však nekonzistence záznamů mezi aplikacemi. Této nekonzistence se uživatel může zbavit pouze dočasně (ve chvíli, kdy všechny aplikace sesynchronizuje). Další nevýhodou těchto aplikací je jejich častá nepřenositelnost. Aplikace jednotlivých zařízení jsou většinou přenositelné pouze na úrovni možnosti exportu dat do společného formátu. To znamená, že konkrétní aplikace může běžet pouze na konkrétní platformě. Na ostatních platformách se musí použít aplikace jiné. Díky tomu si uživatel musí zvyknout na několik různých aplikací.

Mezi typické představitele této kategorie patří především aplikace pro správu úkolů a kontaktů, předinstalované v mobilních telefonech, PDA, smartphonech apod. Dalším zástupcem může být například Rainlendar Lite.

2.4 Aplikace typu klient-server

Systémy, založené na architektuře klient-server, jsou aplikace, jejichž správný chod je podmíněn existencí síťového spojení klientské aplikace a serveru. Data jsou vždy ukládána na serveru a klientská aplikace pouze poskytuje uživatelské rozhraní, které zajišťuje interakci s uživatelem. Díky této architektuře není třeba určovat, která aplikace má aktuální data, jelikož jsou všechna uložená na serveru. Takovéto aplikace navíc mohou umožňovat plnou přenositelnost mezi cílovými platformami. Aby se zmírnila nevýhoda nutnosti připojení na server, mohou tyto aplikace podporovat export dat do datových formátů, podporovaných offline aplikacemi (např. formát iCalendar). Poté je možné data číst i z mobilních offline zařízení a mít potřebné údaje neustále u sebe. Tato možnost může být mnohdy výhodná, přestože vyexportovaná data nejsou plně konzistentní s aktuálním stavem databáze. Tuto nevýhodu lze však často zanedbat, obzvláště v případech, kdy neočekáváme v nejbližší době žádnou změnu v databázi záznamů.

Současné aplikace tohoto druhu jsou většinou implementovány s webovým rozhraním, kvůli jeho pružnosti a plné přenositelnosti z pohledu koncového uživatele.

Pro příklad můžeme uvést tyto webové aplikace:

- Google Calendar
- Yahoo! Calendar
- MS Live Calendar
- eGroupWare
- MS Exchange (webaccess)

Jsou však dostupné i aplikace podporující přístup k serveru pomocí aplikačního klienta. Za tuto kategorii můžeme zmínit například kombinaci *MS Exchange server* a *MS Outlook* (v pozici klientské aplikace).

2.5 Analýza často využívaných aplikací

V této sekci naleznete stručnou charakteristiku vybraných často používaných aplikací. Budou zmíněné jak freeware aplikace, tak komerční řešení. Často se jedná o groupware systémy, což jsou sofistikovanější aplikace sloužící k usnadnění managementu vícečlenného týmu. V takovýchto aplikacích se pak nejedná o organizaci času jedince, ale i o organizaci času ostatních členů týmu (události a úkoly může například zadávat vedoucí projektu).

MS Exchange

Jedná se o komerční produkt firmy Microsoft, který je typickým příkladem propracovaného groupware. Tato aplikace obsahuje velmi širokou škálu funkcí, které jsou vhodné pro organizaci činností velkých firem. Obsahuje nástroje pro správu projektu, organizaci kontaktů, úkolů, emailů, hlasových zpráv apod. MS Exchange očekává MS Outlook jako klientskou aplikaci. Tento software je tedy určený především pro uživatele a firmy, které využívají operační systém Windows. Existují však i cesty, jak pracovat s MS Exchange v klientských aplikacích na jiných systémech (např. Evolution v Linuxu nebo přes webové rozhraní). Pro MS Exchange existuje velké množství rozšíření, které slouží ke zjednodušení práce v týmu. Na organizaci malých firem, případně pro osobní užití jediného uživatele, je však často tento systém příliš robustní a z ekonomického hlediska nevýhodný. Nicméně se jedná o velmi mocný nástroj.

Evolution

Evolution je jednoduchý, ale poměrně komplexní nástroj k organizaci času, poznámek a kontaktů, který je zdarma distribuován pod platformu GNOME. Podporuje standardní formát iCalendar. Podporuje synchronizaci s přenositelnými zařízeními a s Google Calendar. Z grafického hlediska mě příliš nezaujal, ale jinak se jedná o šikovný nástroj pro jednoduchou správu činností. Je schopný pracovat autonomně, ale je také využitelný jako groupware klient (např. je schopný spolupráce s MS Exchange). Navíc se dá aplikace používat jako emailový klient.

eGroupWare

Jak název napovídá, jde o groupware systém. Systém eGroupWare je serverová aplikace, která je distribuována zdarma pod GNU GPL licencí. Poskytuje velmi pěkné webové prostředí s intuitivním ovládáním. Podporuje také připojení groupware klientů, jako MS Outlook nebo Evolution. Je to robustní systém, který poskytuje správu projektů, zdrojů, kaledářních funkcí, emailů a spoustou dalších funkcí. Nevýhodou je poměrně složitá instalace.

Rainlendar Lite

Rainlendar Lite je jednoduchá lokální aplikace s pěkným a intuitivním uživatelským prostředím. Je distribuována zdarma, ale má i komerční verzi – Rainlendar Pro. Rainlendar má vzhled malého kalendáře umístěného na ploše. Podporuje barevná schémata, správu úkolů a událostí, připomínání a export do formátu iCalendar. Ve verzi pro navíc podporuje synchronizaci s aplikací Google Calendar a poskytuje podporu MS Outlook. Aplikace existuje pro operační systémy Linux, MS Windows a Mac OS. Na lokální úrovni se jedná o velmi

příjemný nástroj. Hlavním důvodem je pravděpodobně jednoduchost a elegantní umístění aplikace na ploše, což je velmi praktické. Bohužel v aplikaci zcela chybí správa kontaktů.

2.6 Závěr

V této kapitole jsme se seznámili s architekturami a přístupy různých existujících aplikací, které slouží k efektivní organizaci času. Zmínili jsme některé konkrétní aplikace, které jsou většinou poměrně rozsáhlé a slouží k velmi sofistikované organizaci. Často jsou až příliš složité pro typického uživatele, který by se raději ohlédl po jednodušším nástroji, který bude distribuován zdarma a bude podporovat funkce, které potřebuje nejčastěji.

Kapitola 3

Použité technologie

V této kapitole budou blíže popsány technologie, které byly použity pro vývoj a správu aplikace Java EE Organizér. Zvláštní důraz bude kladen na technologii Java EE[1] a s ní spojený aplikační server GlassFish[2] a databázový server Firebird[1]. Dále bude popsáno API pro tvorbu grafického uživatelského prostředí zvané *Swing*.

3.1 Model třívrstvé architektury

Tato sekce vysvětluje pojem „model třívrstvé architektury“, popisuje strukturu, výhody a jednotlivé vrstvy této architektury. Informace pro tuto sekce byly čerpány ze serveru `ITExpert.cz` [4].

Model třívrstvé architektury vznikl z původního, dnes již zastaralého, dvouvrstvého modelu (klient – server). Aplikace je rozdělena do tří vrstev, kdy každá vrstva poskytuje navenek určité rozhraní. Přes toto rozhraní pak může komunikovat se sousední vrstvou. Vrstvy jsou díky tomu na sobě nezávislé. Každá vrstva také může mít hned několik různých funkčních řešení, pouze stačí dodržovat komunikační rozhraní se sousedními vrstvami. Každá vrstva může běžet na zvláštním počítači, přestože to není vyžadováno. Model třívrstvé architektury je zobrazen na obrázku 3.1.

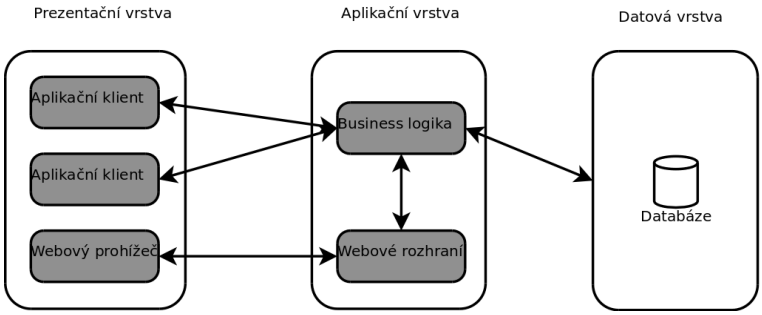
Aplikace je rozdělena do následujících tří vrstev:

- *Prezentační vrstva*
- *Aplikační vrstva*
- *Datová vrstva*

Datová vrstva

Datová vrstva slouží k ukládání perzistentních dat a k přístupu k nim. Obvykle běží na databázovém serveru (SŘBD[2]), ale obecně se může jednat i o jiný typ aplikace. Teoreticky je možné na SŘBD uložit neomezené množství databází. K těm je pak možné přistupovat z neomezeného množství různých aplikací (např. aplikačních serverů, databázových klientů

[1] Java EE – Java Platform, Enterprise Edition
[2] SŘBD – Systém řízení báze dat



Obrázek 3.1: Model třívrstvé architektury

apod.) současně. Naše aplikace využívá SŘBD *Firebird*, který je ovládán dotazovacím jazykem SQL[3].

Aplikační vrstva

Aplikační vrstva zajišťuje spojení aplikace s databází. Obsahuje také aplikační (business) logiku systému. Zajišťuje transformace mezi klientskými požadavky a datovými operacemi. Činnost aplikační vrstvy zpravidla probíhá na žádost vyvolanou klientským programem. Tato vrstva je v podstatě samostatným programem, běžícím typicky na síťovém serveru, ke kterému se klientské aplikace připojují vzdáleně. V případě demonstrační aplikace Java EE Organizér, běží aplikační vrstva na aplikačním serveru *GlassFish V2*.

Prezentační vrstva

Prezentační vrstva slouží k vzájemné interakci aplikace s uživatelem. Zajišťuje načítání uživatelských dat i zobrazování informací. Typickou prezentační vrstvou je například webové rozhraní internetové aplikace (např. internetového obchodu). Často se také využívá tzv. *klientské aplikace*, což je specializovaný program zajišťující vlastnosti prezentační vrstvy, který vzdáleně komunikuje se serverem přes síť. Prezentační vrstva zpravidla reaguje na události od uživatele (stisk tlačítka, pohyb myši, stisk klávesy, ...), systému nebo aplikace. V případě aplikace Java EE Organizér, se jedná o grafickou aplikaci napsanou v jazyce Java pomocí Swing API (viz 3.5).

V Javě EE je velmi silná podpora pro třívrstvou architekturu, pro kterou byla vytvořena. Má silné nástroje pro tvorbu jak prezentační vrstvy (např. webových stránek), tak pro podporu komunikace s databázovými servery.

3.2 Java EE

Pro tuto bakalářskou práci byla stěžejní platforma Java EE 5, která bude v této sekci blíže popsána. Informace pro tuto sekci byly čerpány převážně z tutoriálu o Javě EE [7] a z oficiálních stránek Javy EE [5]. Okrajově jsem čerpal i ze stránek o Javě EE na Wikipedii [12].

[3]Na úrovni aplikační vrstvy se bude jednat o jazyk JPQL, který je pouze dialektem SQL a který je na jazyk SQL převáděn pomocí ovladače.

Oficiální definice Javy EE volně přeložená Stanislavem Hybáškem v jeho diplomové práci[3]:

> *Java EE je standardizovaná platforma určená pro vývoj přenosných, robustních, škálovatelných a dobře zabezpečených serverových aplikací v jazyce Java.*

Jak již tato definice napovídá, je to soubor nástrojů, které usnadňují implementaci a běh robustních informačních systémů. Obzvláště výhodné je použití pro implementaci síťových aplikací založených na vícevrstvých architekturách. Technologie Java EE tedy není jen API rozšířené o další knihovní funkce, ale je to celá platforma, které se musí přizpůsobit celé běhové prostředí (aplikační server). Důvodem, proč byla platforma Java EE vytvořena, byla snaha pomoci vývojářům, aby se mohli soustředit čistě na implementaci aplikační funkcionality (a nemuseli se přitom zabývat zdlouhavým řešením technologií síťových přenosů, komunikací s databází apod.).

3.2.1 Technologie Java EE 5

Výčet technologií Javy EE, v této sekci, je přejat z oficiálních stránek Javy EE[6]. Některé informace byly čerpány z doplňkových dokumentů firmy Sun Microsystems, Inc.[11].

Java EE je koordinovaná sada technologií, které významně usnadňují proces vývoje, nasazení a správy vícevrstvých serverových aplikací. Java EE je postavená na základě platformy Java SE[4], kterou doplňuje tak, aby poskytovala kompletní, stabilní, bezpečnou a rychlou platformu pro podnikové (enterprise) systémy. Tyto technologie jsou řazeny do skupin.

Skupiny technologií Javy EE:

- Java EE 5 SDK
- Technologie webových služeb
- Technologie webových aplikací
- Technologie enterprise aplikací
- Technologie správy a zabezpečení

Java EE 5 SDK

Java EE 5 SDK [5] je souhrn vývojových nástrojů pro platformu Java EE. Obsahuje kompletní API se všemi rozšířeními Javy EE. Navíc zahrnuje aplikační server GlassFish. Podle typu distribučního balíku pak může obsahovat řadu dalších doplňků.

Technologie webových služeb

Mezi základní technologie webových služeb platformy Java EE patří:

- Java API for XML-Based Web Services (JAX-WS)
- Java API for XML-Based RPC (JAX-RPC)

[4] Java SE – Java Platform, Standard Edition
[5] SDK – Software Development Kit

- Java Architecture for XML Binding (JAXB)
- SOAP with Attachments API for Java (SAAJ)
- Streaming API for XML
- Web Services Metadata for the Java Platform

Technologie webových aplikací

Silnou stránkou technologie Java EE je silná podpora pro tvorbu webového prezentačního prostředí. K tomuto účelu má tato platforma k dispozici speciální druh dynamických webových stránek – tzv. JavaServer Pages (JSP). Tyto stránky mohou obsahovat speciální *tagy*[6], založené na bázi Javy, které umožní snadný přístup ke komponentám aplikační vrstvy. Díky těmto komponentám pak můžeme například načítat data z databáze a poskytovat je zpět webovému rozhraní. Pomocí technologií webových aplikací jsme schopni naimplementovat kompletní prezentační vrstvu, zobrazující se koncovému uživateli ve webovém prohlížeči.

Rozdělení technologií webových aplikací

- JavaServer Faces (JSF) – technologie, která umožňuje provázání JSP stránek a tzv. *backing bean* komponent, což jsou Java beany, které obsahují vlastnosti (tzv. *properties*). Tyto backing beany pak, pomocí getterů[7] a setterů[8] těchto vlastností, definují funkcionalitu uživatelských komponent na JSP stránkách.
- JavaServer Pages (JSP) – technologie pro psaní speciálních webových stránek pomocí jazyka podobnému xhtml.
- JavaServer Pages Standard Tag Library – knihovna značek použitelných v JSP stránkách
- Java Servlet – třídy jazyka Java. Používají se k rozšíření možností serverů, které „hostí“ aplikace přistupující skrze programový model dotaz–odpověď. Ačkoli servlety mohou odpovídat na jakoukoli žádost, většinou jsou použity k rozšíření aplikací „hostěných“ webovými servery.

Technologie enterprise aplikací

Technologiemi enterprise aplikací se rozumí technologie a frameworky použité pro implementaci a běh aplikační logiky. Jedná se jak o nástroje pro implementaci serverové aplikace, tak pro implementaci přístupu k ní z klientských aplikací.

Rozdělení technologií enterprise aplikací:

- Common Annotations for the Java Platform
- Enterprise JavaBeans (EJB)
- J2EE Connector Architecture

[6] Tag – Značka (pojem vztahující se ke značkovacím jazykům, vycházejících z SGML)
[7] Getter – Spec. metoda pro získání hodnoty atributu
[8] Setter – Speciální metoda pro nastavení hodnoty atributu

- JavaBeans Activation Framework (JAF)
- JavaMail
- Java Message Service API
- Java Persistence API (JPA)
- Java Transaction API (JTA)

Pro aplikaci Java EE Organizér byly nejpodstatnější technologie EJB (verze 3.0) a JPA, které klientské aplikaci umožňují načítat, skrze aplikační vrstvu, data z databáze.

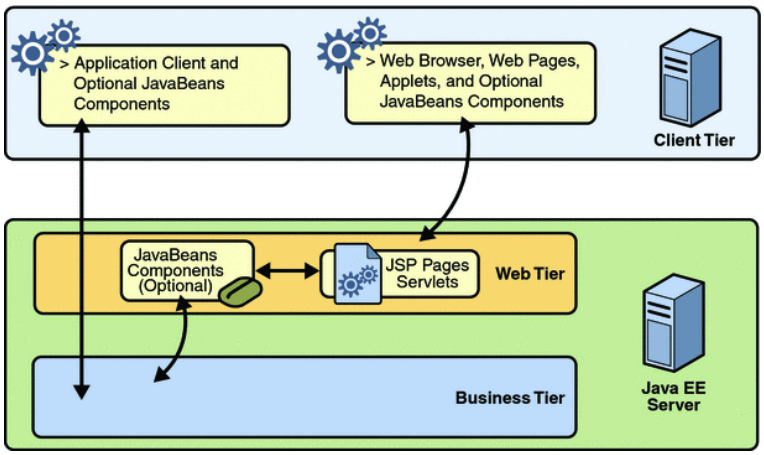
Technologie správy a zabezpečení

Tyto technologie umožňují nasazení serverové aplikace na aplikační server, její sledování a správu za běhu, specifikaci autorizačních modelů a bezpečnostní politiky, které budou aplikační kontejnery využívat.

Rozdělení technologií správy a zabezpečení:

- J2EE Application Deployment
- J2EE Management
- Java Authorization Contract for Containers

3.2.2 Základní architektura aplikací platformy Java EE



Obrázek 3.2: Architektura aplikací v Javě EE

Jak vidíme na obrázku 3.2, aplikace obsahuje obecně několik typů kontejnerů. Na aplikačním serveru jsou webové a EJB kontejnery, zatímco na klientském počítači je to kontejner klientské aplikace. Každý tento kontejner má svou funkci související s vrstvou, na které je umístěn.

Enterprise JavaBean kontejnery

EJB jsou kontejnery aplikační vrstvy, které obsahují a spravují enterprise beany[9].

EJB kontejnery musí zajišťovat následující operace:

- registrovat objekt pro každý enterprise bean
- poskytovat vzdálené rozhraní objektů
- spravovat vytváření a likvidaci instancí objektů
- kontrolovat bezpečnostní omezení objektů
- spravovat aktivní stav objektů
- koordinovat distribuované transakce
- může spravovat všechny perzistentní data objektů

Webové kontejnery

Webové kontejnery implementují běhové prostředí pro webové komponenty, které zahrnují bezpečnost, souběžnost, řízení životního cyklu, transakce, nasazení a další služby. Starají se o spouštění JSP stránek a servletových komponent Java EE aplikací.

Kontejnery klientské aplikace

Kontejnery klientské aplikace (Application client containers) jsou třídy, knihovny a ostatní soubory, distribuované s klientskou aplikací, spouštějící se na stejném virtuálním stroji. Zajišťují také spouštění klientských komponent. Kontejnery poskytují systémové služby povolující spuštění klientského programu. Komunikují s jedním aplikačním serverem pomocí RMI/IIOP a spravují detaily RMI/IIOP komunikace pomocí client ORB, které je s nimi v balíku. Kontejner klientské aplikace je speciální typ EJB kontejneru a je obvykle poskytován stejným prodejcem. Kontejner klientské aplikace není tolik robustní, jako Java EE kontejnery.

3.2.3 Anotace

Java EE 5 využívá zápis metadat pomocí *anotací* (Annotations). Anotace mohou být aplikovány na deklarace tříd, atributů, metod a ostatních prvků programu. Dříve se metadata ukládala do speciálních xml souborů. Tyto xml soubory nebyly, z pohledu vývojáře, nijak příjemné (Obecně je ruční tvorba xml náročná, díky „upovídanosti“ jazyka.). Tyto anotace umožňují specifikovat informace o datech, operacích apod., které neovlivňují přímo kód, ale jsou důležité např. pro běhové prostředí, překladač či vývojová prostředí. Anotacemi můžeme například označit metodu za přetěžující, třídu za entitní apod. Anotace pomáhají prostředí k lepšímu pochopení sémantiky prvků programu. Tento přístup značně usnadňuje práci vývojářům, kteří v takové míře nemusí psát kódy konfiguračních souborů.

[9] Enterprise bean – Komponenta na straně serveru uložená v EJB kontejneru, která implementuje business logiku aplikace.

3.2.4 Java Persistence

Java Persistence je nástroj ke správě relačních dat v Java aplikacích, poskytující objektově-relační mapování. Tato technologie je typicky využívána na straně aplikačního serveru v EJB komponentách, ale je možné ji využít i v klientské aplikaci.

Java Persistence, spojená se specifikací Java EE 5, se skládá ze tří částí:

- JPA[10] – soubor knihoven, které má vývojář k dispozici ke správě perzistentních dat.
- JPQL[11] – speciální jazyk pro tvorbu dotazů na perzistentní data.
- Metadata pro objektově-relační mapování (pomocí anotací).

Jazyk JPQL je speciálním dialektem dotazovacího jazyka SQL. Od SQL se liší v tom, že nepoužívá názvy tabulek a jejích atributů v databázi, ale názvy už přímo namapovaných tříd a jejich atributů. V Javě je možné využít i dotazy v jazyce SQL, pomocí kterých je možné dotazovat se do databáze přímo. Tento způsob se typicky využívá v případech, kdy není použito mapování entitních tříd.

Java Persistence nám umožňuje vytvářet *entitní třídy*. Jsou to třídy, které můžeme pomocí anotací namapovat na objekty objektově-relační databáze. To provedeme tak, že pomocí anotací specifikujeme mapování jednotlivých atributů databázových objektů na atributy těchto tříd. Takovéto entitní třídy musí být serializovatelné, musí obsahovat implicitní konstruktor a musí obsahovat gettery a settery pro atributy mapované na databázový objekt. Entitní třídy, tabulky na které se mapují a mapované atributy se specifikují pomocí anotací. Java Persistence nám také umožňuje definovat, pro jednotlivé entitní třídy, tzv. *pojmenované dotazy* (Named Queries) v jazyce JPQL. Tyto dotazy se typicky využívají pro získávání jednotlivých entit nebo celých kolekcí na základě vztahů některých atributů k definovaným parametrům (např. shoda cizího klíče). Pojmenované dotazy se zapisují pomocí anotací.

Dále nám JPA poskytuje nástroje pro správu entitních tříd, pomocí tzv. *Entity Manageru*. Jedná se o třídu, která umožňuje přidávání, odebírání a změnu jednotlivých entit. Také umožňuje získávání odkazů, na jednotlivé instance entitních tříd i celé kolekce, na základě už zmíněných pojmenovaných dotazů.

3.3 Aplikační server GlassFish

GlassFish je open source software sloužící jako AS[12] pro platformu Java EE. V současné době se nejčastěji používá verze GlassFish V2. GlassFish V3 je dostupný pouze jako prelude a je primárně vytvořena pro Javu EE 6, která v současné době ještě nevyšla.

Server GlassFish byl použit, jelikož obsahuje podporu platformy Java EE na vysoké úrovni. Hlavním důvodem je pochopitelně fakt, že jde o software vyvíjený přímo firmou Sun Microsystems, Inc.

Sun Microsystems, Inc. nabízí také komerční aplikační server (Sun Application Server), který ovšem není pro potřeby naší aplikace vhodný, díky svým licenčním podmínkám. Velkou výhodou serveru GlassFish je tedy možnost zcela bezplatného využití jak k osobním, tak komerčním účelům.

[10] JPA – Java Persistence API
[11] JPQL – Java Persistence Query Language
[12] AS – Aplikační server

Na aplikačním serveru poběží business logika aplikace, která spočívá v obsluhování žádostí klientských programů a přístupům do databáze, ze které vybírá data (popřípadě je edituje, či ukládá). Aplikační server také řeší otázky připojování více uživatelů skrze síť a obsluhování jejich požadavků.

Bližší informace o AS GlassFish naleznete na oficiální webové stránce (viz [2]).

3.4 Firebird

Firebird je databázový server založený na relačním databázovém modelu. Jedná se o dostatečně robustní multiplatformní databázi, která je vhodná pro malé i středně velké firmy. Navzdory tomu obvykle instalátor Firebirdu zabírá pouze okolo 5 MB. Mezi nejdůležitější faktory, které ovlivnili rozhodnutí použít tento databázový server, patřily zejména:

- Plná podpora ACID transakcí
- Licence, která umožňuje bezplatně využívat Firebird i pro komerční využití (Initial Developer's Public License)
- Jednoduchá správa (minimální požadavky na údržbu)
- Existující ovladač pro Javu EE

Firebird je dostupný ve třech základních formách:

- SuperServer
- Classic
- Embended

V přípravě je ještě čtvrtá forma „SuperClassic“, která je plánovaná pro verzi 2.5

Firebird Embended není vhodný pro aplikaci našeho typu, kvůli některým jeho vlastnostem. Například postrádá systém uživatelských práv. Navíc je nezbytné, aby běžel na stejném stroji jako AS.

Pro naší aplikaci jsou tedy vhodné především varianty SuperServer a Classic. Varianta SuperServer vytváří pro každý požadavek nové vlákno a sdílí pro všechny klienty stejnou cache. Naproti tomu varianta Classic vytváří pro jednotlivé uživatele nové procesy a má zvláštní cache pro každého klienta.

Aplikace Java EE Organizér nevyžaduje použití konkrétně Firebird SuperServer nebo Firebird Classic. Rozhodnutí o použité variantě provede správce serveru. Ten se rozhodne na základě provozních podmínek, do kterých bude server nasadazen (množství připojených klientů atd.).

3.5 Swing API

Tato sekce se zabývá vysvětlením a popisem technologie Swing. Informace pro tuto sekci jsem čerpal především z tutoriálu[10] umístěném na oficiálních stránkách Javy, případně ze seriálu (viz [13]) o Swing API na serveru `zaachi.com`.

Swing API (dále jen Swing) je soubor nástrojů jazyka Java sloužících k programování grafického uživatelského rozhraní (GUI). Slouží ke snadné a sofistikované implementaci

GUI a snaží se poskytovat nástroje pro příjemné a efektivní ovládání. GUI je oboustranné rozhraní mezi uživatelem a aplikací, které zajišťuje interakci uživatele s aplikací.

Swing je postaven na starším grafickém rozhraní AWT[13]. Swing, oproti AWT, poskytuje nastavitelnost vzhledu (tzv. *Look And Feel*), možnost využívat ikony, tooltip texty apod. Na druhou stranu nemá tak silnou podporu ve webových prohlížečích (důležité pro tvorbu appletů) a využívá mnohé nástroje přímo z AWT. Swing je součástí JFC[14] a jako takový je obsažen v základním sortimentu tříd Javy SE[15].

Grafické aplikace napsané ve Swingu jsou zpravidla ovládány událostmi. Jakákoli akce, kterou aplikace provádí, je pak podmíněna nějakou žádostí od uživatele, operačního systému nebo aplikace.

Jak již bylo zmíněno, Swing poskytuje nástroje pro přenastavování celkového vzhledu (skinu) aplikace. Uživatelské rozhraní napsané ve Swingu může být zobrazováno různými způsoby. O daném vzhledu může rozhodnout vývojář aplikace. V případě, že tak neučiní, je nastavení na uživateli. V tomto ohledu je Swing značně pokročilejší než AWT, jehož vzhled byl silně závislý na platformě, na které zrovna běžel(z čehož vyplývala určitá omezení). Grafické rozhraní ve Swingu může také přebírat vzhled z konkrétního operačního systému, na kterém je aplikace spuštěna. Tím může uživateli velmi usnadnit práci, jelikož si nemusí zvykat na nové prostředí. Rozhraní může také naopak použít základní vzhled Swingu (tzv. *Metal Look And Feel*), který vypadá stejně pod všemi platformami, na které existuje virtuální stroj. Touto možností se dá dosáhnout vysoké míry přenositelnosti a nezávislosti na možnostech uživatelských prostředí daného operačního systému.

3.5.1 Komponenty Swing API

Swingové aplikace se skládají z grafických komponent, obsažených v balíku `javax.swing`. Těchto komponent existuje celá řada, ale my si zmíníme alespoň jejich základní rozdělení, funkce a nejpoužívanější zástupce.

Komponenty Swingu se hierarchicky dělí do následujících skupin:

- Top-level kontejnery (Top-Level Containers)
- Obecně využívané kontejnery (General-Purpose Containers)
- Kontejnery pro zvláštním využití (Special-Purpose Containers)
- Základní ovládací prvky (Basic Controls)
- Needitovatelné informační prvky (Uneditable Information Displays)
- Často využívané interaktivní prvky (Interactive Displays of Highly Formatted Information)

Top-level kontejnery

Jedná se o komponenty, které se nacházejí na vrcholu hierarchie. Tyto komponenty utvářejí okna aplikace. Podmínkou pro běh grafické aplikace je existence alespoň jednoho top-level

[13] AWT – Abstract Window Toolkit
[14] JFC – Java Foundation Classes
[15] Java SE – Java Platform Standard Edition

kontejneru, ve kterém je implementováno GUI aplikace. Top-level kontejnery obsahují vlastnosti, jako například umístění okna na obrazovce, velikost okna, posouvatelnost, minimalizovatelnost, maximalizovatelnost, titulek, průhlednost a v neposlední řadě speciální grafický kontejner, který se nazývá *content pane*. Narozdíl od zbytku komponent Swingu, top-level kontejnery není možné vkládat do grafických kontejnerů. Ke vkládání dalších grafických komponent do top-level kontejnerů slouží výše zmíněný content pane, což je v praxi některý grafický kontejner, z nižší úrovně hierarchie, do kterého je možné vkládat dílčí komponenty či kontejnery. Top-leve kontejnery se delí na *formuláře* (Frames) a *dialogy* (Dialogs). Ve Swingu jsou implementovány pomocí tříd `JFrame` a `JDialog`. Zpravidla má aplikace jeden formulář, do kterého je vložena podstatná část GUI. Zbylá okna, která aplikace zobrazuje, jsou implementována pomocí dialogů. Dialogy mají dva základní typy. Jedná se o dialogy modální a nemodální. Modální dialog je dialogové okno, které je zobrazeno na popředí a neumožňuje uživateli využívat ostatních oken aplikace, dokud se dialog neuzavře. Takto se dá řešit například zobrazování chyb, důležitých upozornění, potvrzovacích dialogů apod. Nemodální dialogy jsou dialogová okna, která tuto vlastnost nemají (Můžeme je využívat současně s ostatními okny aplikace.). Potvrzovací dialogy, dialogy pro zobrazování upozornění aj. můžeme realizovat pomocí třídy `JOptionPane` a jejích statických metod, které zajišťují zobrazení dialogu s určitými vlastnostmi. To je výhodné především při tvorbě velmi jednoduchých dialogů, které například obsahují jen krátkou zprávu. Složitější dialogová okna se implementují pomocí zmíněné třídy `JDialog`.

Běžné kontejnerové komponenty

Jsou to komponenty, které do sebe umožňují vkládat další grafické komponenty jakéhokoli druhu, jiného než top-level kontejnery. Můžeme tedy vytvářet i stromovou hierarchii kontejnerových komponent. Kontejnerové komponenty se využívají i jako content pane top-level kontejnerů. Do kontejnerů se grafické komponenty vkládají pomocí kolekce. Základní komponentu této úrovně představuje třída `JPanel`. Tato třída reprezentuje pouze prázdný grafický kontejner. Trochu složitějšími komponentami jsou `JScrollPane` (kontejner, který umožňuje zobrazit plně funkční scrollovací posuvníky) a `JTabbedPane` (kontejner, který slouží k organizaci komponent do „karet“). Kromě těchto komponent, které se nazývají *obecně využívané kontejnery*, se můžeme setkat také se složitějšími *kontejnery pro zvláštní použití*. Zástupci těchto specializovanějších kontejnerů jsou `JLayeredPane` (kontejner, který umožňuje skládat komponenty do více vrstev a řeší jejich viditelnost) a `JInternalFrame` („okno v okně“ – vnitřní okno zobrazené uvnitř formuláře či dialogu).

Základní ovládací prvky

Jedná se o základní prvky uživatelského rozhraní, pomocí kterých uživatel ovládá aplikaci. Jsou to například tlačítka, textová pole, zaškrtávací políčka, výběrová tlačítka, seznamy (klasické i rozbalovací) atd. Z těchto prvků se vytvářejí uživatelské formuláře[16], pomocí kterých uživatel intuitivně zadává vstupy do aplikace. Pro příklad si uvedeme tyto třídy:

- `JTextField` (jednořádkové textové pole)
- `JPasswordField` (pole pro zadání hesla)
- `JButton` (tlačítko)

[16] Ve smyslu vyplňovacího formuláře, nikoli top-level kontejneru

- `JRadioButton` (výběrové tlačítko, typicky použité ve skupině
- `JTextArea` (víceřádkové textové pole)
- `JCheckbox` (zaškrtávací pole)
- ...

Needitovatelné informační prvky

Tyto prvky slouží k zobrazování uživatelských informací (nepředpokládá se jiné, než informativní využití). Neslouží tedy pro vstup uživatelských dat, ale pouze jako zdroj informací pro uživatele (například popisky). Typickými představiteli jsou třída `JLabel` (třída pro popisky) a tzv. tooltip text [17]. Také je možné do této kategorie zařadit komponenty třídy `JProgressBar` (komponenta pro zobrazení průběhu operace).

Často využívané interaktivní prvky

Jsou to složitější předvytvořené atomické komponenty implementující nástroje pro často využívané operace. Možnost použít tyto nástroje poskytuje programátorovi prostor pro implementaci důležitějších částí kódu. Tím se usnadní práce nejen programátorovi, ale také uživateli. Ten ,díki existenci těchto předvytvořených nástrojů, může v různých aplikacích provádět stejné operace v prostředí, které je mu známé. Tento přístup je umožněn díky používání obdobných nástrojů pod všemi platformami.

Zástupci této kategorie:

- `JTable` (komponenta vizualizující tabulku)
- `JFileChooser` (komponenta sloužící pro výběr souboru)
- `JTree` (zobrazení stromu)
- `JColorChooser` (komponenta pro výběr barvy)

3.5.2 Správci rozložení

Správce rozložení (Layout Manager) je speciální objekt, který se stará o rozmísťování komponent uvnitř grafického kontejneru. Jestliže chceme do kontejneru umístit grafické komponenty, vložíme je do kolekce a jejich správné zobrazení zajistíme nastavením určitého správce rozložení. Správce rozložení rozmisťuje komponenty na základě pořadí, ve kterém se nacházejí v kolekci, nebo podle údajů obsažených ve speciálních konstantách. Ty je možné přidávat do kolekcí společně s komponentami. Složitější design aplikace se často vytváří tak, že se vytvoří několik grafických kontejnerů, do kterých se rozmístí potřebné komponenty. Tyto kontejnery jsou pak rozmístěny do kořenového kontejneru (obvykle content pane). Jelikož je však tento přístup velmi nepřehledný, je vhodné zapouzdřovat dílčí kontejnery do zvláštních tříd.

[17] Informační poznámka v okénku, které se zobrazí po nastavení kurzoru myši nad grafickou komponentu

3.5.3 Řízení událostmi

Jak již bylo na začátku kapitoly zmíněno, grafické aplikace se ovládají pomocí různých událostí, ať už vyvolaných uživatelem, operačním systémem nebo programem. Každá komponenta může vyvolat řadu událostí, v závislosti na jejím druhu. Zdroje těchto událostí jsou potomky třídy `Event` z balíku `java.awt`. Podle zdroje událostí rozeznáváme několik kategorií událostí (například události vyvolané myší a podobně).

K odchytávání událostí slouží tzv. *listenery* (posluchači). Listenery jsou rozhraní, která slouží pro odchytávání vždy konkrétní kategorie událostí. Abychom mohli listenerem odchytávat danou kategorii událostí, musíme listener zaregistrovat dané komponentě, u které chceme naslouchat událostem. Před touto registrací však musíme implementovat všechny metody daného rozhraní, které se budou volat jako reakce na konkrétní událost dané kategorie. Těmto metodám se říká *handlery* (obslužná rutina) a slouží k obsluhování daných událostí. Pro každou standardní kategorii událostí tedy existuje rozhraní, jehož implementace zajišťuje odchytávání daných událostí a pro každou konkrétní událost existuje metoda v tomto rozhraní, jejíž implementace zajišťuje obsluhu dané události.

Tímto způsobem je možné implementovat chování celé aplikace. Implementace chování aplikace tedy spočívá v implementaci daných listenerů, implementaci jejich metod (handlerů) a následném zaregistrování listenerů daným komponentám.

Tvorba grafické aplikace ve Swingu je tedy složena z vytváření a propojování grafických komponent, kterým následně přidáme jejich funkcionalitu pomocí definic obsluhy událostí těchto komponent.

Kapitola 4

Návrh

V této kapitole se nachází popis základního návrhu aplikace Java EE Organizér. Rozsáhleji bude zmíněn především návrh uživatelského rozhraní aplikace.

4.1 Návrh Aplikační vrstvy

Většina návrhu aplikační vrstvy byla provedena v rámci základního návrhu aplikace v práci Pavla Paláta. Ten vytvořil i patřičná rozhraní, která nastiňovala implementaci jednotlivých EJB komponent.

Aplikační vrstva (aplikace Java EE Organizér), v rámci business logiky, zajišťuje především služby pro správu perzistentních dat. Jako taková musí poskytovat nástroje pro načítání, přidávání, odebírání a úpravu daných entit. Jelikož jsou v databázi dané entitní množiny stejné pro všechny uživatele, je třeba na úrovni aplikační logiky zajistit, aby každá uživatelsky definovaná operace mohla být provedena pouze nad jeho vlastními daty. Tato vlastnost je z pohledu bezpečnosti klíčová. Z tohoto důvodu musí uživatel k žádosti o provedení dané operace přiložit i údaj o své identifikaci, který dostane při přihlášení do systému. Business logika pak musí ověřit, jestli má uživatel právo operaci provést.

V rámci aplikační vrstvy se také vyskytl požadavek na možnost filtrování úkolů, které chce uživatel načíst. Jedná se o filtraci na základě současného stavu úkolu a kategorie, do které spadá. S žádostí o zobrazení úkolů se tedy, kromě identifikace koncového uživatele, předávají i informace, jestli chce uživatel načíst jen konkrétní kategorii (a kterou), případně jestli chce načíst pouze nedokončené úkoly.

4.2 Návrh Prezentační vrstvy

Tato sekce pojednává o návrhu uživatelského rozhraní aplikace.

V rámci prezentační vrstvy jsem se podílel především na návrhu a implementaci:

- základního rozložení grafických komponent
- rozhraní pro správu kontaktů
- rozhraní pro správu úkolů
- hlavní třídy aplikace

4.2.1 Návrh základního rozložení grafických komponent

Základní myšlenka návrhu rozložení byla, aby měl uživatel vždy po ruce nástroje, které bude často využívat. Naproti tomu nástroje, které se využívají méně často, je možné zpřístupnit např. přes menu, jelikož je uživatel nepotřebuje příliš často a při běžné práci by jej spíše obtěžovaly.

Podle těchto kritérií se nástroje aplikace rozdělily následovně:

1. Často využívané nástroje
    - Zobrazení událostí dne
    - Výběr dne z měsíce
    - Zobrazení úkolů
2. Zřídka využívané nástroje
    - Zobrazení kategorií
    - Zobrazení kontaktů
    - Zobrazení nastavení
3. Formuláře pro editaci a přidávání
    - Formulář úkolu
    - Formulář kategorie
    - Formulář kontaktu
    - Formulář události

První kategorie nástrojů byla navržena jako kombinace tří vnitřních oken. Tyto vnitřní okna je možné přesouvat, zvětšovat, zmenšovat, minimalizovat, maximalizovat apod. Vnitřní okna je také možné zavřít (křížkem) a později znovu zobrazit pomocí položky v menu. Je také možné nastavit volbu, kdy budou udržovat svoji pozici a velikost v závislosti na poměru stran okna aplikace. Uživatelské akce těchto nástrojů se provádějí na základě výběru myší, dvojkliku na konkrétní komponentu nebo pomocí popup menu, které se zobrazí po pravém kliknutí myší.

Druhá kategorie nástrojů uvedená v seznamu je navržena jako dialogová okna. Ta se zobrazí v případě, že uživatel zvolí danou volbu v menu, nebo stiskne patřičnou klávesovou zkratku. Tato dialogová okna se ovládají pomocí jednodušších nástrojů, jako například, seznamů a tlačítek.

Třetí kategorii z výše uvedeného seznamu tvoří formuláře, které jsou otevírány uživatelem v případě, že si přeje zobrazit detail některé entity (případně údaje změnit) nebo vytvořit novou entitu. Pro oba účely vždy slouží stejný formulář. Pokud se jedná o zobrazení existující entity, formulář se podle jejího obsahu předem vyplní. Pokud uživatel chce vytvořit novou entitu, formulář zůstane prázdný (až na případy, kdy se snaží uživateli napovědět a předvyplní určité údaje, které vyplynuly z předchozího chování uživatele).

4.2.2 Rozhraní pro správu kontaktů

Toto rozhraní bylo navrženo jako dialogové okno, které se spustí z nabídky „Nástroje“ na liště. Dialogové okno se skládá ze seznamu aktuálně existujících kontaktů, stavové lišty a ovládacích tlačítek. Pro odebrání a úpravu kontaktu je třeba daný kontakt označit v seznamu. Při stisku tlačítka pro odebrání, se aplikace pokusí daný kontakt odstranit. V případě výskytu chyby během této operace, se chyba zobrazí na stavové liště. Při stisku tlačítka, pro přidání kontaktu, se zobrazí dialogové okno s prázdným formulářem, do kterého má uživatel zadat dané kontaktní údaje. Při stisku tlačítka pro úpravu kontaktu se otevře stejné okno, ale tentokrát bude již formulář vyplněný nastavenými údaji daného kontaktu.

Formulář pro přidání a úpravu kontaktu obsahuje textová pole s popiskami, která jsou určena k vyplnění. Každé textové pole má význam kontaktu, který je u něj popsán. Formulář také obsahuje stavovou lištu a potvrzovací a rušící tlačítko. V případě, že uživatel stiskne klávesu enter ve chvíli, kdy má nastavený kurzor klávesnice na některé textové pole, aplikace se zachová stejně, jakoby obsah formuláře potvrdil tlačítkem. Tímto chováním umožní vyplnit a potvrdit všechny údaje bez použití myši.

Podle toho, jestli se jedná o vytvoření nového nebo úpravu existujícího kontaktu, se aplikace pokusí provést danou operaci. Pokud akce skončí chybou, vypíše se chybová hláška na stavovou lištu. Pokud operace proběhne úspěšně, dialogové okno se automaticky uzavře.

4.2.3 Rozhraní pro správu úkolů

Z uvedených rozhraní se jedná o nejsložitější a nejpropracovanější. Jelikož spadá do kategorie často používaných nástrojů (viz výše), je součástí jednoho z vnitřních oken aplikace, které jsou neustále uživateli při ruce (nemusí se proklikávat skrze menu).

Toto rozhraní se skládá z několika základních prvků:

- Seznam naplánovaných úkolů
- Položka seznamu
- Vyskakovací nabídka
- Dialog výběru kategorie pro filtrování
- Formulář pro přidávání a úpravu kontaktů

Položkou seznamu je myšlena každá jedna grafická komponenta, která je zobrazena v seznamu (dá se vybrat). Tyto komponenty mají za účel graficky reprezentovat konkrétní úkoly. Jsou jim nastaveny tooltip texty, které vhodně informují o celém obsahu úkolu. Také umožňují, pomocí vyskakovací nabídky, provádět operace nad daným úkolem. Každá položka seznamu zobrazuje název úkolu, jeho termín ukončení (pokud je nastaven), Stav, ve kterém se úkol nachází, informaci o tom jestli je splněn a grafickou informaci specifikující kategorii. Aktuální stav úkolu je zobrazen prvním písmenem stavu. Barvou tohoto písmena je charakterizována kategorie úkolu. Informace o splněnosti je zobrazena pomocí zaškrtávacího políčka (které uživatel nemůže zaškrtnout přímo).

Vyskakovací nabídka je rozdělena do dvou oddělených částí. První část je zcela nezávislá na vybraném úkolu (přidání nového úkolu, nastavení filtračních informací pro zobrazení), zatímco druhá je přímo závislá na vybraném úkolu(úprava a odstranění úkolu, popřípadě označení úkolu za splněný). V případě, že je seznam prázdný, jsou dostupné pouze prvky první části nabídky.

Rozhraní umožňuje uživateli specifikovat 2 druhy filtrů, díky kterým umožní zobrazit pouze některé úkoly. Prvním filtrem je informace o splnění. Modul umožňuje zobrazit pouze nesplněné úkoly, případně všechny úkoly. Druhým filtrem je možnost volby konkrétní kategorie. Po zvolení této kategorie se zobrazují pouze úkoly, které pod danou kategorii spadají. Tento výběr se uskutečňuje pomocí dialogového okna pro výběr kategorie. Výběrem buď specifikujeme 1 konkrétní kategorii pro zobrazení, nebo umožníme zobrazení všech kategorií. Ihned po uskutečnění výběru se dialogové okno zavře.

Poslední komponentou rozhraní pro správu úkolů je formulář pro přidání a úpravu úkolu. Tento dialog obsahuje množství textových polí s popiskami, do kterých uživatel vyplňuje údaje o daném úkolu. Formulář opět obsahuje stavovou lištu, potvrzovací tlačítko a rušící tlačítko. Navíc obsahuje rozbalovací seznam, ve kterém vybere kategorii, do které kategorie úkol spadá. Po kliknutí na potvrzovací tlačítko (případně po stisknutí enteru, když je kurzor klávesnice na některém textovém poli) se aplikace pokusí o provedení operace přidání, resp. úpravy úkolu. V případě výskytu chyby se text dané chyby vypíše na stavovou lištu. Pokud je operace úspěšná, dialog s formulářem se zavře.

Kapitola 5

Implementace

V této kapitole se dozvíte o některých zvláštnostech a postupech využitých při implementaci projektu.

5.1 Úvod

V rámci týmové bakalářské práce Java EE Organizér proběhla implementace enterprise aplikace pro organizaci času a kontaktů uživatele. Jelikož se jedná o týmový projekt, byla práce na aplikaci rozdělena na 3 části. Pavel Palát pracoval na návrhu aplikace a tvorbě databáze. Jeho práce dále zahrnovala implementaci entitních tříd v serverové aplikaci a jejich provázání s databází a v rámci návrhu tvorbu rozhraní pro EJB komponenty. Jan Kovář měl za úkol návrh a implementaci modulu kalendář. Moje práce zahrnovala implementaci modulů úkoly a kontakty. Tvorba aplikace se skládala z implementace dvou prakticky samostatných aplikací a sice aplikačního klientu a serveru. V následujícím textu bude přiblížena implementace obou samostatných částí.

5.2 Implementace Aplikační vrstvy

V této sekci bude diskutován postup implementace business logiky, která běží na aplikačním serveru. Budou použity především pojmy popsané v sekci 3.2.

Business logika aplikační vrstvy se v Javě EE implementuje výhradně pomocí Enterprise JavaBean komponent (viz 3.2.2). V podstatě se tedy implementují rozhraní, která se mohou volat z prezentační vrstvy aplikačním klientem. Tyto implementace se oznámí běhovému prostředí pomocí anotace `@Stateless`. Důležitá rozhraní pro moduly kontaktů a úkolů byla rozhraní zajišťující správu kontaktů a úkolů. Těchto modulů se také dotýkají rozhraní spravující uživatelské operace, ale tato rozhraní byla implementována v rámci návrhu aplikace, jelikož jsou stěžejní pro celý systém, nejen pro tyto moduly.

Implementace rozhraní pro správu některé konkrétní entity se využívá `Entity Manager` (dále EM). Pomocí tohoto nástroje se na základě konkrétně specifikovaného dotazu či předem vytvořeného pojmenovaného dotazu v jazyce JPQL získávají informace z databáze. Pomocí zvláštních metod EM zajistí JPA odstraňování a úpravu existujících entit.

Implementace všech tří rozhraní jsou analogické a skládají se ze čtyř základních operací:

- uložení nové entity do databáze
- upravení existující entity v databázi

- odstranění existující entity v databázi
- získání odkazů na všechny entity dané třídy, které jsou přítomny v databázi

Všechny výše zmíněné operace jsou vázány na konkrétního uživatele. Z toho plynou omezení na dané operace a uživatel, který chce některou operaci provést musí uvést také svou identifikaci, kterou získá po přihlášení. Uživateli může být zamítnut přístup k daným operacím v případě, že se jedná o neexistujícího uživatele, nebo v případě, že se snaží změnit či odstranit entitu jiného uživatele, případně pokud se snaží změnit uživatele, kterému entita náleží. Každá operace je tedy složena ze série testů na oprávnění uživatele a následném provedení operace pomocí EM.

Pro všechna tři rozhraní je implementace analogická s výjimkou operace pro získávání kolekce referencí existujících úkolů, kde se kromě uživatele, kterému úkoly náleží předávají navíc informace pomáhající vyfiltrovat úkoly, jež nejsou pro klienta podstatné. Je možné vyfiltrovat úkoly, které již nejsou aktivní (jsou označené za splněné) a je možné vyfiltrovat úkoly, které nenáleží požadované kategorii. Jelikož je tato operace z principu aplikační, je prováděna na straně serveru (při implementaci jiného aplikačního klienta, případně webového rozhraní by se v prezentační vrstvě muselo filtrování vždy znovu implementovat, což je nežádoucí a z pohledu návrhu aplikace nevhodné).

5.3 Implementace Prezentační vrstvy

V této sekci naleznete informace o implementaci modulů úkoly a kontakty, na úrovni prezentační vrstvy. Dále bude nastíněna implementace i ostatních stěžejních součástí aplikace, které bylo nutné vyřešit. Budou použity především pojmy týkající se Swing API (viz 3.5) a okrajově pojmy technologií Javy EE (viz 3.2).

Při implementaci prezentační vrstvy jsem zvolil, ne příliš využívaný, přístup odvozování tříd implementujících grafické komponenty a jejich funkcionalitu od komponent rozhraní Swing. Tento přístup mi připadal, z hlediska objektového návrhu, čistší než obvyklý přístup definování tříd, které se na dané komponenty odkazují. Tento přístup však částečně „ztenčuje“ hranici oddělující implementaci funkcionality od grafické stránky aplikace. To může mít za následek mírnou nepřehlednost kódu.

5.3.1 Managery entit

Pro implementaci správy entitních objektů, na úrovni prezentační vrstvy, byly použity třídy se statickými prvky (statické metody a atributy). Všechny ostatní objekty pak mohli využívat těchto tříd a služeb, které poskytovaly. Přitom nebyly nuceny uchovávat reference na konkrétní instance managerů. Tento přístup byl obzvláště výhodný, jelikož se služby těchto tříd využívaly v metodách objektů různých tříd. Z toho důvodu by bylo velmi složité a nečisté, kdyby si objekty museli reference navzájem předávat. Navíc se tyto „managery“ mnohdy potřebují volat navzájem, kvůli vztahům mezi entitními množinami. Jedná se do značné míry o abstrakci EJB komponent z aplikační vrstvy. Tyto managery navíc umožňují lokální ukládání dat do paměti počítače, díky čemuž se značně snižuje komunikace přes síť. Kromě komunikace s EJB komponentami, kterým se zasílají vzdálené požadavky (přidávání nových entit apod.), se manažer stará o aktualizaci zobrazení patřičných grafických komponent. K této úloze však potřebuje referenci na instanci patřičného grafického kontejneru. Tento problém se řeší tak, že při inicializaci kontejneru se manageru předá jeho reference. Manager se pak o aktualizaci grafických kontejnerů stará jen v případě, že již má

nastavenou referenci. Data v těchto managerech jsou pravidelně aktualizována pomocí nezávisle běžícího vlákna. Vždy při aktualizaci manageru se manager opět postará o aktualizaci zobrazení.

5.3.2 Třída `Main`

Výhradní postavení v aplikaci má třída `Main`. Tato třída slouží, mimo jiné, ke spuštění aplikačního klienta (obsahuje metodu `main()`). Třída byla zvolena pro uchovávání údajů o přihlášeném uživateli (pomocí statického atributu). Díky tomuto přístupu k tomuto údaji může přistupovat celá aplikace, aniž by si všechny objekty musely předávat odkaz na konkrétní instanci. Tato třída navíc obsahuje statické atributy, které odkazují na EJB komponenty aplikační vrstvy (pomocí těchto atributů můžeme volat metody EJB komponent z aplikačního klienta). Třída `Main` tedy, z pohledu klienta, slouží jako jakási „brána“ do aplikační vrstvy.

5.3.3 Komponenty hlavního okna aplikace

Grafické rozložení hlavního okna aplikace bylo implementováno pomocí následujících komponent:

- `JFrame` – Top-level kontejner, reprezentující hlavní okno aplikace (v aplikaci byla použita třída odvozená od `JFrame`).
- `JInternalFrame` – Kontejner použitý pro implementaci vnitřních oken aplikace, které obsahují často využívané nástroje (v mém případě především rozhraní pro zobrazení a správu úkolů).
- `JMenuBar` – Kontejner použitý pro lištu hlavní nabídky.
- `JMenu` – Kontejner použitý pro konkrétní nabídky.
- `JMenuItem` – Komponenta použitá pro jednotlivé volby.

Implementace plně funkčních vnitřních oken si vyžadovala použít speciální druh kontejneru, který se nazývá `JDesktopPane`. Této komponentě se nastavil handler, který obsluhoval událost změny velikosti kontejneru. Ten v případě, že uživatel zvolil volbu udržování tvaru rozložení vnitřních oken spočítá pro dané rozměry nové hranice vnitřních oken a změní také jejich pozice a velikost tak, aby byly ve stejném poměru jako při spuštění aplikace. Obdobné handlery byly (ze stejného důvodu) nastaveny i všem vnitřním třídám pro události vyvolaných při změnách pozice nebo velikosti.

Při implementaci hlavního okna byl použit nástroj pro tvorbu grafických komponent Swingu *Matisse*, který je součástí vývojového prostředí *NetBeans IDE*. Tento nástroj značně ulehčil tvorbu hlavního okna i některých jiných grafických komponent (především formulářů).

5.3.4 Popup menu

Vyskytl se požadavek, aby uživatel mohl ovládat některé komponenty (či moduly) skrze popup menu[1]. Jelikož bylo vhodné tento druh komponenty implementovat pro různé moduly,

[1] Popup menu – Vyskakovací nabídka (nabídka, která se zobrazí typicky při pravém kliknutí myší na určitou komponentu)

byla zvolena implementace této komponenty do samostatné třídy rozšiřující `JPopuMenu` (grafická komponenta Swingu představující popup menu). Uvnitř této třídy, byly vytvořeny vnitřní třídy (*inner classes*) implementující funkcionalitu jednotlivých položek nabídky.

Jelikož je třída reprezentující popup menu využívaná z více zdrojů a podporuje operace nad komponentami z různých modulů, přidaly se do třídy inicializační metody, které zajišťovaly správné naplnění nabídky jednotlivými položkami. Každá taková inicializační metoda se pak vztahuje k určitému účelu použití a k jednomu konkrétnímu modulu (Např. pro inicializaci nabídky s položkami pro správu úkolů se použije jiná metoda, než pro inicializaci nabídky pro správu událostí). V rámci mé bakalářské práce proto byla implementována především metoda pro inicializaci nabídky pro správu úkolů. S tím byla spojena i implementace vnitřních tříd, které zajišťovaly jednotlivým položkám jejich funkcionalitu.

Použití třídy pro reprezentaci popup menu se typicky skládá ze tří etap:

- Vytvoření nové instance třídy
- Přidání položek nabídky pomocí patřičné inicializační metody
- Zobrazení komponenty v místě, kam uživatel klikl myší (Toto umístění je třeba komponentě specifikovat v metodě `show()`.

5.3.5 Rozhraní pro správu úkolů

Uživatelské rozhraní pro správu úkolů je do základního rozložení aplikace začleněno jako komponenta obsahující seznam existujících úkolů uvnitř jednoho ze tří vnitřních oken aplikace (viz 4.2.3). Tato komponenta je vnitřnímu oknu nastavena jako jeho obsah (content pane). Komponenta rozšiřuje třídu `JList`. Položky seznamu jsou implementovány jako komponenty rozšiřující `JPanel`, které jsou uložené v modelu seznamu (poněkud netradiční způsob, jelikož se do modelu seznamu typicky ukládají pouze data, která komponenta reprezentuje). O zobrazování položek se stará vnitřní třída seznamu, která implementuje rozhraní `ListCellRenderer`.

Položka seznamu se (po grafické stránce) skládá ze tří komponent:

- Zaškrtávací pole s jednoznakovým popiskem
- Popisek zobrazující název úkolu
- Popisek zobrazující termín splnění
- tooltip poskytující všechny informace o úkolu

Zaškrtávací pole s popiskem bylo implementováno pomocí komponenty `JCheckBox` a zobrazuje informaci o stavu, ve kterém se úkol nachází. Úkol může mít tři stavy (ještě nezačal, aktuální nesplněný úkol a splněný úkol). Aplikace zobrazí políčko jako zaškrtlé v případě, že je úkol splněný. Informaci o konkrétním stavu, ve kterém se úkol v současné chvíli nachází, je pak naznačen jedním znakem v popisku políčka (A – ještě nezačal, W – aktuální nesplněný úkol, C – splněný úkol). Pomocí barvy popředí (forward color) je implementováno zobrazení informace o kategorii, pod kterou úkol spadá.

Tooltip položky je implementován pomocí pohodlného nástroje, který swing u svých komponent nabízí a sice použití HTML[2] tagů. Tento tooltip zobrazuje téměř všechny dostupné informace o úkolu.

[2] HTML – HyperText Markup Language

Zobrazení vyskakovací nabídky pro správu úkolů je implementováno v rámci třídy pro položku seznamu, jako součást handleru reagujícího na událost myši. Jelikož však seznam tyto události odchytává sám, je potřeba v handleru seznamu zajistit zvolání i handleru položky, která je v seznamu vybrána. Tím, že překrýváme handler seznamu, také získáme možnost zavolat vyskakovací nabídku přímo ze seznamu v případě, že je seznam prázdný a žádná položka se v něm nenachází. Tyto nabídky se liší množstvím zobrazených položek (Nabídka vyvolaná z prázdného seznamu neobsahuje položky pro správu existujícího úkolu.).

Formulář pro přidávání a úpravu úkolů je opět podobný, jako pro kontakty. Výběr kategorie, pod kterou má daný úkol spadat je implementován pomocí komponenty třídy `JComboBox`, které je nastaven model, do kterého z aplikační vrstvy načteme všechny kategorie, které má uživatel k dispozici.

Kapitola 6

Závěr

Práce byla vytvořena se záměrem charakterizovat dvě rozsáhlé a často využívané technologie (Java EE a Swing) a nastínit způsob práce s nimi. Také se snaží nastínit problematiku organizačních nástrojů a způsobů, jak je efektivně implementovat pomocí platformy Java EE. Záměrem této práce bylo také nastínit výhody a nedostatky současných aplikací z této oblasti, které mají poměrně velký prostor pro vylepšování. Tato oblast byla zvolena kvůli nedostatku kvalitních serverových aplikací poskytujících specializovanou klientskou aplikaci. Bylo naznačeno, že je mnohdy lepší vytvořit jednoduchý nástroj, než příliš robustní a složitý systém.

Dalším přínosem práce je demonstrace, jak lze vytvářet grafické aplikační klienty pro platformu Java EE. Jelikož se tato technika nevyužívá tak často, jako tvorba enterprise aplikací s webovým rozhraním, je tato demonstrace poměrně cenná.

V rámci práce byla vytvořena aplikace Java EE Organizér, která umožňuje efektivní správu kontaktů, úkolů a událostí. Umožňuje úkoly a události organizovat do uživatelem vytvořených kategorií. Aplikace podporuje pravidelná opakování událostí, spravování stavu úkolů a mechanismus upozorňování na úkoly. Aplikaci tvoří dvě samostatné části – aplikační klient a serverová aplikace.

Aplikace Java EE Organizér by se dala rozšířit o širokou škálu nových funkcí. Zajímavým by například mohlo být rozšíření o webové rozhraní, které by umožňovalo používat aplikaci i např. z mobilních zařízení. Toto rozšíření by se přitom dotklo pouze prezentační vrstvy. Naproti tomu aplikační a datová vrstva by mohly zůstat beze změny. Mezi další možná rozšíření je možné zařadit mechanismus upozorňování na události pomocí zasílání emailu, případně implementace kompletního modulu pro správu emailů (tento zásah by se dotkl jak aplikační, tak prezentační vrstvy). Jedním z nejlákavějších rozšíření je však možnost propojení správy činností mezi uživateli. Tím by bylo možné aplikaci posunout až na hranici groupware systému.

Během této bakalářské práce jsem získal poměrně cenné zkušenosti s technologiemi platformy Java EE a jejich použitím. Zjistil jsem, že významně urychlují proces vývoje vícevrstvých aplikací, díky zapouzdření technologií pro síťové přenosy, mapování objektů do databáze, nebo i automatickým vytvářením a rušením EJB komponent. Obzvláště přínosná byla zkušenost s implementací platformně nezávislého aplikačního klienta pomocí technologie Swing. Zjistil jsem také, že mnou zvolený přístup programování uživatelského prostředí (implementace funkcionality aplikace přímo do třídy rozšiřující grafickou komponentu) není zcela optimální a že konvenční přístupy jsou mnohdy čistší a přehlednější.

Literatura

[1] *Firebird – The RDBMS that's going where you're going* [online]. 2009-05-07 [cit. 2009-05-09].
URL http://www.firebirdsql.org/

[2] *GlassFish – Open Source Application Server* [online]. 2009-05-02 [cit. 2009-05-09].
URL https://glassfish.dev.java.net/

[3] HYBÁŠEK, S.: *Technologie a nástroje vývoje Java EE aplikací*. Diplomová práce, Masarykova univerzita, Fakulta informatiky, Brno, 2007.

[4] *Třívrstvá architektura* [online]. 2006-04-19 [cit. 2009-05-09].
URL http://www.itexpert.cz/trivrstva-architektura/

[5] *Java EE at a Glance* [online]. 2009-05-04 [cit. 2009-05-09].
URL http://java.sun.com/javaee/

[6] *Java EE Technologies at a Glance* [online]. 2009-05-04 [cit. 2009-05-09].
URL http://java.sun.com/javaee/technologies/

[7] JENDROCK, E.; BALL, J.; CARSON, D.; aj.: *The Java EE 5 Tutorial: For Sun Java System Application Server 9.1* [online]. Santa Clara: Sun Microsystems, Inc., October 2008 [cit. 2009-05-08], ISBN 819-3669-11.
URL http://java.sun.com/javaee/5/docs/tutorial/doc/

[8] KOVÁŘ, J.: *Java EE Organizér – modul kalendář*. Bakalářská práce, FIT VUT v Brně, Brno, 2009.

[9] PALÁT, P.: *Java EE Organizér – softwarová architektura*. Bakalářská práce, FIT VUT v Brně, Brno, 2009.

[10] Sun Microsystems, Inc.: *Trail: Creating a GUI with JFC/Swing* [online]. 2008-02-14 [cit. 2009-05-10].
URL http://java.sun.com/docs/books/tutorial/uiswing/

[11] SunMicrosystems, Inc.: *Sun Java System Application Server 9.1 Developer'sGuide* [online]. 2008 [cit. 2009-05-09.
URL http://docs.sun.com/app/docs/doc/819-3672

[12] Wikipedia: *Java Platform, Enterprise Edition* [online]. 2009-05-09 [cit. 2009-05-09].
URL http://en.wikipedia.org/wiki/Java_EE

[13] Zaachi: *Java a základy GUI* [online]. 2008-01-06 [cit. 2009-05-09].
URL http://www.zaachi.com/cs/items/java-a-zaklady-gui.html

Dodatek A

Obsah CD

Přiložené CD má následující adresářovou strukturu:

- `source` – Adresář s kompletním řešením aplikace (zdrojové kódy, xml soubory, NetBeans projekty)
- `manual` – Adresář s uživatelským manuálem ve formátu HTML
- `thesis` – Adresář s technickou zprávou ve formátu PDF i se všemi soubory a zdrojovými kódy, které jsou zapotřebí pro překlad dokumentu