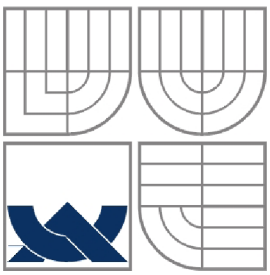
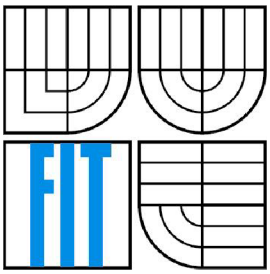


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

FRAMEWORK PRO MODULÁRNÍ APLIKACE NAD KNIHOVNOU SWING

FRAMEWORK FOR MODULAR SWING APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
TOMÁŠ SÁGHY

VEDOUCÍ PRÁCE
SUPERVISOR
Doc. RNDr. Pavel Smrž, Ph.D.

BRNO 2009

Abstrakt

Práce se zabývá vytvořením frameworku pro modulární GUI aplikace v jazyce Java, která využívá ovládání pomocí pásu karet. V první části práce jsou porovnané tři existující frameworky pro tvorbu GUI aplikací. Pak následuje popis použitých komponent, jako Flamingo (zobrazuje pás karet), sada komponent JIDE (práce využívá hlavně JIDE Docking Framework pro správu panelů aplikace), JBusyComponent (zobrazuje zaneprázdněný stav libovolné komponenty). Dále je popsána specifikace OSGi, která je využita pro dynamické spouštění modulů. Implementační část popisuje části a použití frameworku a popisuje nabízené služby.

Abstract

Paper deals with framework design for modular GUI applications in Java, which uses ribbon. In the first part of the paper there are three frameworks for GUI application design introduced. Then the applied components are characterized: the Flamingo (displays ribbon), JIDE component family (paper deals especially with JIDE Docking Framework for application panel management) and JBusyComponent (displays busy state of any component). This is followed by specification of OSGi, which is used for dynamic module startup. In the implementation part of the paper there is a description of the parts, the use of frameworks and the offered services.

Klíčová slova

pás karet, OSGi, Swing, modulární aplikace, služba, framework

Keywords

ribbon, OSGi, Swing, modular application, service, framework

Citace

SÁGHY, Tomáš. *Framework pro modulární aplikace nad knihovnou Swing*. Brno, 2009. Bakalářská práce na FIT VUT v Brně.

Framework pro modulární aplikace nad knihovnou Swing

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Doc. RNDr. Pavel Smrž, Ph.D.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Tomáš Sághy
19. května 2009

Poděkování

Děkuji za cenné rady a odborné vedení Doc. RNDr. Pavel Smrž, Ph.D.

Obsah

Seznam obrázků

Seznam příkladů

1 Úvod

Rapidním vývojem výpočetní techniky se aplikace stávají čím dál složitější a kladou vysoké nároky na vývojové týmy, které musí zápasit s časem a být schopní přizpůsobit aplikace různým a stále se měnícím požadavkům uživatele. Rozdělením aplikace na menší, lépe zvládnutelné celky – moduly – je vývoj jednodušší a rychlejší, navíc se přímo nabízí znovupoužití některých modulů ve více projektech. Moduly mohou být vyvíjeny a testovány nezávisle různými týmy.

Firma Microsoft Corporation v produktu Office 2007 přinesla nový způsob ovládání aplikací, tzv. pás karet, která nahrazuje hlavní menu aplikace a panely nástrojů a nabízí všechny ovládací prvky na jednom místě.

Programovací jazyk Java je díky svým možnostem velmi dobrou volbou při vývoji většiny GUI aplikací. Díky tomu, že aplikace běží v Java Virtual Machine, je přenositelný a ten samý program funguje beze změny na různých operačních systémech.

Bakalářská práce se zabývá návrhem a implementací frameworku pro modulární GUI aplikace využívající knihovnu Swing. Framework s názvem Module Suite Framework X (MSFX) vznikl ve spolupráci s firmou WetCom Databases s.r.o. pro podporu plánovaného rozsáhlého projektu. Využívá pás karet na které jednotlivé moduly vkládají ovládací prvky. Dále poskytuje pokročilou správu panelů aplikace, která umožňuje přeuspořádat panely v hlavním okně aplikace podle potřeby uživatele. Moduly je možné spouštět za běhu, což dává zajímavé možnosti aktualizace a distribuce aplikace.

V první části práce se zaměříme na popis tří existujících frameworků pro tvorbu modulárních GUI aplikací: Swing Application Framework (SAF), Netbeans Platform a Eclipse Rich Client Platform (Eclipse RCP). Z nich detailněji popíšeme Swing Application Framework, která je využita v našem projektu.

V další části se věnujeme některým technologiím jazyka Java, které jsou v naší aplikaci využity. Jedná se o knihovnu Swing (knihovna GUI komponent), použití anotací (přidávání metainformací k elementům jazyka Java) a použití reflexe (přístup k elementům Java tříd a přístup k metainformacím).

Dále jsou popsané GUI komponenty třetích stran, které náš framework používá a standard OSGi, která zabezpečuje běhové prostředí pro modulární aplikaci. Tyto knihovny jsou: Flamingo (zobrazení pásu karet), JBusyComponent (indikace zaneprázdněnost libovolné GUI komponenty), sada komponentů JIDE a komerční produkt JIDE Docking Framework (správa oken aplikace).

Druhá půlka práce popisuje samotnou implementaci frameworku, nabízené možnosti a způsob použití.

1.1 Cíle práce

Cílem práce bylo vytvořit framework pro modulární GUI aplikace napsané v jazyce Java, která využívá pás karet a knihovnu Swing. Součástí práce bylo prostudování existujících řešení, nalézt a prostudovat potřebné komponenty, prostudovat specifikaci OSGi a podle získaných zkušeností navrhnout a implementovat framework.

Systém by měl umožnit vyvíjet moduly nezávisle jako samostatné projekty. Moduly definují své ovládací prvky na pásu karet – framework zobrazí pás karet podle spuštěných modulů. Moduly mají komunikovat pomocí služeb, případně použitím (na to určeného) kódu z jiného modulu – izolování modulů a komunikace pomocí služeb je hlavním rysem servisně orientované architektury

(SOA). Navrhnutý framework by měl být připraven umožňovat aktualizaci modulů z internetu, případně stahování modulů podle potřeby. Pomocí standardu OSGi se zabezpečí izolace modulů a jejich dynamické načtení.

Aplikace vytvořena pomocí navrhnutého frameworku by měla být spustitelná jednak jako standardní Java aplikace, ale i prostřednictvím technologie Java Web Start z internetu. Dále aplikace by měla být spustitelná bez použití OSGi, aby bylo možné aplikaci ladit.

Navrhnutý a implementovaný framework je třeba demonstrovat na ukázkové aplikaci.

2 Přehled existujících frameworků pro tvorbu GUI aplikací

V této kapitole si představíme 3 frameworky pro vývoj komplexních GUI aplikací. První z nich (SAF – Swing Application Framework) je jednoduchý a ostatní dva (NetBeans Platform, Eclipse RCP) jsou velmi komplexní a nabízejí přibližně stejné možnostmi.

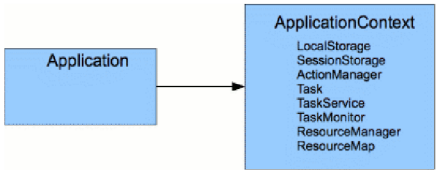
Obě komplexní frameworky vznikli původně jako integrované vývojové prostředí (IDE) a nejznámější aplikace, které je používají jsou právě ty IDE (Netbeans IDE, Eclipse IDE). Obě jsou postavené tak, že aplikace pomocí nich vytvořená se skládá z modulů. Moduly spolu komunikují prostřednictvím služeb – používají služby jiných modulů a nabízejí vlastní služby ostatním modulům. Jeden z hlavních rozdílů mezi nimi je to, že jakou knihovnu GUI komponent používají. Netbeans Platform používá Swing komponenty, které jsou součástí všech distribucí Java SE; naproti tomu Eclipse RCP používá knihovnu SWT (Standard Widget Toolkit)[1], která zobrazuje komponenty pomocí operačního systému, proto vyžaduje nativní knihovnu pro daný operační systém. Porovnání dvou frameworků lze najít například v [3].

Narozdíl od komplexních frameworků SAF nezavádí moduly ani služby, řeší jen základní věci potřebné k běhu složitější GUI aplikace. V našem frameworku budeme využívat právě SAF kvůli její jednoduchosti. Dalšími důvody pro jejich nepoužití jsou: aplikace napsaná pomocí dvou zmíněných frameworků vypadají podobně jako k nim patřící IDE; a kvůli jejich komplexnosti vyžadují delší učící cyklus; také nenabízí možnosti použití Ribbonu, což bylo jeden z hlavních cílů projektu.

2.1 Swing Application Framework

Swing Application Framework [2] je jednoduchý, ale plnohodnotný framework pro vytváření GUI aplikací. Je referenční implementací Java Specification Request (JSR 296), která je stále ve vývoji a bude součástí standardní distribuce Javy SE 7. Zabezpečuje základní architekturu GUI aplikace využívající knihovnu Swing; nabízí základní funkce, které jsou využívány ve většině aplikací. Tyto funkce zahrnují správu akcí, správu a sledování stavu dlouhotrvajících úloh, použití lokalizovaných zdrojů, ukládání/načtení stavu aplikace a životní cyklus celé aplikace.

Třídy `Application` a `ApplicationContext` jsou hlavními třídami aplikace postavené na tomto frameworku (viz obrázek 1). Aplikace definuje třídu zděděnou od třídy `Application`, která zabezpečuje životní cyklus aplikace. Třída `ApplicationContext` poskytuje výše zmíněné funkce pro aplikaci.



Zdroj: převzato z http://java.sun.com/developer/technicalArticles/javase/swingappfr/

Obrázek 1: architektura SAF

Aplikace se spouští voláním statické metody `Application.launch` a ukončuje voláním `Application.exit`, které zabezpečují vykonání všech potřebných metod životního cyklu

[1] webová stránka SWT: http://www.eclipse.org/swt/

aplikace. Metody životního cyklu je možné (některou povinně) předefinovat v třídě aplikace. Tyto metody jsou:

- **initialize** – slouží pro provedení inicializace ještě před samotným zobrazením GUI
- **startup** – slouží na inicializaci a zobrazení GUI, tuto metodu třeba povinně předefinovat
- **ready** – je volána, když je inicializace a zobrazení GUI úplně dokončeno
- **shutdown** – slouží na skrytí GUI a provedení akcí před samotným ukončením aplikace

Příklad 1 ukazuje minimální kód, který inicializuje a spustí SAF a která zobrazí okno se zadaným textem.

```
public class SAFExampleApp extends Application {
  protected void startup() {
    //hlavní okno aplikace
    JFrame window = new JFrame("Simple APP title");
    //konfigurace okna
    window.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    //přidáme posluchač, který ukončí aplikaci při zatvoření okna
    window.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        exit();//volání ukončení aplikace
      }
    });
    //přidáme do okna text
    JLabel label = new JLabel("Hello World");
    window.add(label);
    window.pack();
    //zobrazení okna
    window.setVisible(true);
  }

  public static void main(String[] args) {
    Application.launch(SAFExampleApp.class, args);
  }
}
```

Příklad 1: inicializace a spuštění SAF aplikace (okno s textem)

Správa zdrojů (texty, popisky, ikony, barvy, atd.) zajišťuje třída `ResourceManager`. Je implementováno pomocí resource bundles[2] – což zabezpečuje možnost lokalizace a internacionalizace výsledné aplikace. Každá třída (např. okno, panel) má v podadresáři resources soubor `*.properties` se stejným jménem jako třída. Od `ApplicationContext` (nebo `ResourceManager`) je možné získat `ResourceMap` podle třídy, a z něho potřebné resource. Podle typu požadovaného zdroje se provede konverze (např. když se požaduje ikona, tak hodnota určuje cestu a název souboru z které se samotná ikona načte). Použití je vidět na příkladu 2 (kód) a 3 (resource soubor).

```
ResourceMap rm = Application.getInstance().getContext()
                                  .getResourceMap(this.getClass());
JButton btn1 = new JButton();
btn1.setText(rm.getString("btn1.test"));
JLabel label1 = new JLabel();
label1.setText(rm.getString("label1.text"));
label1.setForeground(rm.getColor("label1.foreground"));
label1.setFont(rm.getFont("label1.font"));
```

Příklad 2: použití zdrojů z resource souborů v SAF

[2] textový soubor obsahující na řádcích klíč a hodnotu; pomocí přípon názvu souboru se zadávají hodnoty pro jiné jazyky; pro podrobnosti viz http://java.sun.com/j2se/1.4.2/docs/api/java/util/ResourceBundle.html

Pokud je třída aplikace zděděna od `SingleFrameApplication` (rozšiřuje třídu `Application`), tak existuje možnost automatického doplnění resources do komponenty v případě, že je zobrazení provedeno metodou show z `SingleFrameApplication`.

```
label1.text=jLabel1
label1.font=Arial-Bold-10
label1.foreground=255, 51, 51
```

Příklad 3: ukázkový resource soubor k příkladu 2

SAF přináší vylepšenou správu GUI akcí. Akce se definují jako metody s anotací `@Action`. K anotaci je možné v resource souboru definovat malou a velkou ikonu, text, titulek, klávesovou zkratku. Při přiřazení akce k nějaké komponentě se tyto hodnoty nastaví do něho. Dále je možné specifikovat k akci název dvou boolean proměnných: první určuje, zda je akce povolená; druhá určuje, zda je aktivní (např. v případě checkbox-u zda je zaškrtnuto). Tyto proměnné se aktualizují (musí vyhovovat standardu JavaBeans[3]), pokud se změní stav GUI komponenty a naopak.

Akce je možné přiřadit i k více komponentám najednou a tím je sdílet (např. do nástrojové lišty a do hlavního menu okna).

Správu akcí zabezpečuje třída `ActionManager`, která v případě prvního použití akce z dané třídy načte akce podle anotací. Od `ApplicationContext` (nebo `ActionManager`) je možné získat seznam akcí (`ActionMap`) pro danou třídu (nebo globální akce) a z něho samotné akce podle jména. Jednoduché použití akcí je vidět na příkladu 4, který demonstruje deklaraci akce a její přiřazení k tlačítku.

```
public void actionExample() {
  ActionMap am = Application.getInstance().getContext().getActionMap(this);
  //vytvoříme tlačítko (text se bere z resource souboru k akci)
  JButton btn = new JButton();
  btn.setAction(am.get("testAction"));
}

@Action
public void testAction() {
  //kód akce
}
```

Příklad 4: deklarace a použití akce v SAF

Pokud je potřeba vykonat dlouhotrvající kód, tak je třeba ho vykonat v jiném vlákně, aby se neblokovalo vlákno EDT, který má na starosti překreslování komponent a zpracování událostí (viz kapitola 3.1 Swing). Framework nabízí na tento účel třídu `Task`, která umožňuje vykonání kódu v jiném vlákně s tím, že je možné předem získat/modifikovat stav GUI a na konci provést aktualizaci GUI podle výsledku. Třída `Task` je rozšířením třídy `SwingWorker`[4] s podporou sledování stavu běžících úloh. Pomocí ní se spouští úlohy na množině vláken (tread pool) – vytvořené vlákna se recyklují a tím se ušetří systémové zdroje na vytváření nového vlákna při každé úloze. Navíc je možné sledovat stav běžících úloh na pozadí pomocí třídy `TaskMonitor` – lze registrovat naslouchač, která bude informovaná o změně stavu běžících úloh. Sledování stavu je užitečné například na indikaci probíhající úlohy v stavovém řádku. Příklad 5 ukazuje deklaraci a spouštění dlouhotrvající úlohy s informováním o průběhu (výpočet simuluje pomocí čekání).

[3] musí poskytovat get a set metody pro všechny vlastnosti a musí podporovat boud properties (viz http://java.sun.com/docs/books/tutorial/javabeans/properties/bound.html)
[4] viz například http://en.wikipedia.org/wiki/SwingWorker

```
void startTask() {
  Application app = Application.getInstance();
  app.getContext().getTaskService().execute(new TestTask());
}

//třída, s dlouhotrvajícím kódem
class TestTask extends Task<String, Void> {
  public TestTask() {
    super(Application.getInstance());
  }

  protected String doInBackground() throws Exception {
    //dlouhotrvající výpočet - jiné vlákno
    for (int i = 0; i < 100; ++i) {
      setProgress(i);//informace o průběhu - nepovinné
      Thread.sleep(100);//simulace výpočtu
    }
    setProgress(100);
    return "result";
  }

  protected void succeeded(String result) {
    //dokončení výpočtu - v EDT
    System.out.println("Result = " + result);
  }

  protected void failed(Throwable cause) {
    //došlo k chybě během výpočtu - v EDT
  }
}
```

Příklad 5: spuštění a deklarace dlouhotrvající akce v SAF

Dále SAF nabízí možnost uložení stavu (libovolných hodnot) aplikace pomocí `LocalStorage`. Data z jednotlivých tříd se uloží pomocí XML serializace do souboru v adresáři podle zvyklostí daného operačního systému. Pomocí `SessionStorage` je možné automaticky uložit stav GUI (pozice a velikost oken, šířka sloupců tabulky, atd.) a při spuštění se stav obnoví.

GUI editor v NetBeans IDE podporuje vytváření aplikace pomocí SAF: podporuje editování akcí, texty ukládá automaticky do resource souborů. Při vývoji je to velmi užitečné a urychluje práci.

2.2 NetBeans Platform

NetBeans Platform je velmi komplexní framework pro vytváření modulárních GUI aplikací postavených na knihovně Swing (viz [10]). Nejznámější aplikace, která ji používá je NetBeans IDE – jedna z nejpoužívanějších vývojových prostředí pro Javu. Jeho počátky sahají do roku 1996, kdy vznikl studentský projekt na MFF UK v Praze, z kterého později vznikla společnost, která vyvíjela komerční NetBeans IDE (viz [1]). V roce 1999 byla společnost koupena firmou Sun Microsystems[5], která vyvíjí samotnou Javu. Následně byli uvolněny zdrojové kódy a dodnes je open-source projektem. Postupem času začali vznikat různé aplikace postavené na NetBeans IDE, to vyvolalo vytvoření NetBeans Platform (v současnosti NetBeans IDE je jen jedna z aplikací postaveno nad platformou). Je součástí JDK od verze JDK 6 update 7, a je na něm postavena utilita VisualVM na sledování stavu běžících JVM, která je také součástí JDK (viz např. [3]).

Aplikace využívající NetBeans Platform je postavená z modulů. Moduly jsou JAR archívy se speciálním Manifest[6] souborem, která určuje název a verzi modulu, požadované jiné moduly atd. Moduly spolu komunikují pomocí služeb, které je možné dohledat buď standardním způsobem,

[5] webová stránka: http://www.sun.com/

[6] speciální soubor s metainformacemi o Java archívu, umístění v /META-INF/MANIFEST.MF, pro bližší informace: http://java.sun.com/docs/books/tutorial/deployment/jar/manifestindex.html

kterou nabízí JRE od verze 1.6, nebo přes virtuální souborový systém. První způsob využívá soubory v META-INF/services/ a třídu `ServiceLoader`[7] pro nalezení implementace pro hledanou službu. Soubor pojmenován jako celý název třídy hledané služby obsahuje názvy tříd, které implementují danou službu.

Framework podporuje načtení a doinstalování nových modulů za běhu a také nabízí možnost automatické aktualizace verzí z internetu. Existuje mnoho hotových modulů, které je možné v nových aplikacích použít a tím snížit náklady a urychlit vývoj.

Využívá se virtuální souborový systém, který obsahuje fyzické soubory, soubory ze ZIP archivů, třídy aplikace nebo i záznamy v databázi. Každý modul může přidat vlastní část souborového systému. Souborový systém je využit také ke komunikaci mezi moduly. Framework nabízí GUI komponenty pro zobrazení a editaci vlastností částí virtuálního souborového systému – což umožňuje pohodlné zobrazení a editaci stromově strukturovaných dat.

Disponuje propracovanou podporou správy umístění panelů v rámci hlavního okna – tzv. docking framework. Dovoluje uspořádání panelů v rámci okna, proto může uživatel přeorganizovat okno podle svých potřeb. Panely je možné dávat na 4 strany hlavního okna i libovolného jiného panelu. Dále také umožňuje dát více panelů na jedno místo a přepínat je pomocí záložek (tabs). Samozřejmostí je, že rozmístění panelů si aplikace pamatuje.

2.3 Eclipse Rich Client Platform

Eclipse RCP je další z komplexních frameworků pro vytváření modulárních GUI aplikací. Nabízí v podstatě stejné možnosti jako NetBeans Platform. Je vyvíjen jako open-source projekt firmou Eclipse Foundation a její vývoj podporuje firma IBM (viz [6]). Je na něm postaven známé vývojové prostředí Eclipse. Z vývojové prostředí se oddělila platforma pro tvorbu libovolných aplikací až v pozdějších verzích.

Narozdíl od NetBeans Platform, který používá knihovnu Swing pro zobrazení GUI komponent, Eclipse RCP používá knihovnu SWT. Díky němu mají aplikace nativní vzhled (taký, jaký je na daném operačním systému obvyklé) na všech platformách. Nevýhodou je, že vyžaduje knihovnu pro každý operační systém – není čistě Javovské řešení. Toto zkomplikuje spuštění a instalaci vytvořené aplikace pomocí technologie Java Web Start[8] na klientském počítači. Knihovnu SWT rozšiřuje knihovna JFace[9], která přináší použití GUI elementů na vyšší úrovni – zavádí architekturu Model-View-Controller.

K správě modulů používá standard OSGi (viz kapitola 4.4 OSGi), konkrétně její implementaci s názvem Equinox (viz [9]), která je referenční implementací standardu. Moduly jsou OSGi bundle. Každý modul může definovat body rozšíření a používat body rozšíření jiných modulů.

Samozřejmostí je automatická aktualizace modulů z internetu, nebo propracovaný správce oken – docking framework.

---

[7] viz http://java.sun.com/javase/6/docs/api/java/util/ServiceLoader.html
[8] spouštění Java aplikace z webové stránky, detaily viz http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html
[9] viz http://en.wikipedia.org/wiki/JFace

3 Použité technologie

V této kapitole popíšeme tři použité technologie v našem frameworku. První z nich je standardní GUI knihovna pro tvorbu uživatelského rozhraní (Swing), potom následuje popis způsobu přidávání metainformací k jednotlivých částem Java tříd (anotace) a způsob zpřístupnění proměnných, metod tříd a metainformací tříd (reflexe).

3.1 Swing

Swing [4] je základný GUI framework pro Java (J2SE). Je rozšířením/náhradou starší AWT. AWT používá komponenty nabízené operačním systémem na zobrazení ovládacích prvků a pomocí nich probíhá interakce s uživatelem – jde o tzv. heavyweight komponenty. Swing komponenty jsou plně zobrazeny v Javě a interakce s uživatelem je také v Javě – tzv. lightweight komponenty. Toto umožňuje jednotný vzhled aplikace na různých platformách. Navíc umožňuje měnit výzor a chování (look-and-feel) aplikace bez zásahu do kódu, také měnit vzhled za běhu (Pluggable look-and-feel). Je možné nastavit vzhled a chování aplikace podle operačního systému, nebo použít libovolný jiný.

Swing je postaven na modifikovaném MVC (Model-View-Controller) architektuře. Data komponentů jsou odděleny a uloženy v modelech. Zobrazená data se získají z modelu a při modifikaci se modifikují údaje v modelu. Samotné zobrazení komponenty a část řízení je delegováno na tzv. UI delegát. Tento UI delegát je součástí aktivního look-and-feel – tímto je docílena možnost měnit vzhled a chování aplikace i za běhu.

GUI aplikace je řízena událostmi – tj. je asynchronní. Uživatelská/jiná akce vyvolá událost (např. stlačení tlačítka, stisk klávesy, pohyb myši, atd.) o které je aplikace informována a na tuto událost reaguje. K jednotlivým typům události je možné u komponenty registrovat posluchače událostí (Listener – třída implementující dané rozhraní). Registrované posluchače jsou informováni o výskytu dané události komponentu.

Veškeré zobrazování a správa události probíhá v jednom vlákně (EDT – Event-Dispatching Thread), protože knihovna není thread-safe. To znamená, že nesmí se měnit vlastnosti komponent z jiného vlákna. Události se také vyvolají v tomto vlákně, proto obsluha události musí být co nejkratší, aby se neblokovalo zpracování dalších události a překreslování okna. Pokud je třeba vykonat delší operace, třeba to provést v jiném vlákně. Swing poskytuje možnosti jak spustit část kódu ve vlákně zpracování událostí tím, že se třída s daným kódem přidá do fronty a bude vykonán později.

3.2 Anotace

Umožňují přidání dodatečných informací k jednotlivým elementům zdrojového kódu (viz v [18]). Tuto možnost přináší Java 5. Anotace je možné přidávat k deklaracím typů (třídy, interface, výčtový typ – enum, anotace), polím a metodám třídy, konstruktorům, parametrům metod, lokálním proměnným a balíkům. Máme možnost definovat vlastní anotace.

Anotace nemění význam zdrojového kódu. Mohou být zpracovány kompilátorem například na generování dalších souborů. Pokud je specifikováno, tak anotace se uloží i do přeloženého souboru (`*.class`). Tyto anotace mohou být zpracováni JVM nebo jiným programem. Navíc je možné k anotacím přistupovat za běhu programu pomocí reflexe (viz kapitola 3.3 Reflexe).

Anotace mohou mít parametry s možností definování výchozí hodnoty. Typ parametrů může být libovolný primitivní typ, řetězec, výčtový typ, anotace a pole anotací. Příklad 6 ukazuje příklad deklarace anotace pro metody, která bude dostupná při běhu aplikace.

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface ViewElement {
  int cislo();
  boolean vlastnost1() default false;
  String retezec();
  TypAnotace vlastnost2() default @TypAnotace;
  TypAnotace[] pole();
}
```

Příklad 6: deklarace anotace a ukázka možných typů parametrů

Použití anotace může zjednodušit vývoj tím, že například odpadá konfigurace pomocí XML souborů. Typickým příkladem toho je JPA (Java Persistence API). JPA je standard pro objektově orietovaný přístup k databázím, využívá anotace ke konfiguraci mapování tříd na tabulky, proměnných na sloupce tabulek, specifikování vlastností sloupců a vztahů entit.

3.3 Reflexe

Jazyk Java nabízí možnost přístupu k všem vlastnostem deklarovaných tříd a vytvořených instancí tříd pomocí reflexe (The Reflection API [13]). Je možné vylistovat proměnné, metody a konstruktory tříd; lze zjistit implementované rozhraní, třídy od kterých daná třída dědí; je možné zjistit a nastavit hodnoty proměnných, volat metody; vytvářet instance pomocí konstruktorů; zjistit metainformace (anotace). Dále reflexe umožňuje pracovat s polem (`array`) a výčtovými typy (`enum`).

Tyto možnosti jsou nejčastěji využity pro práci s předem neznámými třídami (s libovolnýma) například pro psaní testů, v GUI editorech (kdy editor musí zjistit vlastnosti tříd, a ty nabízet k nastavování). Většinou nejsou součástí uživatelského kódu, ale knihoven.

Možnost čtení a nastavování hodnoty proměnných a volání metod je řízen stejným způsobem, jako by se to provádělo bez reflexe (např. není možné nastavovat a číst privátní proměnné třídy z jiné třídy). Poté, co se získá odkaz na proměnnou nebo metody třídy, pak je možné povolit přístup a tak obejít zapouzdření tříd. Samozřejmě, pokud je přítomen Security Manager (například Java Applet nebo nepodepsaná aplikace spuštěna pomocí Java Web Start), tak přístup k proměnným a metodám může být zakázán. V případě, že není možné povolit přístup, tak povolovací metoda vyhodí výjimku, v jiném případě je možné číst a nastavovat libovolnou proměnnou třídy a volat libovolnou metodu třídy.

Použití reflexe je znázorněna na příkladu 7 – pomocí ní se nastaví hodnota proměnné třídy a zavolá ty metody, u kterých je specifikována anotace `@Test`.

```java
//deklarace anotace
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.METHOD)
public @interface Test {}

//deklarace testovací třídy
public class TestClass {
  private String value;

  private void test1(String x, String y) {}
  @Test
  private void test2(String x, String y) {}
  private void test3(String x, String y) {}
}

public void test() throws Exception {
  TestClass obj = new TestClass();
  //získání proměnné třídy
  Field valueField = obj.getClass().getDeclaredField("value");
  //povolení přístupu k meotdě i přes zapouzdření
  valueField.setAccessible(true);
  //nastavení hodnoty proměnné
  valueField.set(obj, "Nová hodnota");

  //zavolání všech metod, které mají deklarovanou anotaci @Test
  for (Method m : obj.getClass().getDeclaredMethods()) {
    if (m.getAnnotation(Test.class) != null) {
      m.setAccessible(true);
      m.invoke(obj, "hodnota X", "hodnota Y");
    }
  }
}
```

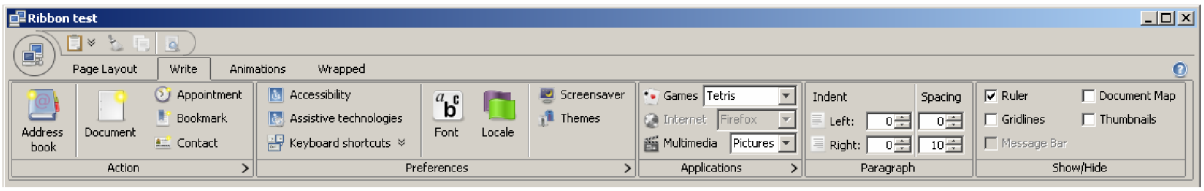
Příklad 7: ukázka použití reflexe – nastavení hodnoty proměnné a zavolání metody

4 Použité komponenty

Framework využívá volně dostupné open-source komponenty pro zobrazení pásu karet (Flamingo) a zaneprázdněný stav komponentů (JBusyComponent). Dále využívá specifikaci OSGi a její open-source implementace pro dynamické načtení modulů. Využívá též komerční produkt JIDE Docking Framework, který přidává možnost přeuspořádat uživatelské rozhraní (tzv. dokovat panely do jiných panelů).

4.1 Flamingo

Flamingo je Swing komponenta, která zobrazuje pás karet[10] (Ribbon) známé z Office 2007. Pás karet je nový koncept ovládání aplikace vyvinutou firmou Microsoft Corporation. Nahrazuje menu, toolbar a některé části dialogů. Jejím cílem bylo zvýšit použitelnost aplikací tím, že všechny důležité ovládací prvky jsou na jednom místě. Komponenta je znázorněna na obrázku 2.

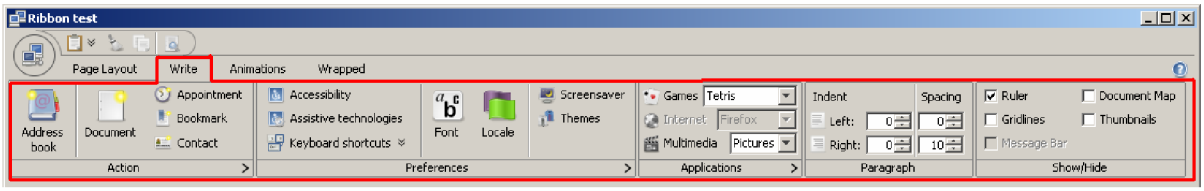


Zdroj: ukázková aplikace z distribuce Flamingo

Obrázek 2: ukázka pásu karet pomocí komponenty Flamingo

Flamingo vyvíjí Kirill Grouchnikov jako open-source projekt. Projekt je hostován na portálu dev.java.net (viz [12]). Informace o použití a možnostech komponenty lze získat z distribuované verze 3.1 a také z blogu tvůrce [11].

Ribbon se skládá z karet (Ribbon task), která je vlastně jedna stránka s ovládacími prvky – obsahuje ovládací prvky patřící k sobě (ukázka je znázorněna na obrázku 3). Výběrem karty se zobrazí jiná stránka ovládacích prvků. Karty jsou zobrazeny podobně jako v případě komponenty `JTabbedPane`[11], která umožňuje stránkování ovládacích prvků např. v dialozích. Přepínat karty je možné pomocí kliknutí na hlavičce karty, nebo programově.

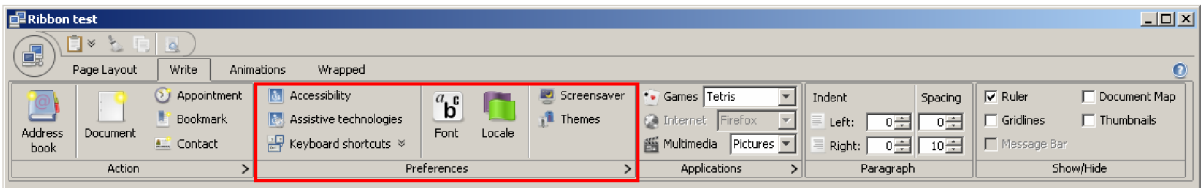


Zdroj: ukázková aplikace z distribuce Flamingo

Obrázek 3: karta pásu karet

Pás karet může obsahovat kontextové karty, které se zobrazí v závislosti na stavu aplikace. Tyto kontextové karty je možné seskupovat do skupin.

[10] české výrazy pás karet (Ribbon), karta (Ribbon task), skupina (Ribbon band) jsou převzaty z české verze online nápovědy Office 2007: http://office.microsoft.com/cs-cz/access/HA102114151029.aspx
[11] viz http://java.sun.com/docs/books/tutorial/uiswing/components/tabbedpane.html



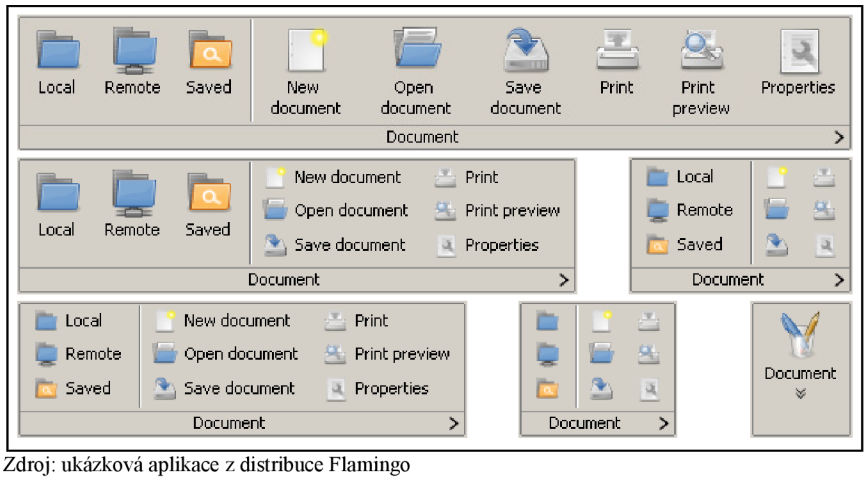
Zdroj: ukázková aplikace z distribuce Flamingo

Obrázek 4: skupina v kartě

Jednotlivé karty obsahují skupiny (Ribbon band) ovládacích prvků (ukázka je na obrázku 4). Ovládací prvky jsou speciální tlačítka (Command button), galerie tlačítek (In-ribbon galery) a také libovolné Swing komponenty. Dostupný prostor pro ovládací prvky je rozdělen podle priority ovládacích prvků. Komponenta se snaží zobrazit ovládací prvky tak, aby se vešli do šířky okna. Tlačítka jako ovládací prvky mohou být různých typů: s velkou nebo malou ikonou, s textem nebo bez, mohou obsahovat šipku pro zobrazení dalších možností, atd. V případě potřeby (když šířka okna není dostatečná) může komponenta skrýt text tlačítek, nebo změnit velké ikony na malé a zobrazit tři nad sebou v sloupci. Příkladem dynamické změny velikosti a typu ovládacích prvků podle šířky okna je možné názorně vidět z obrázku 5 (první 3 tlačítka mají vyšší prioritu, než zbylých 6 tlačítek).

Galerie zobrazují větší množství tlačítek na malém místě. Je možné rolovat zobrazené tlačítka v galerii a je možné po kliknutí na příslušné tlačítko zobrazit tlačítka ve vyskakovacím okně, která zobrazí více tlačítek najednou. Je možné ji použít například na výběr z více prvků.

Jednotlivé skupiny ovládacích prvků mohou být dále seskupovány, skupiny jsou vizuálně rozděleny pomocí svislé čáry. Skupiny mohou mít extra tlačítko se šipkou v pravém dolním rohu, která může sloužit na zobrazení dalšího menu nebo dialogu.



Zdroj: ukázková aplikace z distribuce Flamingo

Obrázek 5: skupina tlačítek na pásu karet při různých šířkách okna

Poslední verze komponenty přinesla možnost zobrazení systémového menu známé z Office 2007. Umožňuje vkládat tlačítka, které se zobrazí nad titulkama karet. Po kliknutí na hlavní systémovém tlačítku se zobrazí systémové menu aplikace.

Pomocí dvojkliku na titulku karet je možné komponentu minimalizovat, v tom případě jsou vidět jen titulky karet. Karty se zobrazí po najetí myší na řádek s titulkama karet. Tato funkce může být užitečná, když potřebujeme co nejvíce místa v hlavní pracovní části okna.

4.2 JIDE

JIDE Software[12] nabízí sadu Swing komponent a utilit pro vytváření GUI aplikací. Základní sada komponent (JIDE Common Layer [14]) je zdarma a je vyvíjen jako open-source projekt na java.net. Ostatní nabízené produkty jsou placené.

Sada open-source komponent (JIDE Common Layer[13]) obsahuje komponenty, které rozšiřují základní Swing komponenty. Jako příklad uvedeme komponentu pro rychlé vyhledávání v tabulce, seznamu nebo stromu způsobem, který je známi z Mozilla Firefox[14] – při vyhledávání se zobrazí na spodní části komponenty malý panel a během psaní vyhledávaného výrazu se první nalezený zobrazí. Dále zajímavá je možnost umístit libovolnou komponentu na jinou – např. zobrazit malou ikonku na editovacím políčku podle toho, zda je správně vyplněn, nebo zobrazit animovanou ikonu, případně progress bar na komponentě na dobu, kdy se obsah načítá. Obsahuje také stylované popisky a tlačítka, rozmístnění dialogů, atd.

Produkt s názvem JIDE Grids[15] se kolekce komponent z větší části založených na standardní tabulce (`JTable`). Obsahuje různě varianty tabulek: s možnosti automatického řazení po kliknutí na hlavičce sloupců, možnost filtrování řádků v tabulce (jen při zobrazení, model je beze změny), s možností spojených buněk, s jednoduchým nastavením stylu každé buňky zvlášť, hierarchické tabulky. Dále komponenty pro výběr barvy nebo datumu.

JIDE Software nabízí také framework pro tvorbu GUI aplikací s názvem JIDE Desktop Application Framework[16]. Je zaměřen hlavně na přenositelnost aplikace mezi platformami. Nabízí různé funkce pro jednotný přístup k souborům, obvyklé chování a výzor menu, oken, dialogů. Je určen pro středně velké projekty, pro které jsou komplexní RCP příliš složité a velké. Aplikace je celek, není složen z modulů. Plně spolupracuje s ostatními JIDE produkty.

4.2.1 JIDE Docking Framework

Jde o produkt, který spravuje umístnění a zobrazení panelů aplikace. Složitější aplikace mají kromě hlavního panelu (kde nejčastěji zobrazují otevřené dokumenty se kterým se pracuje) větší množství jiných panelů, které třeba nějakým způsobem rozmístit a řídit jejich zobrazení. Jedno možné řešení je okno aplikace rozdělit na části a v nich zobrazit další panely (v tomto případě může mít uživatel většinou možnost měnit poměr velikostí jednotlivých částí okna). Elegantnější řešení je, když uživatel může libovolně přemísťovat panely, zobrazit jen potřebné panely. Právě to řeší produkt JIDE Docking Framework [15] [16].

Aplikace je složená z hlavního panelu (není povinný) a z ostatních panelů. Správce rozmístnění (třída `DefaultDockingManager`) spravuje rozmístnění a dokování panelů. Dokování umožňuje umístit panely do 4 stran hlavního panelu nebo libovolného jiného panelu a to i vnořeno (např. panel A je umístněn v levé části hlavního panelu, a panel B v spodní části panelu A). Uživatel aplikace může přemístit panel tím, že ji přetáhne na jiné místo. Když se při přemístění přiblíží k okraji jiného panelu, tak se zobrazí možnost přichytit na danou stranu toho panelu. Panely je možné vytáhnout z hlavního okna a tím se stanou plovoucím (floating), tj. bude zobrazen v samostatném okně. Dále framework umožňuje umístit více panelů na jedno místo (uživatel musí přemístit panel na titulek jiného panelu). V tomto případě je možné přepínat mezi těmito panely. Další možností je

[12] http://www.jidesoft.com/
[13] http://www.jidesoft.com/products/oss.htm
[14] http://firefox.czilla.cz/
[15] http://www.jidesoft.com/products/grids.htm
[16] http://www.jidesoft.com/jdaf/index.htm

minimalizovat panel pomocí tlačítka v titulku panelu. Název minimalizovaných panelů se zobrazí v tenkém pruhu s možností aktivace minimalizovaných panelů (po najetí myší se panel vysune). Panely je možné také maximalizovat.

Může nastat situace, když je v různých chvílích potřebné zobrazit různé sady panelů (např. program pracuje s více typy dokumentů a každý má svoje vlastní panely). Framework nabízí na tuto situaci elegantní způsob: je možné deaktivovat jednotlivé panely podle potřeby. Deaktivovaný panel si pamatuje svůj stav, velikost a umístnění a po opětovné aktivaci se panel plně obnoví do původního stavu.

Stav a rozmístnění panelů je možné uložit a načíst ze souboru (ve Windows také z registry) nebo z libovolného místa (např. z databáze, ze serveru). Je možné také za běhu měnit rozložení a mít tak více rozložení.

Výzor a chování dokovatelných panelů je možné měnit pomocí stylů, které rozšiřují nastavený look-and-feel. Je možné nastavit styl imitující Visual Studio, Office, Eclipse a styl Xerto. Pro bezchybné zobrazení dokovaných panelů je třeba nastavit styl zavoláním funkce `LookAndFeelFactory.installJideExtension` po spuštění programu i po každé změně look-and-feel.

4.3 JBusyComponent

Komponenta, která umožňuje zobrazit libovolnou jinou komponentu tak, že je zaneprázdněný [17]. Jde o open-source komponentu, která je hostovaná na Google Code[17].

Zaneprázdněnost komponenty určuje stav modelu. V případě, že je komponenta zaneprázdněna, tak se zobrazí animovaná ikona uprostřed komponenty; komponenta je blokovaná pro interakci s uživatelem; je překryt napůl průhledným panelem, tím i vizuálně blokován; a změní se kurzor. Model může definovat, zda je možné akci zrušit či nikoliv (v případě, že je, tak se zobrazí tlačítko na zastavení akce). Je možné spojit model s běžícím úkolem a zobrazit také průběh probíhající akce.

Komponenta velmi jednoducho a elegantně řeší zobrazení zaneprázdněnosti komponenty (např. dlouhotrvající výpočet, získání dat z databáze, komunikace se serverem, atd.). Nevýhodou komponenty je její závislost na dvou knihovnách (SwingX[18] a JXLayer[19]) o celkové velikosti kolem 2MB.

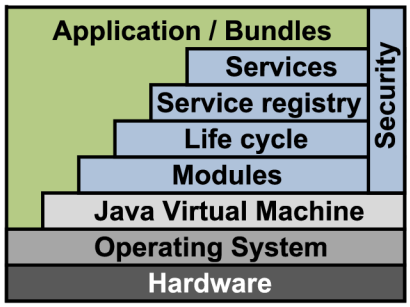
4.4 OSGi

OSGi (Open Service Gateway initiative) je specifikace pro vytváření, nasazení a běh modulárních Java aplikací ([8] a [7]). Specifikaci vyvíjí OSGi Aliance, kterým členem jsou velké firmy zabývající se informatikou jako např. IBM, Sun Microsystems, Ericsson a další. Dnes již existuje čtvrtá verze specifikace.

Nejdůležitější částí OSGi specifikace jsou: definice způsobu správy životního cyklu modulů, registrace a zpřístupnění služeb a běhové prostředí (OSGi kontejner). Dále specifikace zavádí sadu

[17] http://code.google.com/intl/cs/
[18] https://swingx.dev.java.net/ – sada open-source Swing komponent
[19] https://jxlayer.dev.java.net/ – komponenty pro dekoraci Swing komponent – zobrazení jiných komponent nad komponentami

standardních služeb a modulů. OSGi kontejner je vlastně nadstavba nad JVM, její architektura je znázorněna na obrázku 6.



Zdroj: převzato z http://en.wikipedia.org/wiki/OSGi

Obrázek 6: architektura OSGi

Bundly (OSGi používá pojem Bundle pro modul) jsou standardní JAR soubory se speciálními metainformacemi v manifest souboru (ukázku viz příklad 8). Bundly definují závislosti na ostatních modulech (požadované balíčky, které daný modul využívá) a mohou definovat balíčky, které nabízejí ostatním modulům (exportované balíčky). OSGi kontejner má na starosti mimo jiné i nalezení a vyřešení závislostí mezi jednotlivé moduly. Dále je zajištěno to, že třídy z jiných než exportovaných balíčků jsou pro ostatní moduly zcela nedostupné.

```
Manifest-Version: 1.0
Bundle-Name: Hello World
Bundle-SymbolicName: cz.vutbr.fit.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: cz.vutbr.fit.helloworld.Activator
Export-Package: cz.vutbr.fit.helloworld.export
Import-Package: org.osgi.framework
```

Příklad 8: ukázka manifest souboru pro OSGi bundle

OSGi kontejner umožňuje spustit a zastavit moduly za běhu aplikace. Tím umožňuje například načíst moduly až v momentě, když jsou zapotřebí (např. načtou se ty moduly, ke kterým má přihlášený uživatel právo) nebo aktualizovat moduly za běhu. Moduly je možné spouštět a zastavovat pomocí příslušného modulu vzdáleně.

V dnešní době existuje několik implementací OSGi specifikace, z kterých nejznámější jsou: Equinox OSGi [9] – je referenční implementace, Apache Felix[20] a Knoplerfish[21]. Všechny 3 jsou open-source implementace. Aplikace napsaná podle OSGi specifikace funguje ve všech implementacích.

Bundle může definovat třídu, která se stará o životní cyklus modulu (řádek Bundle-Activator v manifest souboru). Třída musí implementovat rozhraní `BundleActivator`, která definuje 2 metody: jedna se volá při spuštění modulu – slouží pro inicializaci modulu a registraci nabízených služeb; druhá pro ukončení modulu – pro odregistrování služeb a případně jiné operace před ukončením modulu.

Služba je ve skutečnosti Java rozhraní, implementace služby je implementace rozhraní služby. Modul registruje implementaci služby nejčastěji po nastartování. Při registraci je možné definovat dodatečné parametry, podle kterých je možné nalézt tu implementaci, kterou potřebujeme (implementací může být více). Služby se registrují, identifikují a hledají podle názvu rozhraní služby.

[20] http://felix.apache.org/site/index.html
[21] http://www.knopflerfish.org/

Modul, který chce nějakou službu použít ji musí vyžádat od kontextu (`BundleContext` – bundle ji dostane při spuštění). V případě úspěchu dostane instanci, kterou používá. Přidělenou instanci třeba vrátit zpátky poté co už není potřebné (případně při ukončení), aby nebyl bloková modul ze které je služba implementovaná (jinak by se modul implementace služby nedal zastavit).

V kontextu je možné zaregistrovat posluchač, který je informován o spuštění nebo zastavení libovolného bundlu.

Podle specifikace, OSGi používá cache pro nainstalované bundle. V cache je uložena spuštěna verze všech bundlů (verzí může být více najednou). Cache řeší všechny implementace jinak. Z toho vyplívá, že aplikace, která využívá OSGi musí mít právo pro přístup k systému souborů.

Zastavit OSGi kontejner je možné tak, že zastavíme systémový modul (s indexem 0). Od kontextu je možné získat odkaz na jednotlivé bundle podle indexu a pomocí toho odkazu je možné moduly zastavovat.

5 Implementace

Náš framework je vyvíjen v prostření Netbeans IDE pod názvem MSFX (Module Suite Framework X). Byl navržen pro potřeby plánovaného rozsáhlého komerčního produktu. Byl kladen důraz na to, že aplikace se bude instalovat na cílové počítače, nebo se bude spouštět přes technologii Java Web Start[22]. Dalším důležitým požadavkem bylo použití moderní a intuitivní GUI, které zahrnuje použití pásu karet ale také pokročilých GUI komponent. Flamingo byla jediná dostupná komponentou pro zobrazení pásu karet pro Javu. A komponenty JIDE byli vybrané proto, že jsme s nimi měli delší zkušenosti a jsme s nimi spokojní.

Ze výše uvedených implementací OSGi jsem vybral Apache Felix, protože tato umožňuje jednoduchým způsobem spustit celou aplikaci, jako samostatný program. Navíc tím, že se OSGi framework spouští programově, je možné provést konfiguraci OSGi frameworku za běhu (a není potřeba konfigurovat externím souborem).

Jednotlivé moduly definují svoje pohledy (View) a ovládací prvky do pásu karet pomocí anotací, které jsou při spuštění modulu automaticky zpracovány. Jednotlivé pohledy jsou složeny z panelů (ViewElement), které je možné zvlášť posouvat a přemísťovat pomocí JIDE Docking Framework. Tyto panely jsou rozmístněny v hlavní části okna aplikace.

Framework obsahuje v sobě málo změněnou verzi knihovny Swing Application Framework. Změna byla nutná z toho důvodu, že při používání modulů má existovat jen jediná instance třídy `Application`, která je vytvořena při inicializaci MSFX frameworku.

Cílem MSFX bylo mimo jiné to, aby moduly byli na sebe nezávislé – aby bylo možné nasazovat, testovat a hlavně vyvíjet moduly samostatně. Moduly spolu komunikují prostřednictvím služeb – moduly mohou registrovat služby, vyhledávat je a používat. Služby jsou Java rozhraní – při registraci se registruje implementace toho rozhraní. MSFX nabízí základní služby modulům, jako například získání instance hlavního okna, získání správce zdrojů/modulů/akcí/služeb, atd. (viz kapitola 5.6 Služby nabízené MSFX).

Panely v pohledech jsou vytvořeny frameworkem až při prvním použití (tj. poprvé, když je pás karet přepnut na kartu, ke které je pohled přiřazen). Tím se urychlí spuštění modulů a ušetří paměť. Dále v případě, že pohledy zobrazují data z databáze, tak se komunikace s databází odloží k prvnímu zobrazení.

Zaregistrované služby může MSFX injektovat do modulu a panelů automaticky při inicializaci modulu/panelu. Injektují se také zdroje z resource souborů – používá SAF k získání zdrojů. Injekce se provádí pro ty proměnné třídy modulu/panelu, které mají příslušné anotace. Injektovat je možné do libovolného objektu (tj. do takových, které nebyli vytvořeny frameworkem) pomocí služby `Injektor` (podrobný popis viz kapitola 5.5 Injekce zdrojů).

5.1 Spuštění a inicializace aplikace

Framework je možné spustit dvěma způsoby: se statickou nebo dynamickou správou modulů. Po spuštění se již postará o spuštění GUI a modulů. Pro moduly je způsob spuštění zcela transparentní.

Při statické inicializaci všechny moduly jsou součástí programu, nepoužívá se OSGi na načtení modulů. Tento způsob je třeba použít při vývoji a ladění aplikace. V případě použití OSGi není

[22] http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp

možné program ladit, protože OSGi používá cache – ukládá načtené moduly do určeného adresáře a spouští je z odtamtud.

Dynamické spuštění frameworku zajišťuje OSGi, která ji načte také jako modul (bundle). Všechny moduly mají deklarovanou závislost na frameworku, proto se načte a spustí ještě před samotnými moduly. Framework OSGi je spuštěn programově – umožňuje se tím provést potřebné úkoly přes spuštěním OSGi (teda i MSFX).

Před spuštěním MSFX (nebo OSGi v případě dynamické správy modulů) je třeba nastavit prostředí: nastavit look-and-feel pro Swing komponenty, inicializovat logování, nastavit default Locale podle jazyka aplikace, atd.

5.1.1 StaticModuleManager – statické moduly

V tomto případě se specifikuje při vytváření správce modulů seznam všech modulů (seznam tříd modulů) aplikace. Všechny moduly musí být dostupné z classpath. Příklad spuštění aplikace se dvěmi moduly je vidět na příkladu 9.

```
public static void main(String[] args) {
  LookAndFeelFactory.installJideExtension(LookAndFeelFactory.XERTO_STYLE_WITHOUT_MENU);

  StaticModuleManager moduleManager =
                new StaticModuleManager(Module1.class, Module2.class);
  moduleManager.startApplication(args);
}
```

Příklad 9: spuštění MSFX se statickou správou modulů

Správce modulů v tomto případě zinicializuje všechny moduly a následně zobrazí hlavní okno aplikace.

5.1.2 DynamicModuleManager – moduly načtené dynamicky

MSFX je z pohledu OSGi jeden z bundlů. Spustí se ještě před MSFX moduly, protože všechny moduly jsou na něm závislé. K spuštění aplikace je třeba spustit OSGi kontejner. Z dostupných implementací jsem si vybral Apache Felix z toho důvodu, že má zveřejněn způsob inicializace a spuštění OSGi frameworku programově (viz [19]).

Moduly je možné spustit při startu OSGi kontejneru, nebo je může spouštět libovolný bundle pomocí nástrojů, které nabízí OSGi. Způsob spuštění modulů při startu OSGi kontejneru je možné řešit pomocí konfiguračního souboru, nebo přímo specifikovat URL spouštěných modulů při inicializaci OSGi. Konfigurace pro Apache Felix se zadává jako `Map` s textovým klíčem. Spuštění OSGi kontejneru Apache Felix je znázorněn na příkladu 11. Spustí framework MSFX a modul MSFXModule. OSGi cache je uložen v podadresáři `felix-cache` v pracovním adresáři.

Příkald 10 demostruje způsob, jakým se dá instalovat a spustit OSGi bundle ze specifikovaného souboru kdykoliv za běhu aplikace (proměnná `context` je `BundleContext`, kterou bundle dostal při inicializaci).

```
try {
  Bundle bundle = context.installBundle("file:bundle/TestModule.jar");
  bundle.start();
} catch (Exception ex) {
  //zpracování výjimky
}
```

Příklad 10: instalace a spuštění OSGi bundle ze souboru

Původním záměrem bylo, že všechny knihovny, které MSFX vyžaduje budou také OSGi bundle. To ale naráží na problém, který vychází ze způsobu, jakým Swing pracuje s look-and-feel. Komponenta Flamingo, komponenty JIDE a komponenta JBusyComponent využívá vlastní rozšíření look-and-feel. Navíc v aplikaci, na podporu které původně MSFX vznikl, se plánovalo využít look-and-feel s názvem Subsatnce[23] od tvůrce komponenty Flamingo. Podstata problému je, že look-and-feel používá tabulku, ve které jsou hodnoty jednotlivých vlastností (barvy, okraje, fonty, atd.) a též názvy tříd, které vykonávají samotné vykreslení komponenty. Swing vytváří tyto třídy podle názvu třídy pomocí statické metody `Class.forName`. Problém je, že třídy které jsou v bundlech nejsou dostupné pro systémové třídy – nejsou v classpath, takže je nemůže knihovna Swing vytvořit. Řešením je, že tyto knihovny nejsou OSGi bundle, a jsou specifikované v classpath při spuštění aplikace (zavaděče, který spustí OSGi kontejner). Aby třídy ze zmíněných knihoven mohli být dostupné pro bundle je třeba je přidat buď k systémovým balíčkům (systémový bundle exportuje tyto balíčky a tím jsou dostupné pro ostatní bundle) nebo k delegovaným balíčkům (specifikované balíčky jsou vyjmuty ze správy balíčků OSGi, načtou se pomocí původního classloaderu). Vybrán byl druhý zmíněný způsob, protože umožňuje specifikovat balíčky hromadně (viz konfigurace Apache Felix, vlastnost `org.osgi.framework.bootdelegation` v příkladu 11).

```
public static void main(String[] args) throws Exception {
  LookAndFeelFactory.installJideExtension(LookAndFeelFactory.XERTO_STYLE_WITHOUT_MENU);
  //seznam modulů, které se mají spustit spolu s OSGi
  List<String> bundles = new ArrayList<String>();
  bundles.add("file:bundles/ModuleSuiteFrameworkX.jar");
  bundles.add("file:bundles/MSFXModule.jar");

  //poskládáme do jednoho řetězce
  String bundlesStr = "";
  for (String bundle : bundles) bundlesStr = bundlesStr + bundle + " ";

  //konfigurace OSGi frameworku
  Map<String, Object> config = new HashMap<String, Object>();
  config.put("org.osgi.framework.bootdelegation",
             "sun.*,com.sun.*,org.jvnet.*,com.jidesoft.*,org.divxdede.swing.busy.*");
  config.put("org.osgi.framework.storage.clean", "onFirstInit");
  config.put("felix.log.level", "1");
  config.put("felix.auto.start.1", bundlesStr);
  //přidá do konfigurace třídu aktivátoru, která spustí výše zmíněné bundly
  List activatorList = new ArrayList();
  activatorList.add(new AutoActivator(config));
  config.put(FelixConstants.SYSTEMBUNDLE_ACTIVATORS_PROP, activatorList);

  try {//Vytvoření a spuštění OSGi frameworku
    Framework m_felix = new Felix(config);
    m_felix.start();
    m_felix.waitForStop(0);
    System.exit(0);
  } catch (Exception ex) {
    //zpracování výjimky - není možné spustit OSGi kontejner
    System.exit(-1);
  }
}
```

Příklad 11: spuštění OSGi kontejneru Apache Felix a vněm MSFX s jedným modulem

5.2 MSFX moduly

MSFX moduly jsou speciální OSGi bundle, které obsahují část GUI aplikace. Tyto moduly spravuje náš framework. OSGi bundle se stává MSFX modulem když v manifest souboru je definována hlavní třída modulu (vlastnost `MSF-Module`).

[23] stránka projektu Substance: https://substance.dev.java.net/

Z třídy modulu se načtou pohledy a GUI elementy. Pokud tato třída implementuje rozhraní `ModuleLifeCycleListener`, tak je modul informován o jeho spuštění a zastavení. V metodě `started` je možné provést registraci služeb a jiné potřebné inicializační kroky.

Vytvoření MSFX modulu v Netbeans IDE

K vytvoření projektu v Netbeans IDE pro MSFX modul je třeba založit projekt typu Java Desktop Application. Založme projekt s názvem `MSFXModule` (implicitně dá soubory do balíčku `msfxmodule`). Vytvořením projektu vznikne třída pro životní cyklus SAF aplikace (`MSFXModuleApp`), vytvoří hlavní okno (`MSFXModuleView`) a okno s informacemi o programu (`MSFXModuleAboutBox`). Vytvořené okna a k nim patřící resource soubory můžeme z projektu vymazat, nebudeme je potřebovat. Z třídy aplikace třeba vymazat kód všech funkcí, aby se dala aplikace zkompilovat – nejsou využity.

K projektu třeba přidat knihovnu MSFX (`ModuleSuiteFrameworkX.jar`) a ujistit se, že v seznamu knihoven je nad knihovnou SAF (ve vlastnostech projektu, sekce libraries). Jinak nebude možné modul spustit, bude hlásit chyby konverze typů.

```
package msfxmodule;

import com.wetcom.msfx.annotation.*;
import com.wetcom.msfx.service.MainFrame;
import org.jdesktop.application.Action;

@View(id = "testView")
@Tab(id = "testTab")
@Band(id = "testBand", buttons = {
  @Button(action = "exit")
})
public class MSFXModule {
  @ViewElement(placement = WindowPlace.CENTER)
  private MainPanel mainPanel;
  @Injected
  private MainFrame mainFrame;

  @Action
  public void exit() {
    mainFrame.closeApplication();
  }
}
```

Příklad 12: minimalistický MSFX modul s jedním tlačítkem, které ukončí aplikaci

Musíme vytvořit třídu, která bude hlavní třídou modulu (vytvoříme třídu s názvem MSFXModule). Minimalistický MSFX modul musí mít jeden pohled (s minimálně jedním panelem), kartu (samozřejmě minimálně s jednou skupinou ve které je alespoň jedno tlačítko). Vytvořme panel s názvem MainPanel (New File... – Swing GUI Forms – JPanel Form) na který vložíme libovolné komponenty, který bude hlavním panelem ukázkového modulu. Specifikujeme jeden pohled, který bude obsahovat vytvořený panel. Dále jednu kartu s tlačítkem Exit, která ukončí aplikaci. Musíme vytvořit resource soubor, ve kterém bude titulek karty, skupiny a tlačítka (akce). Resource soubor podle principů SAF se musí jmenovat `MainPanel.properties` v podadresáři resources. Příklad 12 znázorňuje třídu ukázkového modulu a příklad 13 k němu patřící resource soubor.

```
testTab.Tab.title=tab title
mainPanel.ViewElement.title=view title
testBand.Band.title=band title
exit.Action.title=Exit
```

Příklad 13: resource soubor ukázkového modulu z příkladu 12

```
Manifest-Version: 1.0
Bundle-Name: Test Module
Bundle-SymbolicName: testmodule
Bundle-Version: 0.0.1
Bundle-Description: Test Module
MSF-Module: msfxmodule.MSFXModule
Import-Package: com.wetcom.msfx.service, com.wetcom.msfx.annotation,
org.jdesktop.application
DynamicImport-Package: *
```

Příklad 14: manifest soubor ukázkového MSFX modulu

Dále je třeba modifikovat manifest soubor aby projekt byl jednak OSGi bundle, a též MSFX modul. Manifest soubor je k dispozici v pohledu souborů (Files) v kořenovém adresáři projektu (soubor s názvem `manifest.mf`). Aby se OSGi bundle stal MSFX modulem, třeba specifikovat třídu modulu. Je třeba specifikovat balíčky, které modul používá z ostatních bundlů – modul minimálně používá balíčky z MSFX modulu. Příklad 14 ukazuje typický manifest soubor – konkrétně pro náš vytvořený ukázkový modul.

Po provedení těchto kroků máme hotový ukázkový modul, a je ji možné spustit jako MSFX modul.

Pokud potřebujeme spustit samostatný modul, tak musíme k projektu přidat knihovny potřebné pro běh MSFX a aplikaci spustit pomocí statické správy modulů. Potřebné jsou knihovny Flamingo (flamingo.jar), JIDE (jide-common.jar, jide-dock.jar) a JBusyComponent (swingx-0.9.5.jar, jxlayer.jar, JBusyComponent.jar). Do `main` funkce v třídě aplikace třeba doplnit kód pro spuštění MSFX se statickou správou modulů (viz příklad 9).

5.3 Použití pásu karet

Třída modulu a jednotlivé panely (`ViewElement`) definují pomocí anotací pohledy, karty, skupiny a ovládací prvky pásu karet. Pohledy, karty a skupiny se identifikují pomocí textového identifikátoru. Karty a skupiny musí mít povinně nastavený titulek, který se čte podle principů SAF z resource souboru patřící k třídě ve které jsou definované. Vlastnosti akce se nastavují standardním způsoben v resource souboru (ukázku resource souboru modulu viz příklad 15). Použití resource souborů zabezpečuje možnost lokalizace výsledné aplikace.

```
Test1Tab.Tab.title=test

FirstBand.Band.title=first
SecondBand.Band.title=second
ThirdBand.Band.title=third

panel1.ViewElement.title=panel1
leftPanel.ViewElement.title=left panel

test1Action.Action.text=Back
test1Action.Action.icon=/com/wetcom/test/gui/icons/back.png
```

Příklad 15: ukázka použití pásu karet – resource soubor k příkladu 16

Každý modul může mít definován jeden pohled (pomocí anotace `@View`). Pohled obsahuje sadu panelů a jejich základní rozmístnění. Panely jsou definované jako proměnné třídy modulu s anotací `@ViewElement`. Každý panel patří právě k jednomu pohledu, který je možné určit explicitně, nebo je použit pohled definovaný v modulu. Výchozí pozice panelu, která určuje místo dokování může být (výčtový typ `WindowPlace`):

- `CENTER` – panel se umístí do hlavního pracovního místa okna, tento panel není možné přemístit a ostatní panely jsou dokovány do tohoto panelu
- `LEFT` – dokován vlevo
- `RIGHT` – dokován vpravo
- `TOP` – dokován nahoře
- `BOTTOM` – dokován dolů
- `HIDDEN` – výchozí stav je, že panel není vidět, ale programově je možné ji zobrazit a pak s ním pracovat jako s ostatními panely, tj. dokovat a přemísťovat
- `NONE` – nevytváří se panel, který je možné dokovat, jen se provede načtení ovládacích prvků z třídy a injekce zdrojů (při prvním zobrazení pohledu do které patří)

Třída modulu a panely mohou definovat karty (jen moduly), skupiny a ovládací prvky, které se vloží do pásu karet. Každá anotace má parametr pro textovou identifikace (`id`) a pro určení jejich pořadí v rámci nadřízené komponenty (`index`). Pořadí se určí pomocí zadané celočíselné hodnoty pokud je specifikován, nebo podle pořadí definice/zpracování.

Každá karta má jednoznačně přirazený pohled, který se zobrazí, když se daná karta na pásu karet vybere. Tento pohled se určí buď pomocí parametru anotace, nebo se přiřadí pohled, který je definovaný v modulu. Třída modulu definuje kartu pomocí anotace `@Tab` v případě, že definuje jedinou kartu, nebo pomocí anotace `@Tabs`, která v sobě obsahuje definici více karet.

Skupiny se deklarují pomocí anotace `@Band` buď jako seznam v anotaci karty, nebo zvlášť (když se definuje pouze jedna skupina). Když se definují mimo anotace karty, tak je možné pomocí anotace `@Bands` definovat více skupina najednou. Skupina bude přirazena buď k specifikované kartě, nebo v případě, že je definovaná v rámci anotace karty, tak k té kartě, případně k jediné kartě definované v modulu. K skupině je možné přiřadit akci, která se vykoná po stisknutí malé ikonky v pravém dolním rohu skupiny (v případě, že akce není definována, tak se ikonka nezobrazí).

Ovládací prvky, konkrétně tlačítka s ikonou se definují pomocí anotace `@Button` buď jako parametr anotace skupiny, nebo mimo ní, případně v anotace `@Buttons` (více tlačítek mimo anotace skupiny). Pro přiřazení tlačítka ke skupině platí stejné pravidla, jako pro přiřazení skupin ke kartám. Text a ikona tlačítka se bere z přiřazené akce.

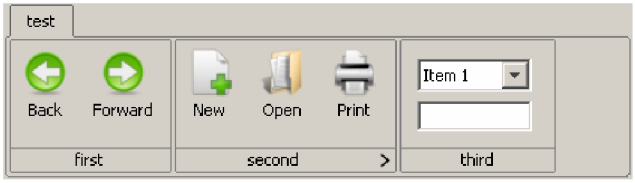
Do skupina místo tlačítek je možné vložit vlastní panel s libovolnými komponentami. MSFX zatím podporuje vložení vlastního panelu do celé skupiny (tj. nemůžou v té skupině být tlačítka). Panel do pásu karet se definuje podobným způsobem, jako panel aplikace: jako proměnná třídy s anotací `@BandElement`. Panel se vytvoří v momentě vytvoření instance třídy ve které je definován. Komponenta zobrazující pás karet respektuje preferovanou šířku vloženého panelu a výšku nastaví na výšku pásu karet.

```
@View(id = "Test1View")
@Tab(id = "Test1Tab", bands = {
  @Band(id = "FirstBand", buttons = {
    @Button(action = "test1Action"),
    @Button(action = "test2Action")
  }),
  @Band(id = "SecondBand", action="bandAction", buttons = {
    @Button(action = "test3Action"),
    @Button(action = "test4Action") ,
    @Button(action = "test5Action")
  }),
  @Band(id = "ThirdBand")
})
public class Test1Module implements ModuleLifeCycleListener {
  @BandElement(band = "ThirdBand")
  private BandPanel bandPanel;
  @ViewElement(placement = WindowPlace.CENTER)
  private Test1Panel panel1;
  @ViewElement(placement = WindowPlace.LEFT)
  private LeftPanel leftPanel;

  @Action
  public void test1Action() {
...
```

Příklad 16: ukázka použití pásu karet v MS**FX**

Ukázka definice ovládacích prvků pásu karet je znázorněn na příkladu 16, k němu patřící resource soubor na příkladu 15, Výsledný pás karet je znázorněn na obrázku 7. Ukázkový kód definuje jednu kartu, k němu přiřazenou kartu s třemi skupinami. První skupina obsahuje 2 tlačítka, druhá 3 tlačítka a ve třetí skupině je vložen vlastní panel. Druhá skupina má přiřazenou akci.



Obrázek 7: pás karet z příkladu 16

5.4 Komunikace modulů

Preferovaný způsob komunikace modulů je použitím služeb. Modul, který potřebuje nabízet své služby jiným modulům definuje rozhraní služby a zaregistruje její implementaci do správce služeb. Správce služeb lze získat v modulu automaticky pomocí injekce (viz kapitola 5.5 Injekce zdrojů), stačí deklarovat proměnnou s typem `ServiceManager` a použít anotace `@Injected`.

Je možné registrovat více implementací jedné služby (např. více modulů implementuje danou službu). Proto je možné vyhledat službu podle parametrů (jako při službách OSGi), nebo získat seznam všech implementací. Narozdíl od OSGi se v MSFX registrují a vyhledávají služby podle třídy rozhraní (a ne podle názvu rozhraní). Cílem bylo zakrýt použití OSGi, hlavně kvůli tomu, aby bylo možné aplikaci spustit i bez OSGi pomocí statické správy modulů.

Je možné zaregistrovat posluchač (třída implementující rozhraní `ServiceListener<S>`, kde S je rozhraní služby) pro službu, která je informována o tom, že byla zaregistrovaná nebo odregistrována její implementace.

5.5 Injekce zdrojů

Injekce zdrojů je užitečná vlastnost, která usnadňuje přístup k službám a jiným zdrojům. Při jejím použití nemusíme získávat instance služeb ručně, ale provede se to automaticky. K injekci je použitá reflexe, čte se seznam deklarovaných proměnných a podle přítomnosti anotace se provede nastavení hodnoty proměnné. MSFX používá dvě anotace: `@Injected` a `@Resource`. První z nich slouží pro injekci služeb a druhá pro injekci zdrojů z resource souborů (známá z SAF).

Injekce se automaticky provádí také pro panely pohledů (`@ViewElement`) a pro vlastní panely do pásu karet (`@BandElement`) do třídy modulu. Panely pohledu jsou nastaveny při prvním zobrazení pohledu ke kterému patří, a panely do pásu karet při vytváření instance třídy ve kterém jsou deklarované.

V případě, že je pro službu zaregistrováno více implementací, tak se injektuje jeden z dostupných implementací (to, že které to bude není specifikován).

Automatická injekce funguje pro třídu modulu, pro panely pohledu, pro panely do pásu karet – instance všech uvedených vytvoří framework. Může nastat ale situace, kdy budeme potřebovat injektovat služby nebo zdroje do tříd, které vytvoříme ručně. Pro tento účel je zaregistrovaná služba Injector, pomocí které je možné injekci provést do libovolné třídy.

Pokud třída, do které se injektuje, implementuje rozhraní `InjectionListener`, tak je instance třídy informována o dokončení injekce (zavoláním metody `injectionComplete`). Tuto metodu je možné použít pro provedení inicializace, která využívá již nainjektovaných služeb a zdrojů.

MSFX garantuje přítomnost implementací základních služeb, kterých výčet je možné nalézt v následující kapitole.

5.6 Služby nabízené MSFX

Služba z pohledu frameworku je každé Java rozhraní. MSFX dovoluje zaregistrovat jednu nebo více implementací pro jednu službu. Služby se registrují pomocí správce služeb (služba ServiceManager). Vyhledávají se pomocí správce služeb nebo použitím injekce (viz kapitola 5.5 Injekce zdrojů). Změny služeb je možné sledovat pomocí posluchače, která je informována o každé změně (registrace a odregistrování implementace služby).

Základní sada služeb je dostupný všem modulům. Jejich implementace je součástí frameworku. Tyto služby jsou:

- `ServiceManager` – správce služeb
- `Injector` – provádí ruční injekci služeb a zdrojů
- `MainFrame` – přístup k funkcím hlavního okna, manipulace s pásem karet, pohledů a panelů, zpracování chyb, ukončení aplikace
- `ModuleManager` – správce modulů (zatím neimplementováno)
- `ActionManager` – správce SAF akcí
- `ResourceManager` – správce zdrojů z resource souborů

5.6.1 ServiceManager

Slouží pro registraci a získání instancí služeb. Podporuje 2 způsoby registrace služeb. První z nich předpokládá existenci jediné implementace (`registerService`) a ta druhá umožňuje registraci více implementací (`registerMultiService`). V prvním případě nově zaregistrovaná

implementace nahradí všechny již zaregistrované implementace – je možné získat jen posledně přidanou implementaci. Druhý způsob přidává další implementace služby již k zaregistrovaným. Při registrování implementace služby je možné specifikovat parametry. Parametry se zadávají jako instance tříd `Property` (obsahuje textový klíč a libovolnou hodnotu).

Získat je možné jednu implementaci služby (`getService`), nebo všechny implementace najednou (`getMultiService`).

Implementace je možné filtrovat zadáním parametrů. Pokud při získání služby jsou nějaké parametry zadané, tak všechny musí existovat a souhlasit s parametry zadané při registraci služby. Parametry pro instanci vybrané implementace je možné získat jako kolekce vlastností (`getProperties`). Možný příklad použití parametrů: mějme službu, která provádí kontrolu pravopisu s více implementacemi; jednotlivé implementace se liší v jazyce; pomocí parametru specifikují jazyk – je možné získat implementaci pro daný jazyk, nebo je možné vylistovat dostupné jazyky.

K jednotlivým službám je možné zaregistrovat posluchač, který je informován o změnách v registraci služby (`addServiceListener`). Posluchač je libovolná třída implementující rozhraní ServiceListener<S> (S je rozhraní služby) s jednou metodou `serviceChanged`.

```
public class TestModule implements ModuleLifeCycleListener {
  @Injected
  private ServiceManager serviceManager;

  public void started() {
    //registrace jednoduché služby
    serviceManager.registerService(SimpleService.class, new SimpleServiceImpl());

    //registrace více implementací služby LangService
    serviceManager.registerMultiService(LangService.class, new CzechLangImpl(),
                new Property("Language", "Czech"));
    serviceManager.registerMultiService(LangService.class, new EnglishLangImpl(),
                new Property("Language", "English"));
  }

  @Action
  public void useServicesAction() {
    //použití jednoduché služby
    SimpleService serv = serviceManager.getService(SimpleService.class);
    serv.helloWorld();

    //vylistování seznamu jazyků implementací LangService
    List<LangService> langs = serviceManager.getMultiService(LangService.class);
    for (LangService lang : langs) {
      for (Property p : serviceManager.getProperties(lang)) {
        if (p.getKey().equals("Language")) System.out.println(p.getValue());
      }
    }

    //získání české implementace LangService
    LangService czech = serviceManager.getService(LangService.class,
                new Property("Language", "Czech"));
    czech.use();
  }
```

Příklad 17: použití správce služeb v MSFX: registrace a použití jednoduché i složitější služby

Příklad 17 demonstruje jednoduché použití služby (bez parametrů, jediná implementace), registrace více implementací jazykové služby s parametrem určující jazyk, vylistování jazyků všech implementací a získání jedné konkrétní implementace.

5.6.2 Injector

Pomocí této služby je možné provádět ruční injekci služeb a zdrojů (metoda `injectObject`). Injektují se služby do proměnných, které mají anotace `@Injected` a zdroje z resource souborů do proměnných, které mají anotaci `@Resource`. Služba provádí injekci pomocí reflexe.

```
public class TestModule implements ModuleLifeCycleListener {
  @Injected
  private Injector injector;

  public void started() {
    InjectionTest test = new InjectionTest();
    injector.injectObject(test);
  }
}
```

Příklad 18: ukázka služby Injector – ruční injekce do třídy z příkladu 19

Při injektování služby je možné zadat parametry do anotace pomocí pole anotací `@Property`, které určují textový klíč a textovou hodnotu (není možné filtrovat implementace podle libovolného objektu). Pokud existuje více vyhovujících implementací injektované služby, tak není definováno, že které z implementací se použije.

Při injekci zdrojů z resource souborů je možné specifikovat parametr `key` u anotace `@Resource`, která určuje, že která hodnota se načte z resource souboru. V případě, že není specifikován, tak se použije implicitní název (jak v SAF, protože injekce zdrojů je prováděno pomocí SAF), která se skládá z krátkého názvu třídy do které se injektuje a z názvu injektované proměnné oddělenou tečkou.

Pokud třída, do které se injektuje implementuje rozhraní `InjectionListener`, tak je informován o dokončení injekce voláním metody `injectionComplete`. Tuto metodu lze použít pro inicializaci instance – inicializace tady a ne v konstruktoru. Metoda provádějící injekci má volitelný parametr, pomocí kterého je možné zakázat informování objektu o dokončení injekce.

Uvedeme příklad, který demonstruje všechny zmíněné možnosti injekce. Příklad 18 ukazuje použití ruční injekce z modulu; příklad 19 ukazuje třídu, do které se injektuje (resource soubor, ze kterého se zdroje čtou je v podadresáři resources s názvem `InjectionTest.properties`).

```
public class InjectionTest implements InjectionListener {
  @Injected
  private ServiceManager serviceManager;
  //vyhledání služby podle 1 parametru
  @Injected(@Property(key = "Language", value = "Czech"))
  private LangService czech;
  //vyhledání podle více parametrů
  @Injected({@Property(key = "A", value = "v1"), @Property(key = "B", value = "v2")})
  private SimpleService test;
  //načte hodnotu klíče InjectionTest.test1;
  @Resource
  private String test1;
  @Resource(key="libovolny.zadany.klic")
  private String test2;

  public void injectionComplete() {
    //zavolán, když injekce je dokončena
  }
}
```

Příklad 19: třída ve které se provede injekce zdrojů i služeb

5.6.3 MainFrame

Služba poskytuje přístup k hlavnímu oknu aplikace, funkce pro manipulaci s pásem karet, pohledů a panelů aplikace a funkce pro zpracování výjimek.

Zpracování chyb/výjimek

Služba nabízí elegantní způsob pro zobrazení chybových hlášek uživateli, které nastanou v modulech. Pomocí metody `exception` je možné zobrazit dialogové okno, ve kterém se zobrazí chybová hláška z poskytnuté výjimky. V dialogovém okně bude navíc tlačítko pomocí které je možné zobrazit detaily chyby – po kliknutí se zobrazí stack trace výjimky. Tuto funkci volá posluchač, který zachytává výjimky, které nebyli v kódu zachyceny (které nastanou např. při provádění kódu ve vlákně EDT).

Ovládání hlavního okna aplikace

Pomocí metod `setSize` a `setVisible` je možné nastavit velikost hlavního okna a zobrazit/skrýt ji. Dále je možné získat instanci `JFrame` aplikace, která představuje hlavní okno aplikace (`getFrame`).

Pomocí metody `addWindowListener`[24] je možné přidat posluchač, který bude informován o změně stavu hlavního okna (tj. zobrazení, skrytí, minimalizace, obnovení, získání/ztráta fokusu).

Pomocí metody `parseWindowElements` je možné ručně spustit načítání ovládacích prvků a pohledů ze specifikovaného objektu. Na daném objektu proběhne proces, který probíhá při inicializaci modulu, tj. načtení ovládacích prvků pásu karet, načtení panelů aplikace a provede se injekce služeb a zdrojů do objektu. Umožňuje vytvořit další pohledy v aplikaci, nebo umožňuje podmíněné zobrazení pohledu a součástí pásu karet, případně umožňuje přidávat více instancí jednoho panelu pro různé účely.

Aplikaci je možné správně ukončit pomocí volání metody `closeApplication`. Voláním se simuluje zatvoření hlavního okna aplikace. Tuto funkci je třeba volat, pokud akce potřebuje ukončit aplikaci – například: tlačítko Exit na pásu karet, nebo automatické ukončení aplikace po uplynutí doby.

Ovládání panelů aplikace

Panel aplikace je interně reprezentována třídou `ViewElement`, kterou je třeba použít pro manipulaci s panely. Instance třídy jednoznačně identifikuje panel ve frameworku. Vyhledat je možné podle textového identifikátoru nebo podle instance panelu (`getViewElement` podle typu parametru), případně je možné získat seznam všech panelů v aplikaci (`getViewElements`). Po nalezení reprezentace panelu je možné panel zobrazit nebo skrýt (`hideViewElement`, `showViewElement`).

Protože každý panel je vložen do komponenty `JBusyComponent` je možné zapnout/vypnout zaneprázdněný stav pomocí metody `setBusy`. Pokud je panel v zaneprázdněném stavu tak uživatel nemůže pracovat s panelem a je vizuálně zobrazeno, že aplikace pracuje. Lze použít například během načítání dat ze souboru/databáze/serveru.

Ovládání pohledů

Pohled je reprezentován instancí třídy `View`, která jednoznačně identifikuje pohled. Pohled je možné vyhledat podle textového identifikátoru (`getView`). Pomocí nalezené identifikační instance je možné přepnout na daný pohled (`showView`). Pohled se aktivuje bez toho, aby se měnila vybraná

[24] viz http://java.sun.com/j2se/1.4.2/docs/api/java/awt/event/WindowListener.html

karta v pásu karet. V případě, že pohled zatím nebyla zobrazena, provede se inicializace pohledu a vytvoření instancí příslušných panelů.

Ovládání pásu karet

Jednotlivé karty pásu karet identifikuje instance třídy `Tab`. Kartu je možné vyhledat podle textového identifikátoru (`getTab`). Pomocí metody `selectTab` se aktivuje vybraná karta, stejným způsobem jako by uživatel kliknul na danou kartu. Přepne se na pohled, který je přiřazený k dané kartě. Tato funkcionalita umožňuje v případě potřeby programově přepínat kartu (a s ní i pohled).

Instance třídy `Band` identifikuje skupinu na kartě. Je možné ji nalézt pomocí textového identifikátoru (`getBand`). Následně je možné skupinu zobrazit nebo skrýt podle potřeby aplikace (`hideBand` a `showBand`).

Dále služba umožňuje manipulaci s panely umístěnými v skupinách pásu karet. Sice moduly mají k dispozici instance těch panelů, ale nastavením jejich viditelnosti se neovlivní zabírané místo na pásu karet. Je třeba získat instanci identifikační třídy `BandElement` buď pomocí textového identifikátoru, nebo pomocí instance panelu (`getBandElement` podle typu parametru). Je možné též získat seznam všech panelů pomocí metody `getBandElements`. Panely je možné zobrazit nebo skrýt (`hideBandElement` a `showBandElement`).

5.6.4 ModuleManager

Služba by měla nabízet možnost spuštění a zastavení modulů, zjistit informace o modulech, možnost aktualizace modulu a získání seznamu spuštěných modulů. Zatím není implementováno. Pro provedení těchto funkcí je možné použít přímo možnosti OSGi, ale toto není žádoucí, protože bude modul záviset na OSGi a tím znemožněn spustit ji pomocí statické zprávy modulů.

5.6.5 ActionManager

Slouží pro nalezení SAF akcí. Pomocí metody `getAction` lze získat instanci akce podle objektu ve kterém se vyhledává a názvu akce. Získanou akci je možné přiřadit k vytvářeným komponentům, nebo je možné ovládat vlastnosti akce (povolit/zakázat).

5.6.6 ResourceManager

Zpřístupňuje zdroje z resource souborů. Tuto možnost můžeme využít vedle automatické injekce. K získání libovolného zdroje (metody `getXxx`, kde xxx je typ zdroje – řetězec, ikona, atd.) je zapotřebí zadat instanci nějaké třídy (podle které se určí resource soubor) a žádaný klíč. Tato služba zpřístupňuje funkce správce zdrojů ze SAF.

Pomocí metody `injectResources` je možné do instance libovolné třídy injektovat zdroje z resource souboru. Při použití této funkce se neinjektují služby, jen zdroje.

5.7 Modifikace provedené v knihovně Flamingo

Knihovna Flamingo je navržen tak, že pás karet je vytvořen na začátku a pak je jen používán, nepočítá s přidáváním a ubíráním karet, skupin ani tlačítek a jiných ovládacích prvků. Proto bylo stažena aktuální verze z CSV repository a následně přidané funkce pro přidávání a smazání karet a skupin.

5.8 Modifikace provedené v knihovně Swing Application Framework

K správné funkci aplikace bylo zapotřebí provést změny v SAF. Kvůli provedeným změnám je modifikovaná verze SAF součástí MSFX.

Z pohledu editoru (Netbeans IDE) je každý modul zvlášť jedna GUI aplikace. Automaticky generovaný kód hledá instanci třídy `Application` podle z něj zděděné třídy, která v daném projektu reprezentuje instanci `Application`. SAF počítá s různými třídami `Application` v jedné aplikace, proto původní implementace by vracela pro každý modul jinou instanci `Application`. Navíc framework nepoužívá třídu Application pro řízení životního cyklu aplikace. Přesněji nepoužívá ji vůbec, je součástí každého modulu jen kvůli tomu, že editor ji vyžaduje a odkazuji na ni v generovaném kódu.

Z toho důvodu bylo provedená taková změna, že se žádná třída zděděna od `Application` se nevytváří (v metodě `launch`) a na jakýkoliv vyhledávací dotaz (`getInstance`) se vrací implicitní hodnota (prázdná implementace `NoApplication`, která je interní třídou v `Application`).

Při kompilaci projektu modulu v Netbeans je třeba dávat pozor na to, aby v nastaveních projektu v části, kde se definují použité knihovny byla originální knihovna SAF pod knihovnou MSFX. Netbeans IDE potřebuje původní knihovnu mít mezi knihovnami, aby bylo možné plnohodnotně použít GUI editor (knihovnu dává automaticky do projektu, když se vytváří nový projekt).

6 Závěr

Cílem práce bylo navrhnout a implementovat framework pro modulární GUI aplikace využívající pás karet, která bude využita v plánované komerční aplikaci. Výsledkem je framework s názvem Module Suite Framework X (MSFX).

Prostudováním existujících frameworků se zjistilo, že SAF sám o sobě nenabízí možnosti modularizace aplikace. Dále, že zmíněné komplexní frameworky (Netbeans Platform a Eclipse RCP) jsou příliš rozsáhlé a vyžadují dlouhý učící cyklus; navíc aplikace vytvořená pomocí nich vypadají, jako IDE postavené na nich. Dále také nepodporují použití pásu karet. Eclipse RCP navíc nevyužívá knihovnu Swing, ale knihovnu SWT. Tyto fakty vedli k potřebě vytvořit vlastní, řádově jednodušší framework. Navíc vlastní framework může být v budoucnu přizpůsobená podle potřeb aplikací, které ho budou používat.

Prostudování frameworků a použitých technologií se docílilo prohloubení znalostí autora v oblasti modulárních aplikací a technologií jazyka Java.

MSFX umožňuje spouštět moduly pomocí OSGi (moduly jsou nezávislé soubory, načítají se dynamicky podle potřeby; také MSFX je jeden z modulů a spouští ho OSGi) nebo standardním způsobem (moduly jsou součást aplikace, jejich třídy se specifikují při inicializaci frameworku). Moduly v Netbeans IDE se dají vytvářet jako modifikované projekty typu Java Desktop Application.

Moduly definují pomocí anotací své ovládací prvky na pásu karet a též své panely, které spravuje framework. Panely je možné díky využití komerční JIDE Docking Framework přemísťovat a dokovat k jiným panelům a tím má uživatel k dispozici pohodlný způsob přizpůsobení okna.

Ve frameworku zatím není implementována služba pro správu modulů, a nefunguje zastavení modulů (při použití OSGi).

Literatura

[1] *A Brief History of NetBeans* [online]. Dostupný z WWW: <http://www.netbeans.org/about/history.html>.

[2] *An Introduction to the Swing Application Framework API (JSR-296)* [online]. Dostupný z WWW: <https://appframework.dev.java.net/intro/index.html>.

[3] *What's the Difference between NetBeans Platform and Eclipse RCP?* [online]. Dostupný z WWW: <http://platform.netbeans.org/articles/diff-nb-eclipse.html>.

[4] FOWLER, Amy. *A Swing Architecture Overview* [online]. Sun Microsystems. Dostupný z WWW: <http://java.sun.com/products/jfc/tsc/articles/architecture/>.

[5] *JIDE Docking Framework Developer Guide* [online]. Dostupný z WWW: <http://www.jidesoft.com/products/JIDE_Docking_Framework_Developer_Guide.pdf>.

[6] *Eclipse (software)* [online]. Dostupný z WWW: <http://en.wikipedia.org/wiki/Eclipse_(software)>.

[7] *OSGi™ Service Platform Release 4* [online]. Dostupný z WWW: <http://www.osgi.org/javadoc/r4v401/>. Javadoc OSGi specifikace.

[8] *OSGi Alliance Specifications* [online]. Dostupný z WWW: <http://www.osgi.org/Specifications/HomePage>.

[9] *Equinox* [online]. Dostupný z WWW: <http://www.eclipse.org/equinox/>. Webová stránka projektu.

[10] *NetBeans Platform* [online]. Dostupný z WWW: <http://platform.netbeans.org/description.html>. Webová stránka platformy.

[11] Grouchnikov, Kirill. *Pushing Pixels* [online]. Dostupný z WWW: <http://www.pushing-pixels.org/?cat=9>. Blog vývojáře komponenty Flamingo.

[12] Grouchnikov, Kirill. *Flamingo Swing component suite* [online]. Dostupný z WWW: <https://flamingo.dev.java.net/>. Stránka komponenty Flamingo.

[13] *Trail: The Reflection API* [online]. Dostupný z WWW: <http://java.sun.com/docs/books/tutorial/reflect/index.html>.

[14] *JIDE Common Layer (Open Source Project)* [online]. JIDE Software. Dostupné z WWW: <http://www.jidesoft.com/products/oss.htm>.

[15] *JIDE Docking Framework* [online]. JIDE Software. Dostupné z WWW: <http://www.jidesoft.com/products/dock.htm>.

[16] *JIDE Docking Framework Developer Guide* [online]. JIDE Software. Dostupné z WWW: <http://www.jidesoft.com/products/JIDE_Docking_Framework_Developer_Guide.pdf>.

[17] *Enhance any swing components with a busy state* [online]. Dostupné z WWW: <http://code.google.com/p/jbusycomponent/>. Stránka projektu komponenty JBusyComponent.

[18] *Annotations* [online]. Dostupný z WWW: <http://java.sun.com/j2se/1.5.0/docs/guide/language/annotations.html>.

[19] *Launching and Embedding Apache Felix* [online]. Dostupné z WWW: <http://cwiki.apache.org/FELIX/launching-and-embedding-apache-felix.html>.

Seznam příloh

Příloha A: Obsah CD

Adresář source

Složka obsahuje zdrojové kódy frameworku v Netbeans IDE projektech, všechny potřebné knihovny pro překlad. Dále je obsažen zdrojový kód a projekty ukázkových modulů a spouštěcích aplikací. Obsažen je také zdrojový kód modifikované verze knihovny Flamingo, také s projektem pro Netbeans IDE. Jednotlivé podadresáře jsou (každá /kromě posledního/ obsahuje zdrojové kódy a projekt pro Netbeans IDE):

- `ModuleSuiteFrameworkX` – MSFX framework
- `MSFXModule1` – ukázkový MSFX modul – GUI k službě (viz Příloha B)
- `MSFXModule2` – ukázkový MSFX modul – služba (viz Příloha B)
- `LnfMSFXModule` – MSFX modul pro nastavování look-and-feel
- `ModuleLauncherBundle` – OSGi bundle, která zobrazí okno s tlačítkami, pomocí kterých se spouští MSFX moduly
- `StaticLauncher` – spouštěč se statickou správou modulů (bez OSGi)
- `OSGiLauncher` – spouštěč s dynamickou správou modulů (s OSGi)
- `OSGiManualLauncher` – to samé jako projekt `OSGiLauncher`, jen moduly jsou spuštěny pomocí tlačítek v okně, kterou zobrazí bundle `ModuleLauncherBundle`
- `Flamingo` – obsahuje projekt s modifikovaným zdrojovým kódem knihovny Flamingo
- `Libraries` – nemobsahuje projekt, ale knihovny potřebné pro překlad všech aplikací

Projekty jsou mezi sebou provázány, takže není možné je kompilovat bez příslušných projektů, které používají.

Adresář binary

Tato složka obsahuje zkompilovanou verzi aplikací: se statickou správou modulů, s pomocí OSGi a též pomocí OSGi, ale se spouštěním modulů pomocí tlačítka ve zvláštním okně. Jednotlivé podadresáře jsou:

- `StaticLauncher` – spouštěč se statickou správou modulů (bez OSGi)
- `OSGiLauncher` – spouštěč s dynamickou správou modulů (s OSGi)
- `OSGiManualLauncher` – to samé jako projekt `OSGiLauncher`, jen moduly jsou spuštěny pomocí tlačítek

Programy se spustí spuštěním jar souboru se stejným jménem, jako jméno adresáře ve kterém jsou. Každý z adresářů obsahuje všechny potřebné knihovny a moduly – jsou spustitelné samostatně.

Adresář text

V tomto adresáři je text této práce v původním formátu a v PDF. Dále jsou zde umístněny všechny použité obrázky v textu.

Příloha B: demonstrační program

Ukazuje použití MSFX modulů a služeb. Všechny tři spouštěcí projekty spouští tuto ukázkovou aplikaci pomocí různými způsoby. Aplikace se skládá z modulu pro službu, modulu používající službu a z modulu, která umožňuje měnit look-and-feel aplikace. Všechny ikony použity jsou z volné sady ikon z adresy http://www.slunecnice.cz/sw/vista-toolbar-icon-collections/.

Modul MSFXModule2

Modul služby, zobrazuje informace o prováděných operacích se službou a zobrazuje statistiku v pravém panelu. Zaregistruje službu, která poskytuje seznam položek a umožňuje načíst jejich obsah. Seznam i obsah položek se generuje pomocí náhodných čísel a trvání vlastní operace je simulována pomocí čekání náhodné délky.

Modul MSFXModule1

Modul, který používá službu z předchozího modulu. Modul po prvním přepnutí na její kartu čeká na zaregistrování služby. Poté co zjistí, že služba je zaregistrována, tak načte seznam položek. Po načtení je možné vybrat položku v seznamu a načte se její obsah. Během načítání seznamu položek i obsahu položky je příslušná komponenta zaneprázdněna.

Modul LnfMSFXModule

Modul, který umožňuje měnit look-and-feel aplikace. Možné je vybrat jeden z nainstalovaných look-and-feel.