Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Diplomová práce

Převod testovacího nástroje SimCo na Eclipse plugin

Plzeň, 2013 Michal Bokr

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 15. 5. 2013

Michal Bokr

Abstract

The title of this thesis is testing tool Simco to Eclipse plugin transition. Simco is framework, parts of which were created by Tomáš Kubíček and Matěj Prokop as a part of their Master's thesis at the University of West Bohemia in Pilsen in 2011. This thesis aims to look into basic principles of component-based software engineering, introduces creating plugins for the development enviroment Eclipse as well as in this enviroment. The result of this thesis is the user friendly plugin integrated in Eclipse IDE and linked with framework Simco.

Obsah

1 Úvod

Podle TIOBE indexu, který je měřítkem oblíbenosti programovacích jazyků, je aktuálně nejpoužívanějším programovacím jazykem jazyk C a na druhém místě je jazyk Java. Tyto dva programovací jazyky se drží stabilně v popředí déle než deset let a v prvenství popularity se již několikrát prostřídaly. Programovací jazyk Java má na trhu důležité zastoupení. Pro vývoj Java programů existuje velké množství vývojových prostředí zdarma, ale existují i placená řešení. Podle některých zdrojů až 70 procent Java vývojářů používá jako své primární vývojové prostředí Eclipse IDE. Eclipse vznikl ve společnosti IBM. V roce 2004 se od společnosti oddělil a vznikla nezisková nadace Eclipse Foundation, která nyní Eclipse IDE vyvíjí. Své popularitě vděčí jak své nezávislosti, která například přílákala Google, aby zvolil toto IDE pro vývoj aplikací pro Android zařízení, tak své flexibilní architektuře. Vše je založeno na mechanismu pluginů, které Eclipse IDE umožňují mimo jiné pracovat s jinými programovacími jazyky.

Cílem této práce je vytvořit Eclipse plugin pro nástroj Simco. Simco je testovací nástroj, jehož části byly vytvořeny Tomášem Kabíčkem a Matějem Prokopem a popsány v jejich diplomových pracích na Katedře informatky a výpočetní techniky Západočeské univerzity v Plzni v roce 2011. Hlavním tématem této práce je problematika pluginů, jejich struktura a možnosti jejich vývoje pro a v prostředí Eclipse.

Důležitým aspektem pro úspěšný převod Simco, aby fungoval jako Eclipse plugin, je porozumění tomuto nástroji. Simco nástroj je založený na principech komponentového programování a slouží pro simulační testování komponentů. Proto se tato práce zabývá také problematikou komponentového programování a zvláště komponentového frameworku Spring DM

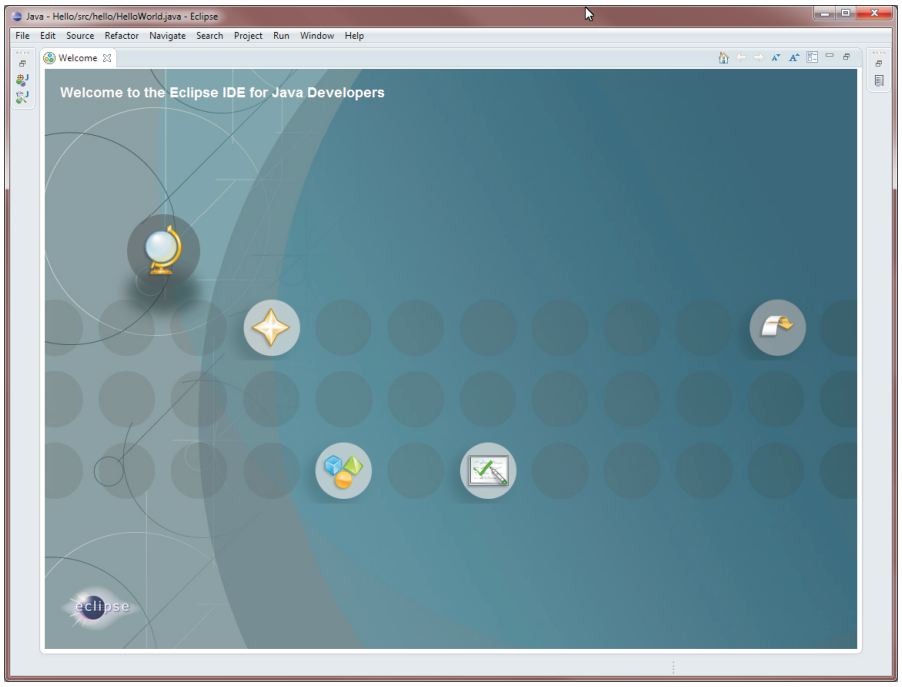
V první části se práce věnuje vývojovému prostředí Eclipse, dále Eclipse pluginům, jejich vytvoření v tomto prostředí a jejich strukturou. Další část pojednává o základních principech komponetového programování a o testovacím nástroji Simco. Realizační část obsahuje analýzu možností řešení a implementaci pluginu vyvíjeného v rámci této diplomové práce.

2 Eclipse

Eclipse je populární vývojové prostředí neboli IDE (Integrated Development Enviroment), které je především určené pro programování v jazyce Java, v kterém je samotné i vytvořeno. Eclipse je otevřený software a založen na filosofii komponent, což umožňuje jeho snadné rozšíření (podrobněji další části). Eclipse poskytuje vývojářům velkém množství nástrojů pro kompilaci, spouštění a ladění programů, sdílení kódu a mnoho dalšího. Existuje mnoho verzí vývojového prostředí Eclipse, které jsou určeny pro různé programovací jazyky a technologie. [1]

2.1 Základní koncepce

Po instalaci jakékoliv verze Eclipse je zobrazena uvítací stránka (Welcome), která má za úkol představit uživateli vývojové prostředí viz obrázek 1. Grafika uvítací stránky je v každé verzi jiná, ale vždy zde lze najít odkazy na jednoduché příklady a průvodce. [1]



Obrázek 1 – Ukázka uvítací stránky Eclipse verze Juno (Welcome)

2.1.1 Editor

Editor slouží k úpravám souborů, např. editor Java tříd či HTML editor. V jednom momentě může být spuštěno více editorů, avšak pouze jeden může být aktivní. [1]

2.1.2 View

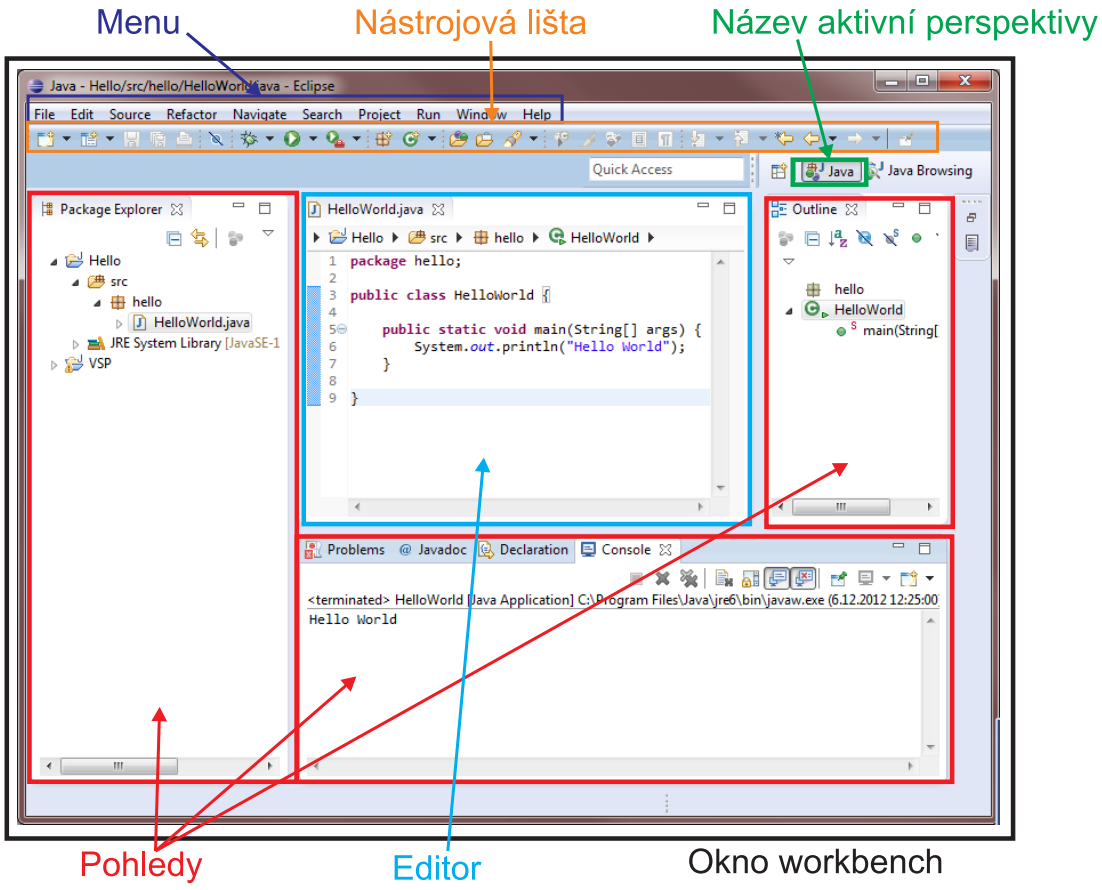
Pohledy slouží jako podpora pro editory. Poskytují různé možnosti prohlížení, zobrazování informací, otevírání souborů a orientace ve workbench viz 2.1.4. Pohled často mívá své menu, které ovlivňuje pouze tento pohled.[1]

2.1.3 Perspective

Perspektiva slučuje a zobrazuje editory (Editor) a pohledy (View). Dále také určuje položky menu a nástrojových lišt. Uživatel si může nakonfigurovat perspektivu přesně tak, jak mu to vyhovuje. V rámci jedné instance workbench (viz 2.1.4) může být v jednom momentě zapnuta pouze jedna perspektiva. [1]

2.1.4 Workbench

Pracovní plochou (Workbench) v Eclipse můžeme nazvat veškeré uživatelské rozhraní. Může být spuštěno více instancí workbench najednou. Workbench se především skládá z jedné nebo více perspektiv (Perspective), které se skládají z editorů (Editor) a pohledů (Views). [1] Dále v okně workbench nalezneme menu a nástrojovou lištu viz obr. 2



Obrázek 2 – Okno Workbench

2.2 Eclipse RCP

Eclipse RCP (Rich Client Platform) je flexibilní platforma, která poskytuje programátorům možnosti ji využít k tvorbě různorodých aplikací. Jinak řečeno Eclipse není jednotný monolitický program. Jeho základem je jádro, které je obklopeno malými funkčními bloky. Tyto komponenty jsou nejmenší instalovatelnou součástí Eclipse RCP. [2]

2.3 Eclipse architektura

Jak bylo zmíněno, Eclipse obsahuje samostatné softwarové komponenty. Na Eclipse IDE můžeme pohlížet jako na Eclipse aplikaci s důrazem na podporu vývoje softwaru. Nyní budou stručně popsány nejdůležitější části Eclipse architektury zobrazené na obrázku 3.

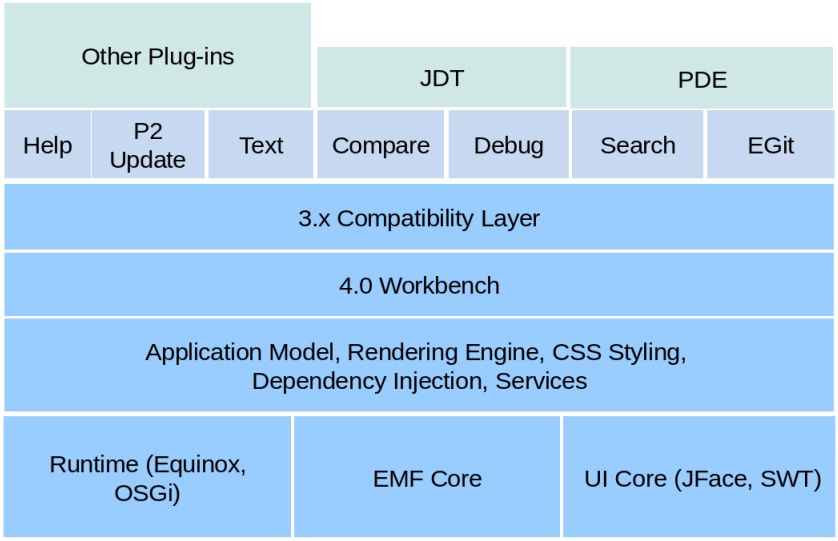
OSGi je specifikace, která popisuje modulární přístup k Java aplikacím viz 6.4. Equinox je jedna z implementací OSGi a je používána na platformě Eclipse. Equinox runtime poskytuje potřebný framework[1] pro spuštění modulární Eclipse aplikace. SWT (Standard Widget

[1] Obecně je framework nebo také vývojová platforma skutečná nebo konceptuální struktura sloužící jako

Toolkit) je knihovna komponent poskytující grafické prvky pro standardní uživatelské rozhraní používané vývojovým prostředím Eclipse.

JFace je nadstavbou SWT a poskytuje rozhraní pro práci s komplexnějšími prvky SWT. Workbench poskytuje framework pro aplikaci a odpovídá za zobrazení všech součástí uživatelského rozhraní.

Nejaktuálnější verze Eclipse 4 má o trochu jiný programovací model než starší verze Eclipse 3.x. Eclipse 4 však poskytuje kompatibilní vrstvu (3.x Compatibility Layer), která mapuje 3.x API (Application Program Interface)[2] na 4.0 API. Takže komponenty založené na 3.x mohou běžet na Eclipse 4 beze změn. Eclipse aplikace, které nejsou primárně používány jako nástroj pro vývoj software se nazývají Eclipse RCP aplikace. Eclipse RCP 4 aplikace obvykle používá základní komponenty platformy Eclipse a přidává další jednotlivé komponenty. [2]



Obrázek 3 – Architektura Eclipse [2]

podpora nebo průvodce pro budování něčeho, co rozšiřuje strukturu v něco užiteného. V počítačových systémech je framework součástí vrstvené struktury určující jaké programy mohou nebo by měly být vytvořeny a jak se vzájemně propojují. Zdroj: http://whatis.techtarget.com/definition/framework

[2] API je sada rutin, protokolů a nástrojů pro vytváření softwarových aplikací. Dobré API usnadňuje vytvořit program prostřednictvím všech stavebních bloků. Programátor je pak poskládá dohromady. Zdroj: http://www.webopedia.com/TERM/A/API.html

3 Pluginy a Eclipse

Tato kapitola popisuje vnitřní strukturu vývojového prostředí Eclipse a především se zaměřuje na strukturu pluginů v Eclipse.

3.1 Plugin

Plugin (označován také plug-in) nebo také zásuvný modul je hardware nebo software, který přidává specifický prvek nebo službu do rozsáhlejší aplikace. Jinak řečeno rozšiřuje její funkčnost. Modul obvykle poskytují speciální aplikační rozhraní zvané API (Application Programming Interface), které umožňuje její snadné rozšíření. Nová komponenta je tedy připojena *(*anglicky: *plugged in)* do existujícího systému. [1]

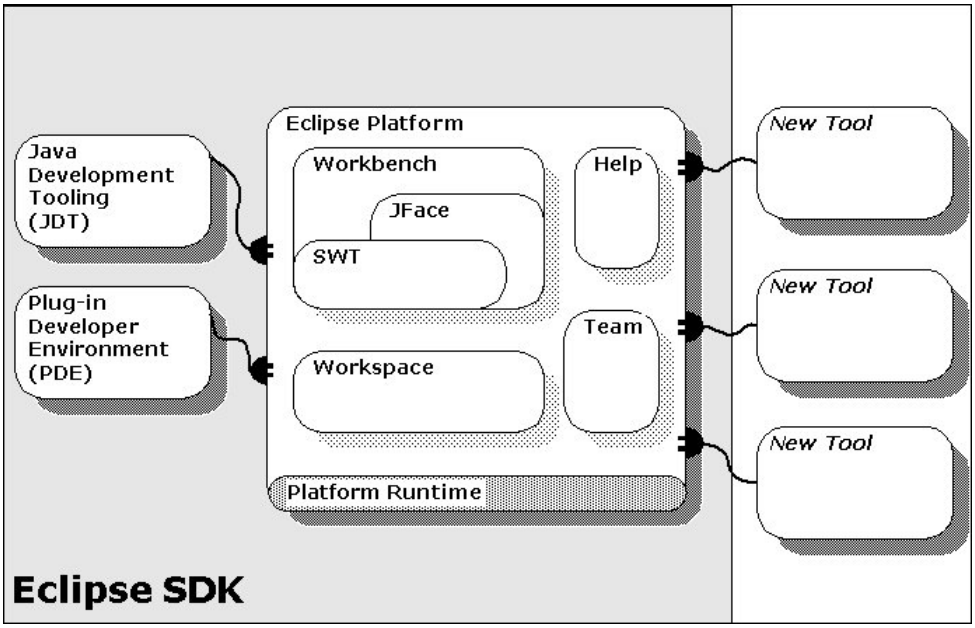
3.2 Platforma Eclipse

V předcházejí části jsme zmínili komponenty nebo také funkční bloky, které se nazývají *pluginy*. Obdobná softwarová komponenta v OSGi se nazývá *bundle*. Tyto pluginy jsou nejmenší instalovatelnou součástí Eclipse RCP. Platforma Eclipse (Eclipse platform) je strukturovaný koncept založený na pluginech. [2, 1]

3.2.1 Architektura platformy Eclipse

Platforma obsahuje několik strukturovaných subsystémů, které jsou složeny ze sady pluginů implementující nějakou ústřední funkci. Některé pluginy přidávají viditelné vlastnosti platformě díky systému rozšíření (system extensions). Ostatní nabízejí knihovny tříd, které mohou být použity k provedení rozšíření systému. Eclipse SDK (Software Development Kid)[3] zahrnuje základní platformu a dva hlavní nástroje, které jsou využitelné pro vývoj pluginů viz obrázek 4. Tyto nástroje jsou nejen užitečné, ale také ukazují výborný příklad toho, jakým způsobem lze nové nástroje přidat do platformy.

[3] Programový balíček, který umožňuje programátorovi vývoj aplikací pro konkrétní platformu. Typicky SDK obsahuje jedno nebo více rozhraní API, programovací nástroje a dokumentaci. Zdroj: http://www.webopedia.com/TERM/S/SDK.html



Obrázek 4 – Architektura Platformy Eclipse [1]

Platform Runtime Platforma Runtime Core implementuje Runtime Engine, který nastartuje základ platformy a dynamicky najde a spustí pluginy. Platforma udržuje evidenci nainstalovaných pluginů a jejich funkčnost, kterou poskytují. Obecným cílem Platform Runtime je, že uživatel by neměl ztrácet dostupnou pamět nebo výkon kvůli pluginům, které jsou naistalovány, ale nejsou využívány. Plugin může být nainstalován a přidán do registru, ale nebude aktivován, dokud funkce jím poskytované nebudou vyžádány uživatelem. Platforma Runtime je implementována tak, aby používala OSGi model služeb.[1]

Workbench UI Workbench implementuje plugin Workbench UI, definuje mnoho extensions points a tím umožňuje jiným pluginům rozšiřovat a vytvářet menu, nástrojové lišty, dialogy a průvodce apod. Části Workbench SWT a JFace jsou popsané v podkapitole 2.3. [1]

Workspace Workspace je centrální prostor pro uživatelská data. Resource plugin poskytuje API pro vytváření, navigaci, manipulaci se zdroji ve workspace. Workbench využívá těchto API k zajištění této funkčnosti uživateli. Workspace obsahuje soubor zdrojů. Pokud náš plugin vyžaduje Resource plugin, tak tento Resource plugin je spuštěn dříve než náš plugin a zdroje z workspace jsou nám k dispozici. Z pohledu uživatele existují tři různé typy zdrojů: projekty (Project), adresáře (Folder) a soubory (File). Projekt je kolekce libovolného počtu adresářů a souborů. Adresáře a soubory představují odpovídající entity v souborovém systému. Workspace je organizován do stromové struktury, kde projekty jsou uzly nejvyšší úrovně, a adresáře a soubory pod nimi. [1]

Team Support Team pluginy umožňují jiným pluginům definovat a registrovat implementaci pro týmové programování, přístup do úložiště, správu konfigurace úložiště a verzování. Eclipse SDK obsahuje také podporu pro CVS klienta. [1]

Debug Support Debug pluginy usnadňují dalším pluginům implementovat prostředí pro ladění (Debuger). [1]

Help System Help pluginy definují rozšiřující místa (extensions points), které mohou použít jiné pluginy s cílem rozšířit nápovědu nebo jinou dokumentaci. [1]

Java Development tools (JDT) Java Development Tools rozšiřují platformu workbench tím, že poskytují specializované funkce pro editaci, prohlížení, sestavování, ladění a spouštění Java kódu. [1]

Plug-in Development Environment (PDE) Plugin Development Environment nabízí nástroje pro automatické vytváření, manipulaci, ladění and sestavování pluginů. [1]

3.2.2 Konfigurační soubory

Eclipse plugin charakterizují dva konfigurační soubory, které se také nazývají plugin manifest.

- `MANIFEST.MF` - Obsahuje OSGi konfigurační informace.
- `plugin.xml` - Obsahuje informace o Eclipse mechanismu rozšiřování (Extension mechanism)

Eclipse plugin používá soubor `MANIFEST.MF` k definici svého API. Říká například jaké Java balíčky (Packages) mohou být použity jinými pluginy, definuje své závislosti na jiných pluginech, apod.

Soubor `plugin.xml` poskytuje možnosti k definování rozšiřujících míst (extension point) a rozšíření (Extension). Extension point definuje rozhraní, které může být využito ostatními plugin pro jeho rozšíření. Extensions přidávají funkčnost k těmto rozhraním. Funkčnost může být přidána ve formě zdrojového kódu jako knihovna, rozšíření platformy nebo dokumentace. [2]

3.2.3 Instalace

Aktuální verzí Eclipse je Juno. Pro vývoj pluginů existuje speciální balíček.

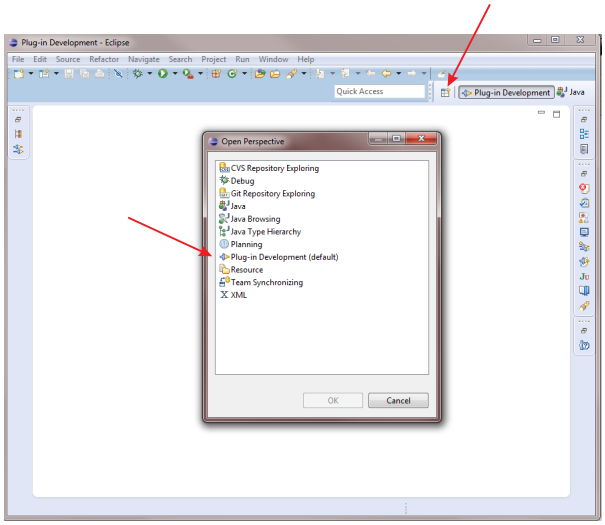
- Eclipse Juno 4.2 for RCP and RAP Developers
- Ke stažení na http://www.eclipse.org/downloads

4 Ukázka jednoduchého pluginu

Tato kapitola se věnuje více samotnému vývoji pluginů v Eclipse. Abychom mohli popsat strukturu Eclipse pluginu lépe, ukážeme si nejdříve krok po kroku, jak vytvořit jednoduchý Eclipse plugin, jakým způsobem ho nainstalovat, spustit a ladit v prostředí Eclipse.

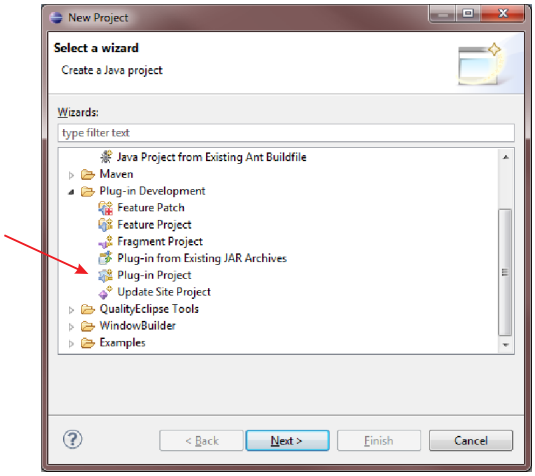
4.1 Vytvoření projektu

1. K tvorbě pluginů nabízí vývojové prostředí perspektivu PDE, tedy přepneme na **Plug-in Development perspective**.



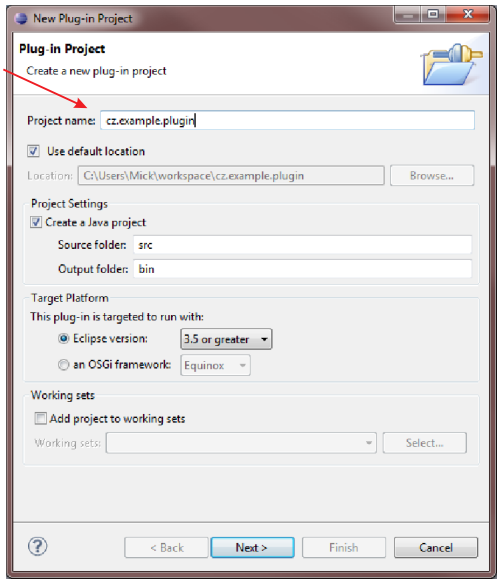
Obrázek 5 – Otevření perspektivy PDE

2. V menu **File**, zvolíme **New** > **Project** ke spuštění průvodce (Wizard) **New Project**. V tomto okně nalezneme adresář Plug-in Development a zvolíme **Plug-in Project**. Potvrdíme tlačítkem **Next.**



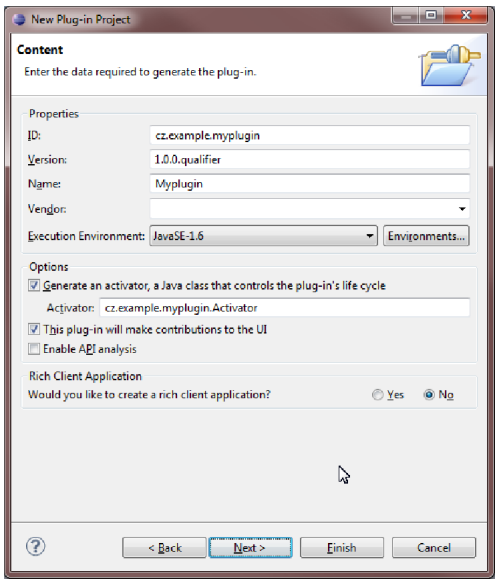
Obrázek 6 – Okno New Project

3. V okně **Plug-in project** nastavíme jméno projektu, např. `cz.example.myplugin`



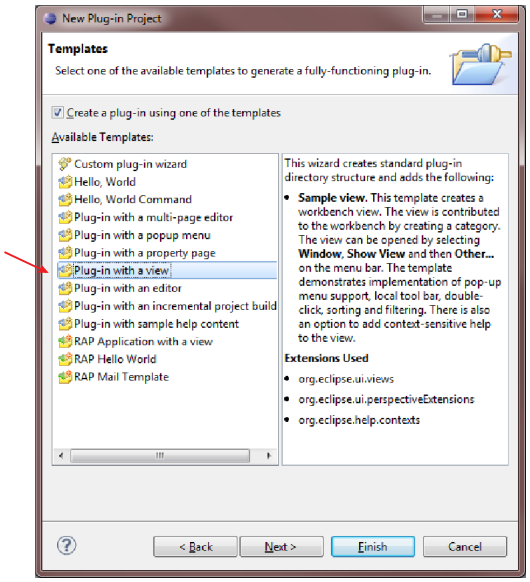
Obrázek 7 – Okno Plug-in project

4. V okně **Content** průvodce nastaví jméno projektu jako id pluginu. Průvodce také generuje plug-in class pro plugin. Výchozí hodnoty jsou pro náš plugin dostatečné, informace potvrdíme tlačítkem **Next**.



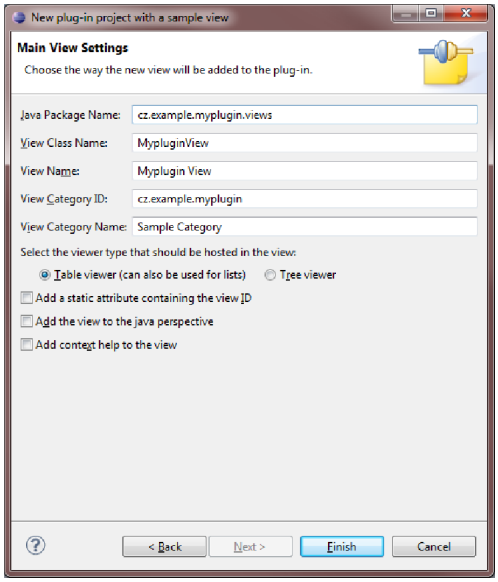
Obrázek 8 – Okno Content

5. V následujícím okně **Templates** můžeme zvolit z mnoha typů šablon, které budou automaticky vygenerovány. Vybereme **Plug-in with a view** a stiskneme tlačítko **Next**.



Obrázek 9 – Okno Templates

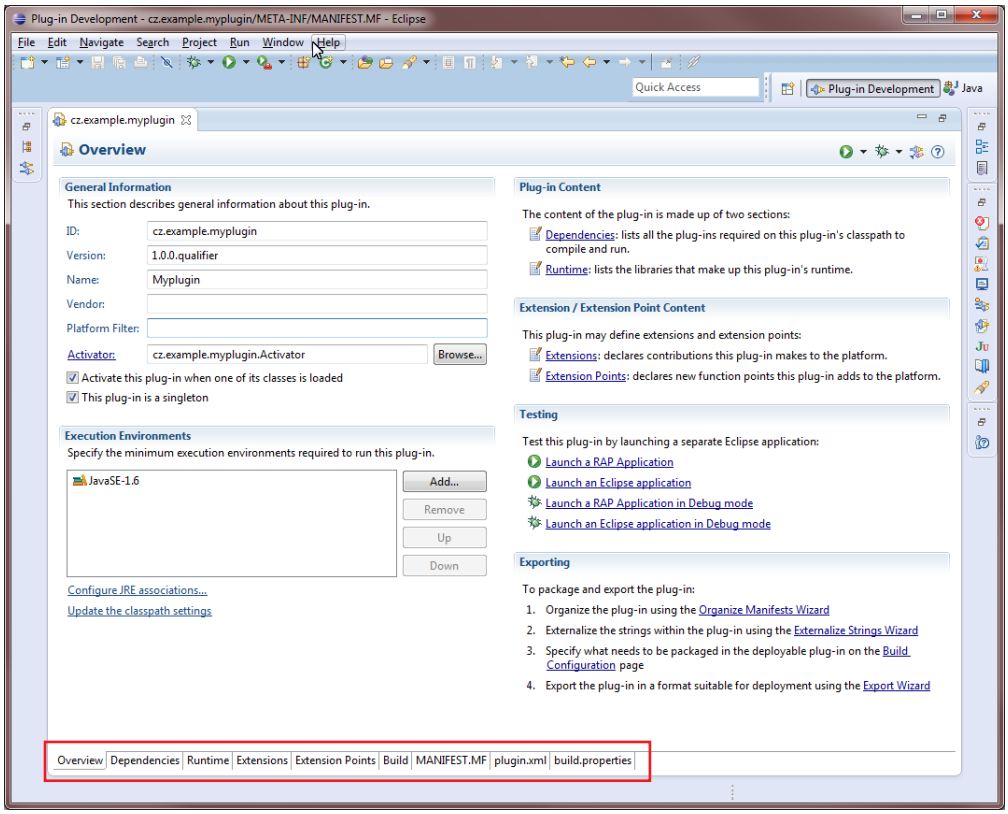
6. V posledním okně **Main View Settings** vše odznačíme, abychom co nejvíce zjednodušili vygenerovaný plug-in manifest soubor. Můžeme změnit názvy tříd a pro dokončení klikneme na **Finish**.



Obrázek 10 – Okno Main View Settings

4.2 Orientace v PDE

Náš projekt je automaticky otevřen na první stránce **Overview**. Dole vidíme, že je zde devět různých záložek viz obrázek 11.



Obrázek 11 – Okno Overview

Overview zobrazuje souhrn dat z `MANIFEST.MF` a umožňuje jejich úpravu. Také je možné úpravu dělat přímo přepnutím na záložku **MANIFEST.MF**.

Dependencies udávají závislosti pluginu na jiné pluginy v systému.

Runtime definuje knihovny, které jsou nutné při spuštění (pomáhá k urychlení doby načítání).

Extensions ukazuje, jakým způsobem tento plugin rozšiřuje svojí funkčnost poskytovanou jinými, které jsou již v systému zavedeny (deklarovaný extension `org.eclipse.ui` využívající extension point `org.eclipse.ui.views`).

Extensions Point umožňuje definici míst rozšíření, tj. definuje jakým způsobem mohou jiné pluginy využít tento pro své rozšíření. Náš plugin neumožňuje žádné rozšíření.

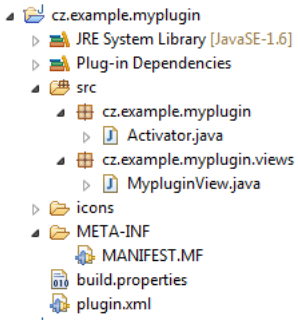
Build konfigurace sestavení, které knihovny budou použity. Upravuje soubor `build.properties`.

MANIFEST.MF zobrazuje soubor `MANIFEST.MF`, který definuje OSGi konfigurační informace.

plugin.xml zobrazuje stejnojmený soubor, který definuje aspekty rozšiřování pluginu.

build.properties soubor konfigurace sestavení.

V pohledu Package Explorer vidíme vše, co vygeneroval průvodce **New Plug-in Project** viz obr. 12



Obrázek 12 – Obsah ukázkového pluginu

4.2.1 Aktivátor (Activator) nebo Plug-in třída (Class)

Zajišťuje kontrolu životního cyklu pluginu. Eclipse systém automaticky vytváří vždy jen jednu aktivní instanci této třídy.

4.2.2 View Class

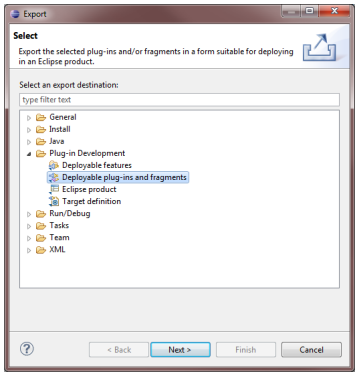
Vygenerovaný kód pro jednoduchý pohled (View) plugin.

4.3 Sestavování programu (Building)

K sestavování programu můžeme použít průvodce Eclipse nebo Eclipse podporu pro Apache Ant skript. Apache Ant je Java knihovna a nástroj využívající příkazového řádku, jehož posláním je řídit postupy sestavování softwarových aplikací. Hlavní použití Ant je k sestavování Java aplikací. Ant poskytuje celou řadu úkolů pro kompilaci, sestavení, testování a spouštění Java aplikací. Ant však může být efektivně využíván k sestavování například C nebo C++ aplikací. Obecně platí, že může být použit jako pilot pro každý typ procesu, kterýlze popsat cíli (Target) a úkoly (Tasks). [4]

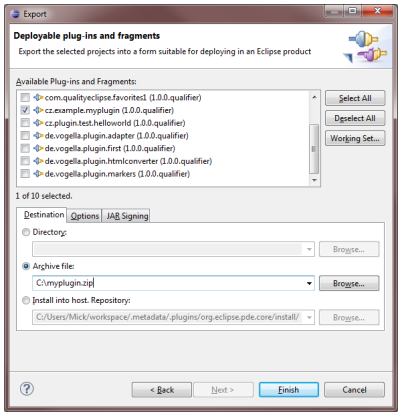
4.3.1 Průvodce Eclipse

1. V menu **File** zvolíme **Export** zde adresář **Plug-in Development** a v něm položku **Deployable plug-ins and fragments**.



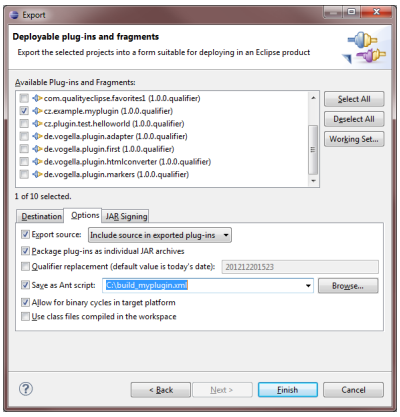
Obrázek 13 – Okno Export

2. Vybereme plugin, který chceme exportovat a zvolíme adresář (Directory), nebo archiv (Archive)



Obrázek 14 – Okno Export, záložka Destination

3. Ve stejném okně, jako v předchozím bodu v záložce **Options,** můžeme provést několik nastavení, např. můžeme uložit Ant skript nebo zahrnout zdrojové kódy do archivu pluginu.



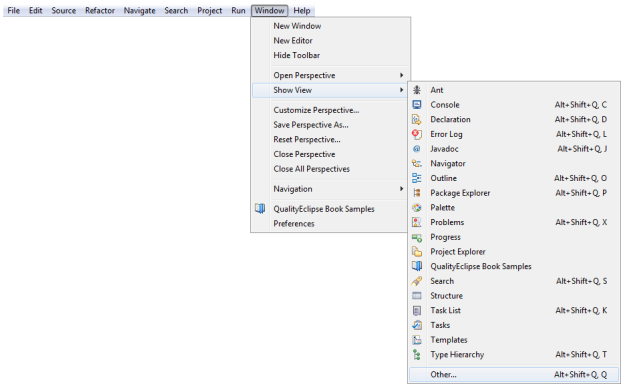
Obrázek 15 – Okno Export, záložka Options

4.4 Instalace pluginu

1. Aplikace Eclipse musí být vypnutá.
2. Soubor s příponou *.jar* přesuneme do adřesáře `.../Eclipse/Plugins`.

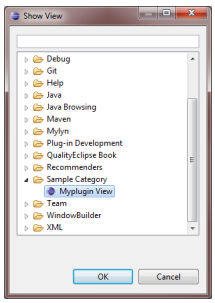
4.5 Spuštění pluginu

1. Námi vytvořený plugin zprostředkovává pohled (View). Z hlavního menu zvolíme **Window > Show View > Other...**



Obrázek 16 – Menu Window > Show View > Other..

2. V následujícím okně **Show View** nalezneme náš plugin a spustíme ho. Pokud jsme postupovali podle zmíněného návodu, bude se **Myplugin view** nacházet v adresáři **Sample Category**.



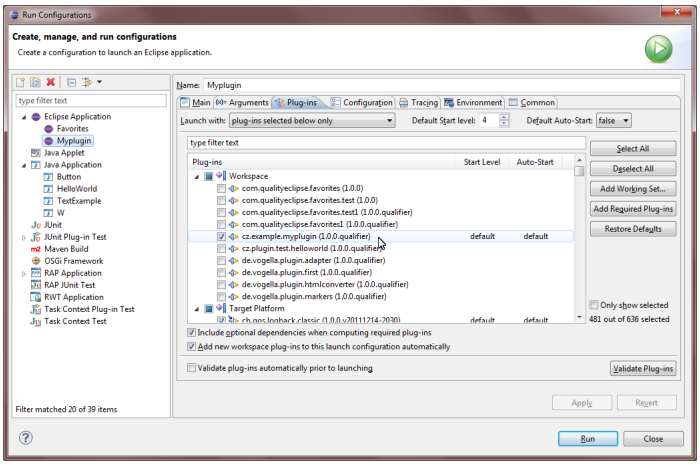
Obrázek 17 – Dialogové okno Show View

4.6 Ladění pluginu

Během vývoje pluginu ho můžeme spouštět tak, že přepneme do záložky **Overview** viz 4.2 a klikneme na **Launch an Eclipse Application**. Další možností je přístup z hlavního menu **Run** > **Run**. Vždy je však dobré vytvořit si konfiguraci pro spuštění (Run configuration), zvláště pokud máme ve workspace Eclipse více projektů.

4.6.1 Vytvoření konfigurace

Pro vytvoření konfigurace pro spuštění vybereme **Run** z hlavního menu a poté **Run configurations**. Klikneme pravým tlačítkem na položku **Eclipse Application** a zvolíme **New**. Pojmenujeme naší konfiguraci, obvykle volíme stejný název jako název projektu. Můžeme však mít více konfigurací pro jeden projekt. Nyní přejdeme na záložku **Plug-ins** a zvolíme v první řádce Launch with: **plug-ins selected below only** a z Workspace pluginů označíme pouze ten náš. Tímto krokem jsme zajistili, že budeme spouštět a testovat pouze náš plugin z celého Workspace.



Obrázek 18 – Okno Run Configuration

4.6.2 Debuger

Ladění pluginu je stejné jako ladění jakékoliv Java aplikace. Hlavně využíváme vytváření Breakpoints (bod přerušení vykonávání počítačového programu) ve zdrojovém textu programu a další možnosti perspektivy Debug. Analogicky, jako jsme vytvářeli konfiguraci pro spuštění, můžeme vytvořit konfiguraci pro ladění (Debug).

4.7 Odinstalace pluginu

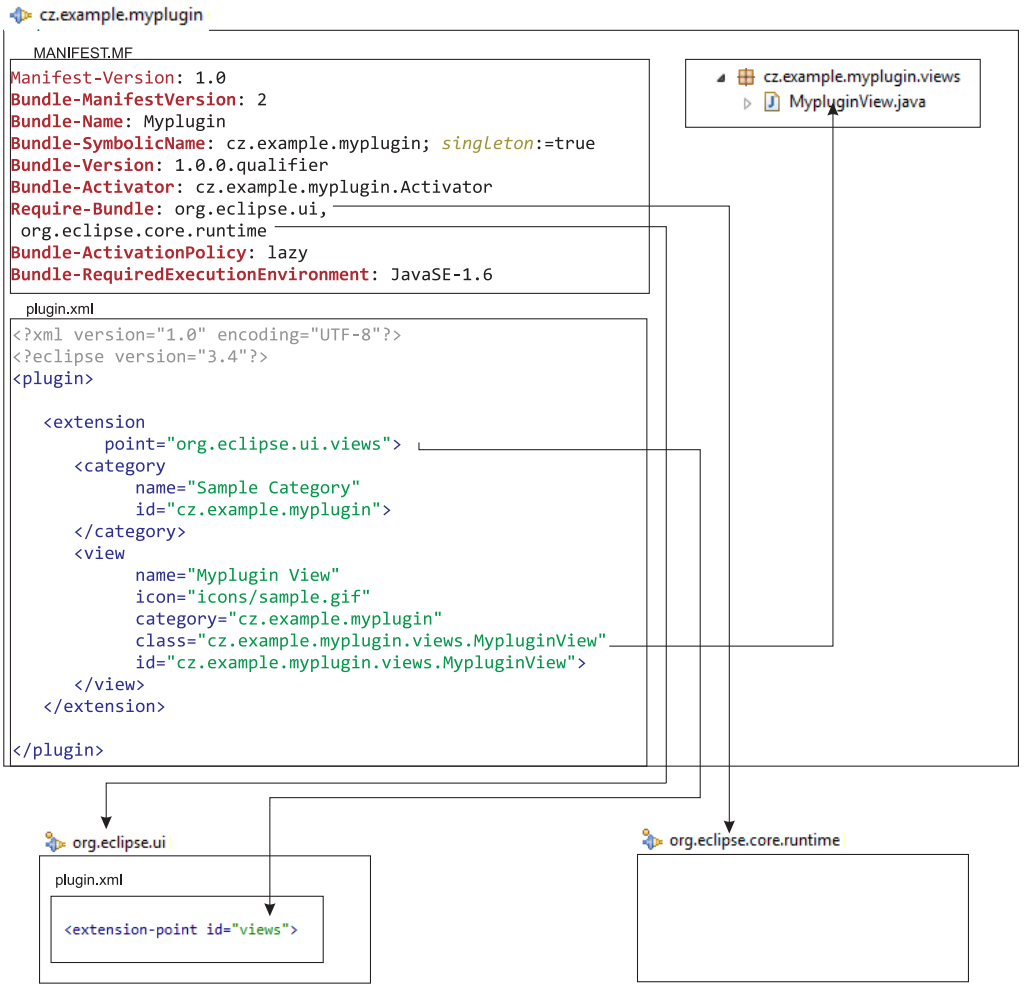
1. Zavřeme plugin.
2. Vypneme Eclipse.
3. Smažeme *.jar* soubor z adřesáře `.../Eclipse/Plugins`.
4. Znovu zapneme Eclipse. Pokud vyskočí chybové hlášení, náš plugin nebyl vypnut při vypínání Eclipse.

5 Struktura pluginu

Sturkturu pluginu rozbereme na pluginu, který jsme vytvořili v předchozí kapitole. Podíváme se na jeho důležité části podrobněji.

5.1 Strukturní propojení

Chování každého pluginu je určeno v kódu, ale závislosti a služby pluginu jsou deklarovány v `MANIFEST.MF` a `plugin.xml`, viz obr. 19. Díky této struktuře používá Eclipse tzv. lazy activation policy. Tento mechanismus zajišťuje aktivaci pluginů po načtení jeho první třídy. To umožňuje systému aktivovat pluginy líně, tedy až ve chvíli, kdy jsou potřebné. Používáním tohoto modelu lze docílit běhu nemejšího počtu aktivních pluginů. [5]



Obrázek 19 – Ukázka deklarace nového rozšíření se zvýrazněnými referencemi mezi pluginy (inspirace obrázkem z [3])

5.2 Plugin manifest

Jak bylo již výše zmíněno plugin manifest obsahuje dva konfigurační soubory `MANIFEST.MF` a `plugin.xml`. Nyní rozebereme jejich obsah.

5.2.1 Deklarace pluginu

Uvnitř každého MANIFEST.MF jsou OSGi konfigurační informace.

```
1  Manifest-Version: 1.0
2  Bundle-ManifestVersion: 2
3  Bundle-Name: Myplugin
4  Bundle-SymbolicName: cz.example.myplugin; singleton:=true
5  Bundle-Version: 1.0.0.qualifier
6  Bundle-Activator: cz.example.myplugin.Activator
7  Require-Bundle: org.eclipse.ui,
8  org.eclipse.core.runtime
9  Bundle-ActivationPolicy: lazy
10 Bundle-RequiredExecutionEnvironment: JavaSE-1.6
```

Výpis 1 – Ukázka souboru MANIFEST.MF

Manifest-Version Verze manifestu.

Bundle-ManifestVersion `Bundle-ManifestVersion` indetifikátor definuje, jakými pravidly dané specifikace se bundle bude řídit. Výchozím nastavením jsou to pravidla specifikace 1. [6]

Bundle-Name Hlavička `Bundle-Name` definuje jméno bundlu. Toto jméno by mělo být krátké a dobře čitelné. [6]

Bundle-SymbolicName `Bundle-SymbolicName` je indetifikátor, který je určen k jednoznačné indetifikaci pluginu. Používají se stejné konvence jako pro Java balíčky (packages), v našem případě `cz.example.myplugin. [3]`

Bundle-Version Každý plugin určuje svojí verzi pomocí tří čísel oddělených tečkami. První číslo udává hlavní číslo verze (major version number), druhé číslo vedlejší (minoritní) verze (minor version number) a poslední číslo ukazuje úroveň služby (service level). Také je možné zadat volitelný kvalifikátor, který má alfanumerické znaky. Při spuštění, v případě, že existují dva pluginy se stejným identifikátorem (Bundle-SymbolicName), Eclipse zvolí ten nejnovější porovnáním hlavního čísla verze, pak vedlejšího čísla verze, úrovní služeb a pokud existuje, kvalifikátorem. [3]

Bundle-Activator Plugin může volitelně specifikovat plugin aktivátor. V našem případě je to třída `Activator`.

Bundle-ActivationPolicy V našem případě nastaveno na `lazy` (líný). To znamená, že plugin bude aktivován, pokud je načtena jedna z jeho tříd. Další možností je tzv. *eager activation.* Ta je použita, pokud identifikátor Bundle-ActivationPolicy není aplikován.

Bundle-RequiredExecutionEnviroment Potřebné prostředí pro spuštění. V našem případě je to Java Standart Edition verze 1.6.

5.2.2 Závilosti (Dependencies)

Plugin zaváděč (loader) instancí odděluje zavaděč tříd (class loader) pro každý načítaný plugin. Používá deklaraci manifestu Require-Bundle k určení, které další pluginy budou tomuto viditelné během spuštění. Když je zaváděč připraven načíst plugin, prohledá deklaraci Require-Bundle a najde všechny potřebné pluginy. Pokud požadovaný plugin není k dispozici, pak zavaděč vyvolá vyjimku a nenačte závislý plugin. Pokud lze plugin spustit bez požadovaného, potom tento plugin může být označen jako volitelný (optional) v deklaraci manifestu. [3]

```
1 Require-Bundle: org.eclipse.core.runtime; resolution:=optional
```

5.2.3 Rozšíření a místa rozšíření (Extensions a extensions points)

Plugin deklaruje místa rozšíření tak, aby ostatní mohly rozšířit funkčnost původního pluginu. Tento mechanismu zajišťuje vrstvu oddělení, aby původní plugin nemusel vědět o existenci rozšiřujícího v době sestavování původního. Pluginy deklarují rozšiřující body v rámci svého plugin manifestu, jako je například rozšiřující bod pohledů (views extension point) deklarován v `org.eclipse.ui` plugin:

```
1 <extension-point
2    id="views"
3       name="%ExtPoint.views"
4       schema="schema/views.exsd"/>
```

Výpis 2 – Ukázka deklarace extension-point

V nápovědě Eclipse je tento popis:

> „Tento rozšiřující bod je používán k definování dalších pohledů pro workbench. Pohled je vizuální komponenta v rámci pracovní plochy, která se obvykle používá k navigaci v hierarchii informací.“ [3, 1]

Ostatní pluginy deklarují rozšíření (extension) k původní funkčnosti aktuálního podobně jako náš vytvořený plugin. V tomto případě Myplugin definuje kategorii pohledů s názvem **Sample Category** a třídu `cz.example.myplugin.views.MypluginView` jako nový typ pohledu. Celou deklaraci vidíme v konfiguračním souboru `plugin.xml`.

```
1   <extension
2         point="org.eclipse.ui.views">
3      <category
4            name="Sample Category"
5            id="cz.example.myplugin">
6      </category>
7      <view
8            name="Myplugin View"
9            icon="icons/sample.gif"
10           category="cz.example.myplugin"
11           class="cz.example.myplugin.views.MypluginView"
12           id="cz.example.myplugin.views.MypluginView">
13     </view>
14  </extension>
```

Výpis 3 – Ukázka souboru plugin.xml

Každý typ rozšiřujícího bodu může vyžadovat různé typy atributů, které definují rozšíření. Typicky mají ID atributy podobnou formu jako identifikátor pluginu. ID kategorie poskytuje způsob, jak jednoznačně identifikovat kategorii pro **Myplugin View**. Tento přístup umožňuje Eclipse načíst informace o rozšíření deklarovaných v různých pluginech bez načtení celých těchto pluginů a tak snížit množství času a paměti potřebné pro fungování. [3]

5.2.4 Aktivační třída (Activator class)

Výchozí nastavení aktivační třídy poskytuje metody pro přístup ke statickým zdrojům v rámci pluginu a pro přístup a incializaci specifických předvoleb pluginu. Nastavení aktivátoru není povinné, ale pokud je specifikován v manifestu pluginu, je aktivátor první třídou informovanou o načtení a poslední třídou informovanou o chystaném ukončení pluginu.

6 Komponentové programování

V této kapitole budeme nejdříve mluvit o komponentovém programování a poté o frameworku Spring DM, který umožňuje dynamický komponentový návrh. Abychom lépe pochopili, jak Spring DM funguje, podíváme ses na jeho součásti - Spring a OSGi. Nakonec popíšeme elementární části Simco frameworku, který je založený na Spring DM a určený pro simulační testování komponent.

6.1 Základní myšlenky

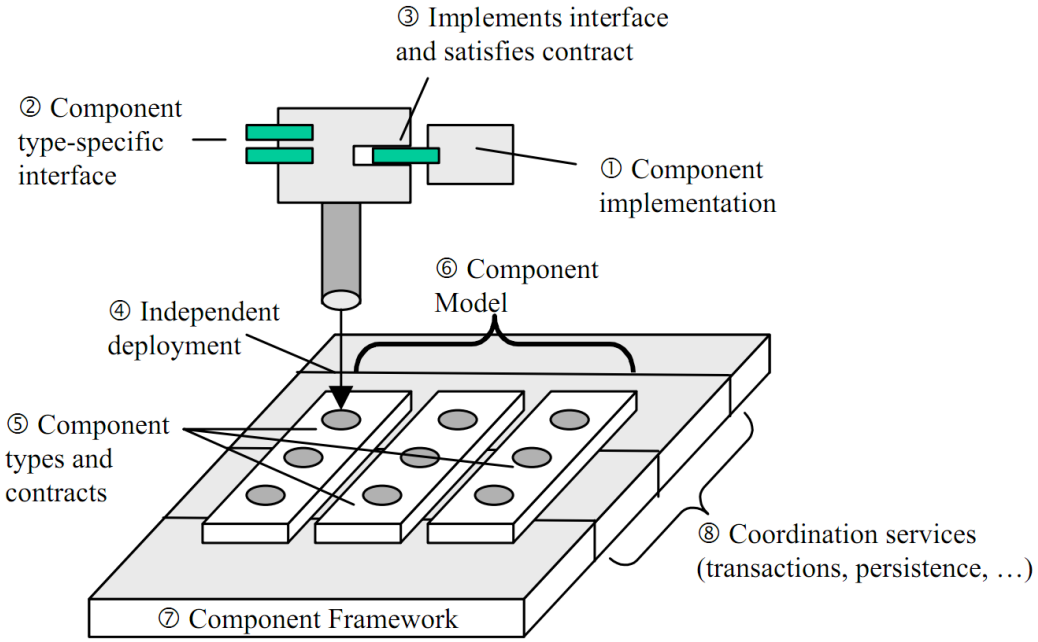
Hlavní myšlenkou komponentového programování (Component-Based Software Engineering CBSE nebo také Component-Based Development CBD) je vytváření co nejmenších samostatných segmentů. Snahou je, aby komponenty byly tzv. „black boxes", neboli černé skřínky, tzn., aby jejich implementace byla skrytá. Pro další využití komponenty je k dispozici pouze rozhraní (interface), které popisuje funkcionalitu komponenty tedy dostupné metody.

6.2 Základní pojmy

Definice softwarové komponenty podle [7]

Komponenta je

- neprůhledná (skrytá) implementace funkcionality
- využívána třetí stranou (third-party)
- odpovídá komponentovému modelu



Obrázek 20 – Schéma komponentově orientovaného návrhu [7]

Následující text popisuje jednotlivé očíslované části schématu zobrazeného na obrázku 20. Komponenta (1) je softwarová implementace, která může být spuštěna na fyzickém nebo logickém zařízení. Komponenta implementuje jedno nebo více rozhraní (2). Dále umožňuje spojení komponent, které nazýváme kontraktem (contract) (3). Kontrakt vyjadřuje skutečnost, kdy jedna komponenta využívá druhou pomocí rozhraní. Tyto povinnosti kontraktu zajišťují, že se nezávisle vytvořené komponenty řídí takovými pravidly, které jim umožní komunikovat a být nasazeny do standardních běhových prostředí (run-time environment) (4). Komponentový systém je založen na několika různých komponentových typech, kde každý typ zaujímá speciální roli (5) a je popsán rozhraním (2). Komponentový model (6) je soubor komponentových typů, jejich rozhraní a specifikace přípustných možností interakce mezi komponentovými typy. Komponentový rámec (dále framework) (7) poskytuje různé služby (8) na podporu a dodržování komponentového modelu. [7]

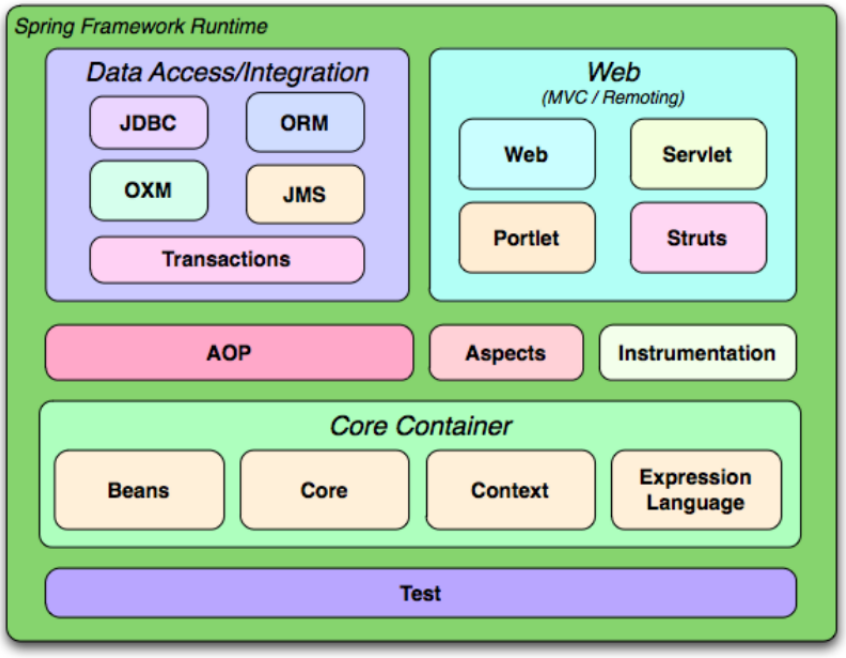
Důležitými pojmy v této problematice tedy jsou komponentový model a komponentový framework. Komponentový model zavádí konstrukční omezení na vývojáře komponent a komponentový framework navíc vynucuje dodržování těchto omezení k poskytování užitečných služeb.

6.3 Spring framework

Než si ukážeme komponentový model Spring DM, je nutné pochopit základní koncepci frameworku Spring. Na stránkách o tomto frameworku se můžeme dočíst:

> *Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications. Spring handles the infrastructure so you can focus on your application. [8]*

Tedy Spring framework je Java platforma, která poskytuje komplexní infrastrukturu podporující vývoj Java aplikací. Spring zajišťuje tuto infrastrukturu, takže vývojář se může soustředit na aplikaci. Spring je především zaměřen na vývoj a provoz podnikových a informačních systému (Java Enterprise Edition, neboli JEE aplikací). Je to modulární systém, což umožňuje použít pouze ty části, které uživatel potřebuje. Moduly se skládají ze tříd a metod a jsou zachyceny na obrázku 21. Více informací o konkrétní funkcionalitě modulů se můžete dočíst zde. [8] Pro nás důležitou vlastností jsou objekty, které jsou tvořeny frameworkem tzv. beany (beans)[4]. Z pohledu komponentového programování je bean komponenta, která implementuje alespoň jedno rozhraní. Beany jsou popsané v konfiguračním XML souboru Springu.



Obrázek 21 – Obecné schéma Spring frameworku [8]

6.3.1 Dependency injection

Důležitou vlastností Spring frameworku je *dependency injection*. Určuje, jakým způsobem se vytvářejí, konfigurují a propojují jednotlivé komponenty. Systém, který se o to stará, se nazývá lightweight *container* (lehký kontejner) nebo *IoC container* (protože toto jádro je postaveno na návrhovém vzoru Inversion of control). Odpovědnost za vytvoření a provázání je přesunuta

[4] Více o Java Beans se dočtete zde http://www.earchiv.cz/axxxk160/a706k162.php3

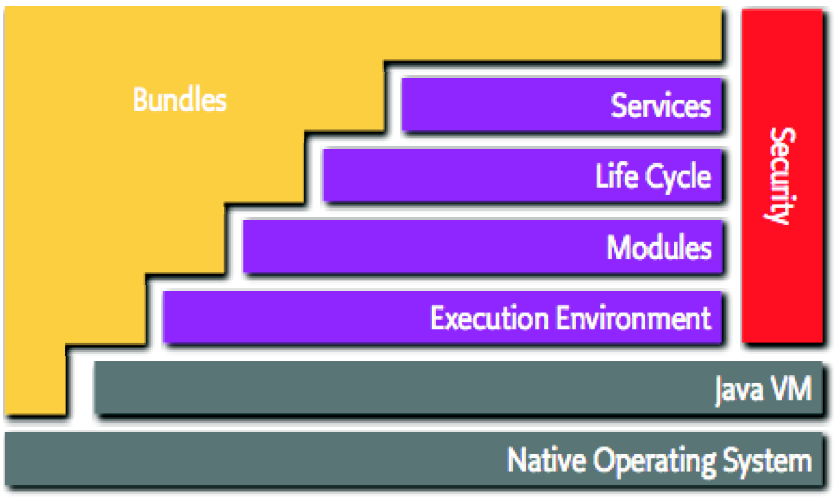
z aplikace na framework. Vývojář nevytváří reference na jiné komponenenty programově, ale reference jsou vkládány na základě XML konfiguračního souboru za běhu programu. V terminologii Springu je kontejner zmiňován jako *application context* (aplikační kontext) nebo jen *context*. Dependency injection je jednoduchý, ale poměrně výkonný model. Umožňuje psát lépe testovatelný a méně propojený kód. [9]

6.4 OSGi (Open Services Getaway initiative)

OSGi je modulární systém pro Javu, který poskytuje dynamický komponentový model. Aplikace demodulované do podoby komponent mohou být instalovány, spuštěny, zastaveny, aktualizovány a odinstalovány bez nutnosti zastavit JVM (Java Virtual Machine). Již jsme zmínili, že komponenty se v OSGi nazývají bundly. V kapitole 5.2.1 jsme rozebírali jednotlivé části v manifest souboru `MANIFEST.MF`.

Nyní popíšeme vrstvy zobrazené na obrázku 22.

- **Bundles** - Bundly jsou komponenty vytvořené programátory.
- **Services** - Vrstva služeb bundly dynamicky propojuje poskytováním modelu *publish-find-bind* (volně přeloženo zveřejni-najdi-propoj).
- **Life-Cycle** - API životního cyklu bundlů.
- **Modules** - Vrstva definuje, jak může bundle importovat a exportovat kód.
- **Security** - Tato vrstva zajišťuje bezpečnostní aspekty.
- **Execution Enviroment** - Definuje, jaké metody a třídy budou k dispozici v dané platformě. [10]

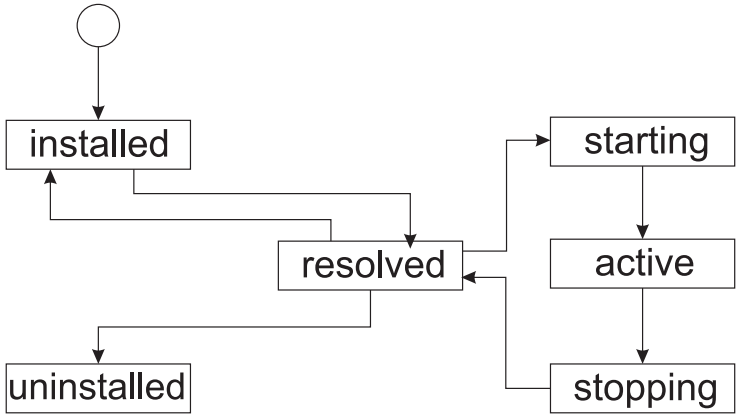


Obrázek 22 – Architektura OSGi [10]

6.4.1 Životní cyklus bundlu

Vrstva *Life-Cycle* popisuje stavy, kterými může procházet bundle obrázek 23.

- **installed** - Bundle byl úspěšně nainstalován.
- **resolved -** Bundle do tohoto stavu přechází, pokud jsou všechny závislosti uvedené v `MANIFEST.MF` uspokojeny.
- **starting -** Bundle je spouštěn.
- **active -** Bundle je aktivní (běží).
- **stopping -** Bundle se zastavuje.
- **uninstalled -** Bundle byl odinstalován a není již k dipozici.



Obrázek 23 – Životní cyklus bundlu

6.4.2 Implementace OSGi

Existuje několik rozšířených, na sobě nezávislých a volně použitelných (open source) implementací:

- Equinox - Equinox je nejrozšířenější a pro nás nejdůležitější implementací OSGi, protože je součástí vývojového prostředí Eclipse.
- Apache Felix - Apache Felix je vyvíjen Apache Software Foundation. Menší a kompaktní implementace OSGi.
- Knopferfish - Knopferfish je populární framework vyvíjen firmou Makewave.
- Concierge - Tato implementace je zvláště vhodná pro mobilní telefony nebo embedded systémy.[5].

[5] Více o embedded systémech http://www.fi.muni.cz/usr/jkucera/pv109/2005/xmrstik.htm

6.5 Spring Dynamic Modules

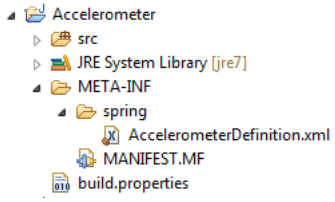
Spring DM je dynamický komponentový model využívající možností Spring frameworku a OSGi. Cílem technologie je spolupráce Spring a OSGi jednoduchým způsobem.

Samotný Spring framework má několik nevýhod. Tyto nevýhody se projevují především při vývoji velkých a složitých aplikací. Musí být nakonfigurováno velké množství beanů a takové množství se těžko udržuje. Navíc Spring trpí nedostatkem modularity a neposkytuje žádnou skutečnou podporu, jak ji zlepšit. Například zde není žádný způsob, jak omezit viditelnost beanů. Není žádoucí, aby každý bean byl vidět v kontextu celé aplikace.

Na druhé straně OSGi neposkytuje žádnou podporu vzorů nebo nástrojů v návrhu a implementaci bundlů. Volba správného návrhu je ponechána na vývojáři. Použitím Spring frameworku využijeme možností dependency injection a jiných principů.

Obě technlogie se tedy vhodně vzájemně doplňují. Abychom v aplikaci mohli využít Spring DM, musíme udělat úpravy v OSGi souboru `MANIFEST.MF` a navíc ještě přibudou konfigurační soubory Springu. Přechod od OSGi bundlu ke Spring DM bundlu bude vypadat následovně:

- V adresáři bundlu `META-INF` přibyde adresář `spring` a uvnitř něho konfigurační XML soubory, z nichž Spring vytvoří aplikační kontext.
- V souboru `META-INF/MANIFEST.MF` musí být importovány potřebné Spring bundly.



Obrázek 24 – Ukázka struktury Spring DM bundlu

Běžný OSGi bundle používá aktivační třídu `Activator` s metody `start(BundleContext context)` a `stop(BundleContext)`. Avšak při využití možností Springu tato třída již není nutná, protože se o aktivaci postará Spring DM, pouze je nutné správně nadefinovat inicializační metodu v konfiguračním XML souboru. Další nespornou výhodou je fakt, že v tomto konfiguračním souboru můžeme nadeklarovat, které služby OSGi bundle potřebuje a které poskytuje. To vše pouze za použití namespace `osgi`, které dává i další možnosti k nastavení. [9]

6.5.1 Blueprint

Na základě Spring DM modelu, OSGi aliance představila Blueprint kontejner. Samotný Spring DM vyžaduje OSGi 4.0 framework. Blueprint kontejner je součástí specifikace OSGi 4.2 a počítá s tím ve svém API. Změny oproti Spring DM jsou především v XML konfiguračním souboru, více naleznete zde[11].

6.6 Simco

Simco je framework vytvořený studenty Západočeské univerzity v Plzni pro simulační testování komponent. Jádro bylo vytvořeno Tomášem Kabíčkem [12] a grafické rozhraní Matějem Prokopem [13]. V tomto frameworku nejsou testovány modely komponent, ale komponenty samotné. Název Simco vzniknul zkrácením anglického spojení *SIMulation of COmponent.* Aplikace, která je na tento framework napojena, je nazývána *Simco aplikace.* Simco je založeno na zmiňovaném komponentovém frameworku Spring DM.

6.6.1 Jádro

Jádro Simca umožňuje kontrolu nad celou Simco aplikací. Princip celého fungování spočívá v tom, že každý krok, který se v Simco aplikaci provede (ať už se zavolá metoda některé komponenty, nebo se provede návrat z volané metody), bude spojen s událostí v Simco frameworku. Tyto události musí být vkládány do kalendáře, kde budou všechny uloženy a kde dochází k manipulaci s nimi. Žádná akce se v Simco aplikaci neprovede, dokud není Simco frameworkem vyvolána příslušná událost. Ke každé události je přiřazen podprogram, který se provede při spuštění dané události. Používá se princip diskrétní událostní simulace. [12]

Jádro Simco frameworku se tedy skládá z několika základních objektů:

- komponenty - testované aplikace (Spring DM bundly),
- událost - objekt reprezentuje událost simulace běhu Simco aplikace,
- kalendář - obsahuje události, které se budou postupně zpracovávat, a tím zajišťuje kontrolu nad během celé Simco aplikace,
- simulace - objekt manipulující s celou simulací.

Komponenty Simco pracuje s několika typy komponent:

- simulované (`SIMULATION`) - běh těchto komponent je pouze nasimulovaný,
- reálné (`REAL`) - tyto komponenty provádí reálný výpočet,
- prostředníky (`INTERMIDIATE`)- prostředník reálné komponenty,
- framework (`FRAMEWORK`)- implementace Simco frameworku.

Běh simulované komponenty není založen na reálném výpočtu, ale pouze simuluje reálný systém, který by komponenta představovala. Hodnoty, které simulovaná komponenta vrací, a doba výpočtu jsou uloženy v konfiguračním souboru XML.

Reálné komponenty provádí reálný výpočet. Tyto komponenty nesmí vědět o Simco frameworku, aby mohly být kdekoliv použitelné bez jakýkoliv vnitřních změnit. Další komponentou je tedy `INTERMIDIATE`, která zajišťuje komunikaci mezi reálnou komponentou a simulovanou.

Posledním typem komponenty je komponenta `FRAMEWORK`, která obsahuje implementaci Simco frameworku. [12]

Události Každá událost bude přirazena k nějaké činnosti Simco aplikace. Pokud probíhá komunikace, Simco framework ji zachytí, vytvoří událost a vloží jí do kalendáře. Vložená událost může způsobit výskyt dalších událostí. [12]

Typy událostí:

- `REAL_CALL` - Reálná metoda komunikuje s některou z jiných komponent Simco aplikace.
- `REAL_RETURN` - Návrat z volané metody reálné komponenty.
- `SIMULATION_CALL` - Simulační metoda komunikuje s některou z jiných komponent Simco aplikace.
- `SIMULATION_RETURN` - Návrat z volané metody simulační komponenty.
- `REGULAR` - Jedná se o událost vyskytující se v pravidelných intervalech.
- `CASUAL` - Jedná se o událost, u které bude možno nadefinovat, s jakou pravděpodobností se bude vyskytovat.

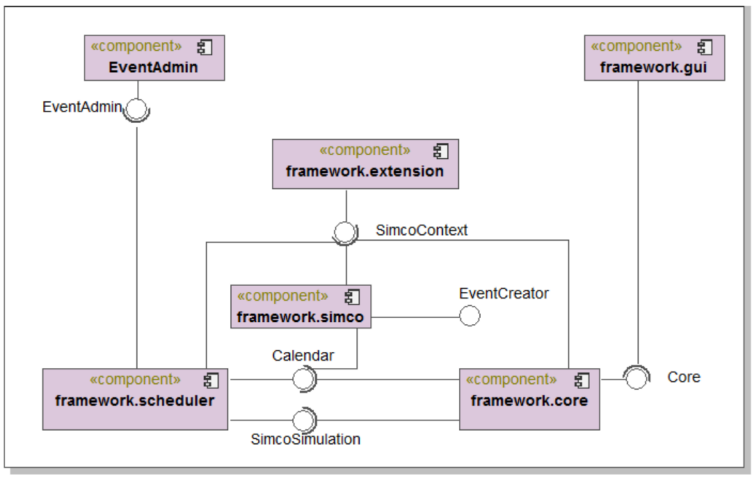
Implementace Celý systém dodržuje zásady komponentového programování - komponenty jsou „black box", tedy nevědí, jakou funkčnost implementují, ale vidí, co ostatní poskytují a obvykle sami něco poskytují. Jádro se skládá ze čtyř Spring DM bundlů:

- framework.scheduler
  - bundle, který zajišťuje implementaci diskrétní událostní simulace.
  - zajišťuje funkčnost kalendáře, běhu simulace a ukládání historie jejího průběhu.
- framework.core
  - bundle, který obstarává komunikaci vnějších modulů s jádrem Simco frameworku.
  - obsahuje všechny funkce potřebné pro ovládání a vizualizaci činnosti Simco frameworku - na něj je napojeno uživatelské rozhraní viz kapitola6.6.2.
- framework.simco
  - bundle, který zprostředkovová spojení Simco frameworku se Simco aplikací.
  - funkčnosti využívají Simco aplikace, které jsou přímo napojeny na Simco framework, tedy komponenty `SIMULATION` a `INTERMEDIATE`.
- framework.extension

– bundle, který poskytuje data ostatním bundlům.

– obstarává práci s XML soubory a obsahuje datové třídy, které využívá zbytek Simco frameworku.[12]

Propojení jednotlivých komponent je znázorněno na UML diagramu na obrázku 25.



Obrázek 25 – UML komponentový diagram Simco frameworku [12]

Konfigurační XML soubor scénáře Tento konfigurační soubor je důležitou součástí celého Simco frameworku. Vytváří postup celého průběhu simulace, proto se mu říká scénář (scenario).

Ve scénáři je možno nadefinovat:

- seznam komponent Simco aplikace a jejich vlastnosti,
- události typu `CASUAL` a `REGULAR`,
- ostatní nastavení Simco aplikace a Simco frameworku.

Kořenový element XML souboru má název `simCoScenario`. Seznam komponent Simco aplikace je definován v elementu `components`. Jednu komponentu definuje element `component`, který má atribut type. Tento atribut může mít hodnoty `SIMULATION`, `REAL`, `INTERMEDIATE`, `FRAMEWORK` – podle toho, o jakou komponentu se jedná. Element komponent má dále vnořené prvky `symbolicName` a `settingsFile`. `SymbolicName` určuje jedinečné jméno komponenty (shodné s OSGi atributem symbolicName) a `settingsFile` určuje cestu ke konfiguračnímu XML souboru dané komponenty. [12]

```
<component type="REAL">
  <symbolicName>NameOfComponent</symbolicName>
  <settingsFile>D:\pathToComponent\xmlFile.xml</settingsFile>
</component>
```

Výpis 4 – Definice komponenty v konfiguračním XML scénáři

Dále je možno nadefinovat události typu `REGULAR` a `CASUAL`. Tím lze do jisté míry určit, jak se bude Simco aplikace chovat. U těchto událostí se definuje především to, jak často se mají provádět, a jaké parametry má mít metoda, kterou daná událost volá. Všechny události se ve scénáři definují v elementu `events`. Analogicky k definici komponent je jedna konkrétní událost definovaná v elementu `event`, který má atribut type s hodnotou buď `REGULAR`, nebo `CASUAL`, podle typu události. Oba typy událostí mají element `source`, `serviceName`, `methodName`, `methodParameters` a `eventDetail`. Každá událost představuje volání některé OSGi služby. Element `source` určuje, která komponenta má volání provést. Hodnota elementu je `symbolicName` dané komponenty. Musí se jednat o simulační komponentu. Vlákno této simulační komponenty volání provede. Element `serviceName` určuje jméno rozhraní, přes které je volaná služba registrována. `MethodName` definuje jméno volané metody dané služby a v elementu `methodParameters` jsou nadefinovány parametry volané metody.

Jediné, čím se oba typy událostí liší, je element `eventDetails`. V něm je nadefinováno, jak často se má daná událost opakovat. U události typu `REGULAR` se pouze nastaví perioda opakování události. Definice `REGULAR` události je ve výpisu 5. Událost v tomto výpisu se bude opakovat po deseti krocích, respektive za deset jednotek simulačního času (což v případě, že na nějaký čas není naplánovaná žádná událost, neznamená deset kroků). [13]

```
<event type="REGULAR">
  <source>NameOfComponent</source>
  <serviceName>simco.application.NameOfComponent.InterfaceOfComponent</
      serviceName>
  <methodName>setMethod</methodName>
  <methodParameters>
    <parameter dataType="java.lang.Integer" value="12" />
    <parameter dataType="java.lang.Integer" value="12" />
    <parameter dataType="java.lang.Integer" value="12" />
  </methodParameters>
  <eventDetails>
    <detail key="period" val="10" />
  </eventDetails>
</event>
```

Výpis 5 – Ukázka definice události REGULAR v konfiguračním XML scénáři

Událost typu `CASUAL` představuje větší možnosti ohledně definice výskytu. Používá se zde náhodná veličina generovaná s exponenciálním (`exponencial`) nebo gaussovským (`gauss`)

pravděpodobnostním rozdělením. U exponenciálního rozdělení stačí v elementu `eventDetails` zadat střední hodnotu (`mean`), u gaussovského rozdělení je nutno zadat kromě střední hodnoty také směrodatnou odchylku (`standardDeviation`). [12]

Konfigurační XML soubor komponenty Tyto XML soubory obsahují nastavení dané komponenty. Lze v nich nadefinovat, jak dlouho bude trvat simulace výpočtu jednotlivých metod OSGi služeb dané komponenty a jaké hodnoty mají dané metody vracet. Definují se v nich také třídy, které komponenta obsahuje. Stačí definovat pouze třídy implementující rozhraní, přes které je registrována některá OSGi služba dané komponenty.

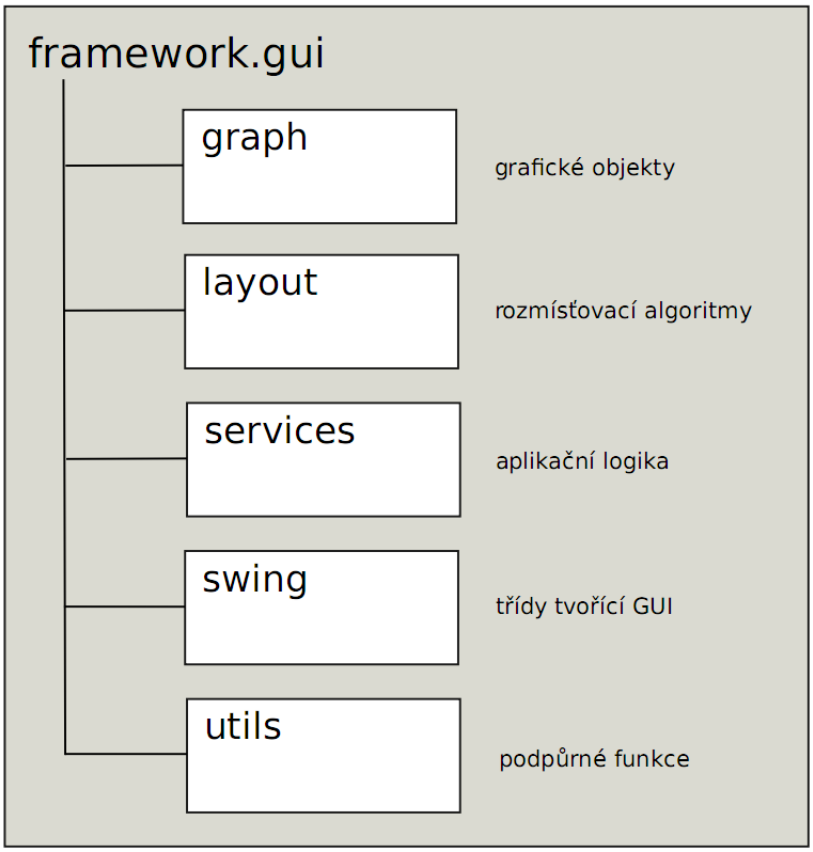
Tyto soubory musí být k dispozici ke každé reálné a simulační komponentě Simco aplikace. Kořenový element tohoto konfiguračního souboru má název `componentSettings`. Definice tříd se provádí v elementu `classes`, kde každou třídu reprezentuje element `class`. Všechny možnosti definice konfiguračního souboru komponenty jsou vidět ve výpise 6. Element `calculationTime` určuje, jak dlouho má metoda simulovat výsledek, a return určuje, jakou hodnotu má metoda vracet. [12]

```
1  <componentSettings name="NameOfCompoment">
2    <classes>
3      <class name="simco.application.simco.NameOfCompoment.NameOfClass">
4        <methodSettings>
5          <method name="getMethod">
6            <calculationTime value="3" />
7            <return dataType="java.lang.Integer" value="8" />
8          </method>
9        </methodSettings>
10     </class>
11   </classes>
12 </componentSettings>
```

Výpis 6 – Ukázka konfiguračního XML souboru komponenty

6.6.2 Grafické rozhraní

Grafické rozhraní zabyvající se vizualizací komponent je implementováno jako další Spring DM bundle *framework.gui* a je napojeno na jádro aplikace, jak je vidět na obrázku 25. Jednotlivé balíky tohoto bundlu jsou znázorněny na obrázku 27. K vizualizaci komponent využívá toto grafické prostředí knihovnu *Zest*. Zest je sada vizualizačních nástrojů obsahujících komponenty vytvořené pro IDE Eclipse. Tato knihovna využívá rozmísťovací algoritmus objektů *Spring layout*. Tento algoritmus je podrobněji popsán v [13].



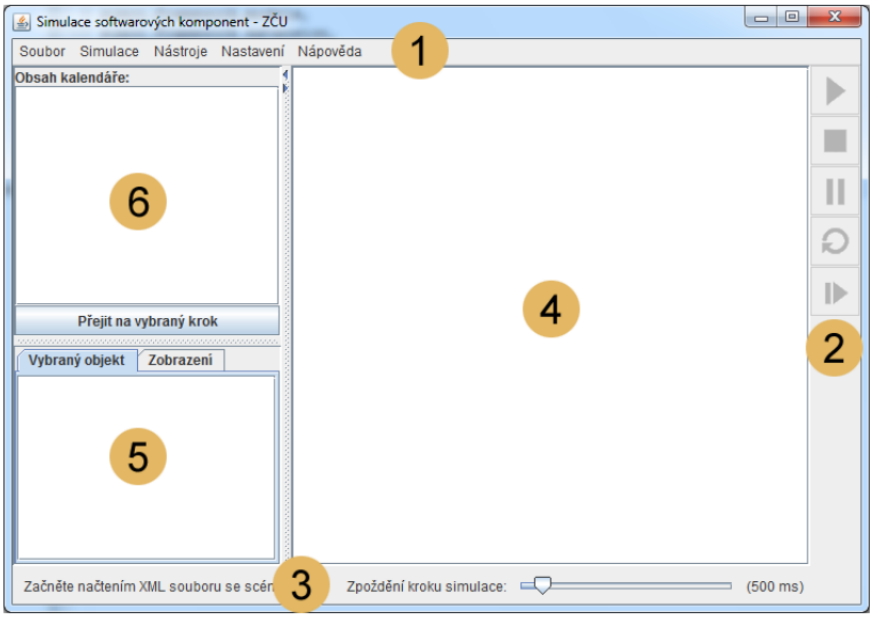
Obrázek 26 – Balíky grafické komponenty frameworku Simco [13]

Napojení na Simco jádro Funkce pro nastavení a ovládání simulace jsou poskytovány simulačním jádrem skrze OSGi službu, které je definována prostřednictvím rozhraní s názvem `ICore`. Prostřednictvím tohoto rozhraní je možné nastavovat simulační scénář, ve kterém jsou uloženy informace o simulovaných komponentách a předdefinovaných událostech. Jsou zde funkce pro ovládání simulace, tj. pro její spuštění, zastavení a krokování. [13]

Části uživatelského rozhraní Uživatelské prostředí je tvořeno jedním hlavním oknem, které je logicky rozděleno na šest částí. Rozdělení hlavního okna na jednotlivé části je zachyceno na obrázku 27.

První označenou částí na obrázku je hlavní menu, které zprostředkovává funkce poskytované prostředím. Jednotlivé funkce jsou popisovány dále. Částí označenou číslem 2 je nástrojový panel obsahující tlačítka pro ovládání stavu simulace. Pod číslem 3 je stavový panel zobrazující informace o aktuálním stavu simulace. Částí číslo 4 je kreslící plátno, na které jsou zobrazovány simulované komponenty, jejich rozhraní a vazby mezi nimi. Pod číslem 5 se skrývá panel se dvěma záložkami. Záložka „Vybraný objekt" slouží k zobrazování informací o vybraném objektu na kreslícím plátně. Záložka „Zobrazení" slouží k nastavení stylu zobrazení objektů na kreslí-

cím plátně. Poslední, šestou částí je kalendář událostí, který zobrazuje události probíhající v simulaci. [13]



Obrázek 27 – Hlavní okno grafického prostředí Simco [13]

7 Analýza

Protože aplikace, kterou budeme tvořit, je Eclipse plugin pro framework Simco, budeme jí nazývat *Simco plugin* (dále jen plugin). V této kapitole budou popsány požadavky na tento plugin. Dále budou nastíněny možnosti řešení požadavků.

7.1 Specifikace požadavků na aplikaci

Naším úkolem je provést takové změny ve frameworku Simco, aby jsme mohli vytvořit plugin, který s ním umožní snadnější práci tj. vytvořit celou aplikaci uživatelsky přívětivější a lehce integrovatelnou do Eclipse. Vzhledem k tomu, že celý framework je komplexní a dobře fungující, budeme se snažit, aby zásahy do programového kódu jednotlivých bundlů byly co nejmenší. Budeme se soustředit na přínosy, které můžeme získat s propojením platformy Eclipse a frameworku Simco.

7.1.1 Import komponent

Jistě užitečným vylepšením je možnost importu komponent, které Simco testuje. Momentální situace je taková, že pokud chce uživatel importovat komponenty do Eclipse, musí je importovat jednu po druhé. Simco plugin by toto generování měl zajistit hromadně.

7.1.2 Generování konfiguračního XML souboru scénáře

Konfigurace XML scénáře se skládá z několika částí a jeho strukturu jsme popsali v kapitole 6.6.1. Nás bude zajímat především jeho první část, která nese informaci o názvu komponenty, jejím typu a cestě ke konfiguračnímu XML souboru komponenty. Pokud nám uživatel při importu komponent definuje, která komponenta reprezentuje jaký typ, máme všechny potřebné informace, abychom tento XML soubor vygenerovali.

7.1.3 Generování kostry konfiguračního XML souboru komponenty

Každá z reálných a simulovaných testovaných komponent v Simco frameworku obsahuje XML konfigurační soubor, který obsahuje různé nastavení dané komponenty. U každého konfiguračního souboru je vždy nutné uvést třídu (třídy) implementující rozhraní, přes které je registrována některá OSGi služba dané komponenty. Bylo by vhodné umožnit generovat kostru takového souboru.

7.1.4 Spouštění frameworku Simco

Momentální jedinou možností, jak zprovoznit celé Simco a spustit ho i se soubory, které mají být testovány, je importovat jednotlivé bundly do workspace Eclipse a samozřejmě splnit všechny závislosti. Uživatel si tímto krokem vytvoří ve workspace změť několika různých projektů. Vzhledem k faktu, že z uživatelského hlediska v programovém kódu frameworku není potřeba

nic měnit, bylo by vhodné, aby jednotlivé bundly nebyly ve workspace, ale byly uživateli skryté (součástí platformy Eclipse). Dalším možným vylepšením může být vytvoření konfigurace při spuštění, kde si uživatel vybere konfigurační XML scénář, který se následně v Simco spustí. Nyní Simco při spouštění pouze načítá konfigurační soubor, který nastavuje především vzhled aplikace - velikost písma, barvy, jazykovou lokalizaci, apod.

7.1.5 Grafické prostředí

Důležitou součástí z uživatelského hlediska je grafické rozhraní umožňující práci s frameworkem Simco. Zde se nabízí možnost umístit hlavní okno Simca do prostoru okna editoru Eclipse. Editor v Eclipse, má speciální vlasnost, že musí být navázán na nějaký typ souboru. V našem případě můžeme navázat grafické prostředí na XML soubory. Zároveň ale nechceme ztratit možnost tyto XML soubory prohlížet a editovat. Mohli bychom tedy vytvořit multi editor, kde by jedna ze záložek zobrazovala GUI a další umožňovala nativní zobrazení a editaci XML. Grafické prostředí je naprogramováno s využitím prvků z knihoven Swing. Výhodou Swingu je rozsáhlé spektrum použitelných komponent, pokrývající požadavky na prakticky jakékoli uživatelské rozhraní. Swing je nezávislý na platformě a díky tomu vzhled swing aplikací je na všech platformách stejný. Naopak grafické prostředí Eclipse používá prvky z knihovny SWT. SWT je jako Swing naprogramován v jazyce Java, ale zároveň využívá nativní knihovny operačních systémů. SWT je díky tomu rychlejší, ale poskytuje méně komponent a vzhled aplikace se mění v závislosti na operačním systému, na němž běží. Pokud bychom tedy chtěli GUI Simco integrovat do editoru v Eclipse, museli bychom programový kód aplikace přepsat a nemáme zaručeno, že se nám podaří pro každou komponentu ze Swing knihovny najít odpovídající komponentu z knihovny SWT. Pro ilustraci problému uvádím tabulku (1), která ukazuje některé komponety Swingu a jejich ekvivalenty v SWT.

Swing	SWT
JButton	Button (Style=SWT.PUSH)
JCheckBox	Button (Style=SWT.CHECK)
JCheckBoxMenuItem	MenuItem (Style=SWT.CHECK)
JColorChooser	ColorDialog
JComboBox	Combo or CCombo
JDesktopPane	No equivalent in SWT; use GEF if needed
JEditorPane	StyledText
JFileChooser	FileDialog or DirectoryDialog
JInternalFrame	No equivalent in SWT; use GEF if needed
JLabel	Label or CLabel
JLayeredPane	No equivalent in SWT; use GEF if needed
JList	List
JMenu	Menu, or MenuItem (Style=SWT.CASCADE) if in a menu
JMenuBar	Menu (Style=SWT.BAR)
JMenuItem	MenuItem (Style=SWT.PUSH)
JOptionPane	MessageDialog or InputDialog
JPanel	Composite or Group
JPasswordField	Text (Style=SWT.SINGLE); use setEchoChar(char)
JPopupMenu	Menu (Style=SWT.POP_UP)
JProgressBar	ProgressBar, ProgressIndicator, or ProgressMonitorDialog
JRadioButton	Button (Style=SWT.RADIO)

Tabulka 1 – Komponenty Swing a jejich ekvivalenty v SWT [14]

7.2 Možnosti řešení

Rozbereme možnosti, které nám nabízí samotný Eclipse a vývoj pluginů pro implementaci daných požadavků. Úkolem je integrovat veškerou funkčnost do Eclipse tak, aby ji uživatel nalezl na místě, kde ji očekává, a využíval ji jemu v tomto prostředí obvyklým způsobem.

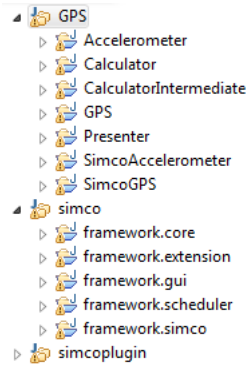
7.2.1 Import komponent

Budeme předpokládat, že komponenty budou v jednom adresáři a budou již připravené. Připraveností je myšleno, že se bude jednat o již hotové projekty (komponenty) přichystané pro Simco. Nabízí se hned několik možností, jak uchovat komponenty pohromadě:

- Projekt s komponenty
- *Working set* s komponenty
- Workspace s komponenty

Projekt s komponenty Na první pohled se může zdát toto řešení jako správné. Ve stromové struktuře bychom mohli procházet komponenty a dělat případné úpravy. Vše by bylo pohromadě v jednom projektu. Toto řešení nebude fungovat, protože Eclipse neumožňuje vytvořit projekt s podprojekty. Mohli bychom sice v souborovém systému projekty překopírovat do adresáře projektu ve workspace, a dokonce bychom mohli tyto soubory libovolně procházet a měnit, avšak nebudeme schopni projekty spustit. Toto řešení je tedy nevhodné.

WorkingSet s komponenty Dalším nabízejícím se řešení je vytvoření *working setu*. Working sety pod sebou zdržují libovolný počet elementů. Elementy mohou být různé datové soubory, ale typicky jsou to projekty. Working set slouží pouze jen pro přehlednost ve workspace (obrázek 28), který obsahuje větší množství elementů, ale v adresářové struktuře operačního systému se nic nezmění. Očividně je vhodné združovat projekty, kterém spolu nějakým způsobem souvisí. Tento předpoklad splňují i komponenty, které budeme importovat.



Obrázek 28 – Ukázka zobrazení workspace s working sety v Eclipse

Workspace s komponenty Také je zde možnost vytvořit nový workspace a do něj přenést komponenty. Z pohledu Eclipse není problématické používat více workspace a přepínat mezi nimi. Při testování komponent nebudeme potřebovat jiné projekty, než komponenty samotné. Vlastní workspace tedy skýtá výhodu nejen logického oddělení, jako v případě working setů,

ale i oddělení fyzického, tedy na úrovni souborového systému. Nesnáze a citelné zpomalení prostředí Eclipse, které je zaplněno větším množstvím projektů, je další fakt, který nás utvrzuje ve správnosti použití odděleného workspace. Proto budeme implementovat toto řešení.

7.2.2 Generování XML souborů

Generování obou konfiguračních XML souborů (scénář, nastavení komponenty) zajistíme stejným způsobem. Eclipse k vytváření veškerých zdrojů (projektů, souborů apod.) používá tzv. *wizardy*. Wizard je průvodce pomáhající uživateli nastavit všechny potřebné informace. Budeme se držet zažitých konvencí Eclipse a ke generování XML souborů použijeme wizardy.

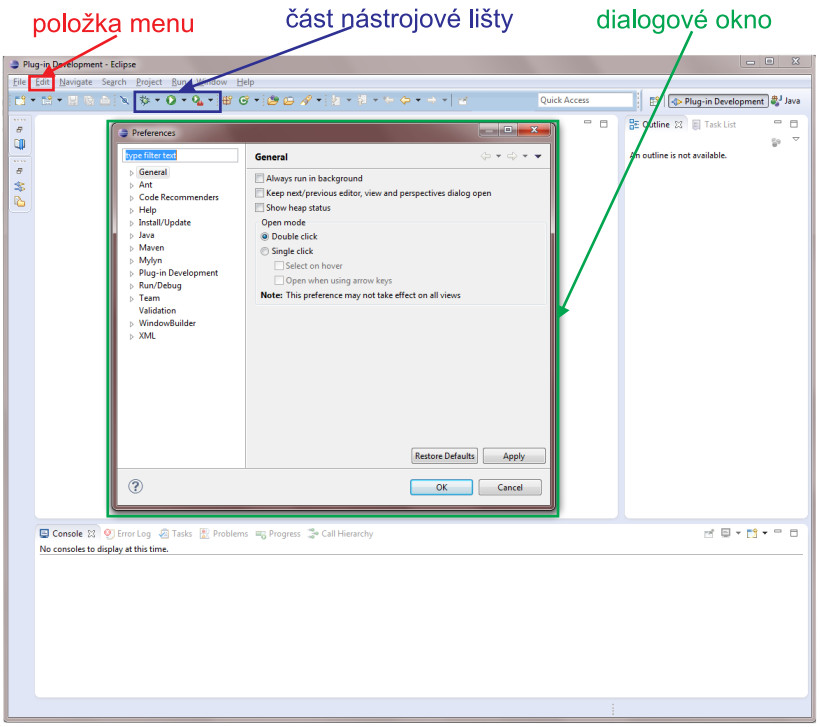
7.2.3 Simco projekt

Abychom odlišili spouštění frameworku od ostatních aplikací a umožnili zvláštní konfiguraci, před spuštěním bude vhodné vytvořit nový typ projektu. U tohoto projektu budeme moci vytvořit vlastní spouštěcí konfiguraci (*run configuration*). Vytváření projektu budeme také realizovat speciálním wizardem. Během tvorby tohoto projektu můžeme navíc uživateli umožnit import komponent do Eclipse a vygenerování kostry XML scénáře.

7.2.4 Integrace do grafického prostředí Eclipse

Eclipse je rozmanité vývojové prostředí a nabízí se několik možností, kam umístit ovládací prvky vytvářeného pluginu (viz obrázek 29):

- položka hlavního menu,
- dialogové okno,
- část nástrojové lišty.



Obrázek 29 – Možnosti grafické integrace Simco pluginu do Eclipse

Položka hlavního menu Všechny funkčnosti pluginu by mohly být dostupné pod jednou položkou hlavního menu. Když se však podíváme na stávající položky menu, zjistíme, že se týkají celého prostředí Eclipse a konfigurování jeho obecných funkčností. Vytvoření položky hlavního menu pro jeden specifický plugin, jako je Simco plugin, by bylo z hlediska grafického rozhraní Eclipse nelogické.

Dialogové okno Pro plugin bychom mohli vytvořit dialogové okno, které by bylo vyvoláváno po nějaké akci. Takovou akcí by mohlo být např. tlačítko v nástrojové liště. Na obrázku 29 vidíme ukázku dialogového okna *Preferences* (Nastavení). Dialogová okna v prostředí Eclipse nabízí většinou bohaté možnosti nastavení, což u našeho pluginu nebude potřeba. Navíc tyto funkce nejsou dostupné „přímo“, uživatel by musel nejdříve nějakou akcí otevřít dialogové okno a poté provést další akci pro spuštění některé z funkcí pluginu.

Část nástrojové lišty Položky v nástrojové liště (*toolbar*) se shlukují do větších částí, které spolu souvisejí. Příkladem je souhrn položek *Launch* (Spuštění) z obrázku 29, který shlukuje položky nejen na běžné spuštění aplikace, ale také na spouštění v ladícím módu (*Debug*), apod. Simco plugin by tedy mohl být částí nástrojové lišty. Velkou výhodou je jeho jednoduchá uživatelská dostupnost.

Navíc, pokud plugin není používán, lze příslušnou nástrojovou lištu jednoduše skrýt postupem **Window** > **Customize Perspective,** přechodem na záložku **Toolbar visibility** a zde zrušením výběru položky, která nemá být v dané perspektivě vidět**.**

7.2.5 Shrnutí

Budeme implementovat plugin, který bude mít vlastní typ projektu a umožní importovat komponenty připravené pro Simco framework. Dále bude umožněno generovat kostry XML konfiguračních souborů díky wizardům. Veškerá funkčnost bude dostupná z nástrojové lišty Eclipse, kromě spouštěcí konfigurace, která bude přístupná v dialogovém okně Run configurations, jak je uživatel Eclipse prostředí zvyklý.

8 Implementace

Z předcházející analýzy jsou jasné hlavní prvky aplikace, které budeme implementovat. V této kapitole popíšeme samotnou implementaci Simco pluginu.

8.1 Struktura aplikace

Aplikační kód programu je rozdělen do několika balíků viz tabulka 2. Strukturu aplikace tvoří nejen programový kód, ale důležitou součástí jsou soubory plugin manifestu.

Balík	Základní funkčnost
simcoplugin	Obsahuje aktivační třídu pluginu.
simcoplugin.handlers	Obsahuje handlery obsluhující commandy aplikace.
simcoplugin.launcher	Launcher pro Simco projekt.
simcoplugin.natures	Nature Simco projektu.
simcoplugin.projects	Podpora pro importování projektů do workspace.
simcoplugin.tabs	Záložky Run configurations pro Simco projekt.
simcoplugin.wizards	Implementace všech wizardů a jejich stránek.
simcoplugin.generationXML	Generování konfiguračního XML souboru.

Tabulka 2 – Rozdělení aplikace do balíků s jejich základní funkčností

8.2 Soubory plugin manifestu

V následujícím textu si popíšeme a ukážeme zajímavé části souboru `plugin.xml`, který rozšiřuje funkčnosti, jejichž chování je naprogramováno v programovém kódu. Každý popis obsahuje i ukázku, ke které se text vztahuje. Většina elementů obsahuje atribut `id`, který reprezentuje jedinečný identifikátor elementu, a `name` který představuje jeho jméno.

8.2.1 Konfigurační soubory

V pluginu pracujeme se dvěma typy XML souborů. Abychom tyto XML soubory odlišili od jiných, definujeme jejich *content-type* (typ obsahu).
Využijeme tedy extension point `org.eclipse.core.contenttype.contenTypes` a vytvoříme nový content type, jehož základ bude vycházet z `org.eclipse.core.runtime.xml`, který definujeme atributem `base-type`. Také nastavíme příponu souboru v atributu `file-extesions` na `xml`. Když máme nadefinovaný content-type, musíme určit, které soubory jsou tímto typem

určeny. Musíme je odlišit od jiných XML souborů. K tomu poslouží element `describer` a třída `org.eclipse.core.runtime.content.XMLRootElementContentDescriber2`, která slouží k detekci nejvyšších elementu XML dokumentu. Elementem `parameter` nadefinujeme hodnotu kořenového elementu na `componentSettings`. Stejným způsobem realizujeme content type pro XML soubor scénáře, pouze změníme hodnotu kořenového adresáře na `simCoScenario`.

```
<extension
          id="simcoplugin.contenttype"
          point="org.eclipse.core.contenttype.contentTypes">
  <content-type
       base-type="org.eclipse.core.runtime.xml"
             file-extensions="xml"
             id="simcoplugin.contenttype.compSettings"
             name="Component Settings XML"
             priority="normal">
          <describer
                class="org.eclipse.core.runtime.content.
                    XMLRootElementContentDescriber2">
             <parameter
                   name="element"
                   value="componentSettings">
             </parameter>
          </describer>
  </content-type>
</extension>
```

Výpis 7 – Definice content type

8.2.2 Simco projekt

Vytvoříme typ projektu pluginu, který pojmenujeme *Simco projekt*. Pro práci s novým typem projektu musíme definovat *project nature* Simco projektu rozšířením `org.eclipse.core.resources.natures`. Project nature formuluje charakter projektu. V elementu `run` nastavíme třídu `simcoplugin.natures.ProjectNature`, kde je uloženo ID Simco projektu. Dále rozšířením `org.eclipse.ui.ide.projectNatureImages` v elementu `image` atributu `icon` nastavíme cestu k ikoně reprezentující Simco projekt v Eclipse a nastavením `natureID` tuto ikonu propojíme se Simco projektem.

```
<extension
          id="simcoplugin.projectNature"
          point="org.eclipse.core.resources.natures">
       <runtime>
          <run
             class="simcoplugin.natures.ProjectNature">
          </run>
       </runtime>
</extension>
<extension
          id="simcoplugin.projectNatureImage"
          point="org.eclipse.ui.ide.projectNatureImages">
       <image
             icon="icons/Component.ico"
             id="simcoplugin.natureImage"
             natureId="simcoplugin.projectNature">
       </image>
</extension>
```

Výpis 8 – Definice project nature

8.2.3 Průvodci (Wizards)

Užitím extension point `org.eclipse.ui.newWizards` nejdříve vytvoříme novou kategorii wizardů, což umožní sjednotnit přístup k nim z prostředí Eclipse. Poté vytvoříme samotné wizardy. V ukázce 9 je definován wizard, který vytváří nový (Simco) projekt, (proto je atribut `project` nastaven na `true`) jehož implementace je umístěna ve třídě `simcoplugin.wizards.SimcoProjectNewWizard`. Dále je nastavena cesta k ikoně wizardu v atributu `icon` a ikona u popisu funkce wizardu v atributu `descriptionImage`. Podobným způsobem vytvoříme i wizardy pro generování konfiguračních XML souborů a zařadíme je do stejné kategorie nastavením atributu `category` na `simcoplugin.category.wizards`.

```
<extension
          id="simcoplugin.wizards"
          point="org.eclipse.ui.newWizards">
       <category
             id="simcoplugin.category.wizards"
             name="Simco Wizards">
       </category>
       <wizard
             category="simcoplugin.category.wizards"
             class="simcoplugin.wizards.SimcoProjectNewWizard"
             descriptionImage="icons/About.ico"
             icon="icons/Component.ico"
             id="simcoplugin.wizard.new.simco"
             name="Simco Project"
             project="true">
       </wizard>
</extension>
```

Výpis 9 – Definice wizardu

8.2.4 Příkazy a jejich obsloužení (Commands and handlers)

Pro některé úkony nepotřebujeme vytvářet wizard, ale chceme pouze provést jednorázovou akci. Takovou potřebnou akcí v Simco plugin je například přepnutí do jiného workspace a v terminologii eclipse pluginů se tato akce jmenuje *command* (příkaz). Příkaz vytvoříme rozšířením `org.eclipse.ui.commands`, abychom mohli nastavit vlastní chování commandu. Musíme ještě vyrobit *handler* (manipulátor) rozšířením `org.eclipse.ui.handlers`. V elementu `handler` nastavíme atribut `commandId` na id commandu v našem případě `simcoplugin.commands.workspaceCommand`, čímž propojíme command s handlerem. Dále v handleru v atributu `class` nastavíme třídu, která bude implementovat chování commandu - `simcoplugin.handlers.SwitchWorkspaceHandler`. Protože máme v aplikaci více commandů, vytvořili jsme podobně jako u wizardů jejich kategorii `simcoplugin.commands.category` definovanou v elementu `command` atributem `category`.

```
1  <extension
2          id="simcoplugin.commands"
3          point="org.eclipse.ui.commands">
4       <category
5             id="simcoplugin.commands.category"
6             name="Simco Commands">
7       </category>
8       <command
9             categoryId="simcoplugin.commands.category"
10            description="Refresh workspaces projects"
11            id="simcoplugin.commands.workspaceCommand"
12            name="Refresh Command">
13      </command>
14 </extension>
15 <extension
16         id="simcoplugin.handlers"
17         point="org.eclipse.ui.handlers">
18    <handler
19            class="simcoplugin.handlers.SwitchWorkspaceHandler"
20            commandId="simcoplugin.commands.workspaceCommand">
21      </handler>
22 </extension>
```

Výpis 10 – Definice commandu s handlerem

Uživateli dáme ještě možnost daný command spustit klávesovou kombinací. K tomu slouží *binding* (svazek), využijeme tedy extension point `org.eclipse.ui.bindings`. V elementu `key` nastavíme důležité atributy.

- `ContextId` je indefikátor kontextu, ve kterém je binding aktivní. Nastavíme kontext na `org.eclipse.ui.contexts.window`, to znamená, že binding bude aktivní v každém hlavním Eclipse okně.
- `SchemeId` je indefikátor schématu, kdy je binding aktivní. Nastavíme schéma na `org.eclipse.ui.defaultAcceleratorConfiguration`, to znamená, že binding bude aktivní, pokud bude nastaveno výchozí schéma. Pokud bychom vytvořili vlastní schéma, binding by byl aktivní, jen ve chvíli, kdy by bylo toto schéma použito (možno měnit v nastavení Eclipse a v položce **Keys**).
- `Sequence` představuje klávesovou sekvenci. Nastavíme na `M1+M3+5`. „M“ značí modifikátor nezávislý na platformě. Na operačním systému Windows tato kombinace značí ctrl+alt+5.[6]

[6] Podrobnější informace o všech možnostech nastavení sekvencí naleznete v nápovědě Eclipse pro extension point `org.eclipse.ui.bindings`.

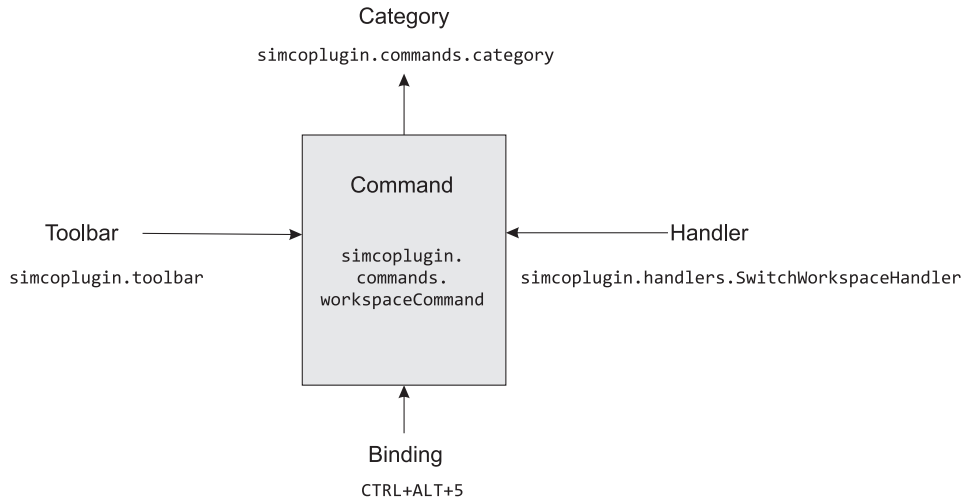
```
<extension
          id="simcoplugin.bindings"
          point="org.eclipse.ui.bindings">
       <key
             commandId="simcoplugin.commands.workspaceCommand"
             contextId="org.eclipse.ui.contexts.window"
             schemeId="org.eclipse.ui.defaultAcceleratorConfiguration"
             sequence="M1+M3+5">
      </key>
</extension>
```

Výpis 11 – Definice bindingu

Na obrázku 30 je znázorněno propojení commandu pro změnu workspace s následujícími vlastnostmi a částmi:

- command přirazen ke kategorii (**category**) `simcoplugin.commands.category`,
- **binding** s nakonfigurovanou kombinací CTRL+ALT+5,
- **handler** `simcoplugin.handelrsSwitchWorkspaceHandler`, který implementuje chování commandu.

V kapitole 8.2.6 zasadíme vytvořený command do nástrojové lišty Eclipse.



Obrázek 30 – Propojení commandu s handlerem, bindingem a toolbarem

8.2.5 Konfigurace spouštění (Run configurations) Simco projekt

Podobně jako jsme vytvořili content type pro soubor vytvoříme nový *launch configuration type* (typ konfigurace spuštění) pro plugin. Rozšíříme `org.eclipse.debug.core.launchConfigurationTypes` a delegujeme spuštění na *launcher*

(spouštěč) implementovaný ve třídě `simcoplugin.launcher.Launcher`. Mód je nastaven na `run`, tedy implementovaný launcher umožňuje spuštění projektu. Další možností je debug pro ladění.

Rozšířením `org.eclipse.debug.ui.launchConfigurationTypeImages` přiřadíme vytvořenému konfiguračnímu typy ikonu, která se bude zobrazovat v konfiguraci.

```
1  <extension
2            id="simcoplugin.launchConfigurationType"
3            point="org.eclipse.debug.core.launchConfigurationTypes">
4         <launchConfigurationType
5               delegate="simcoplugin.launcher.Launcher"
6               delegateName="Simco Project Launcher"
7               id="simcoplugin.launchConfigurationType"
8               modes="run"
9               name="Simco Project"
10              public="true">
11        </launchConfigurationType>
12 </extension>
```

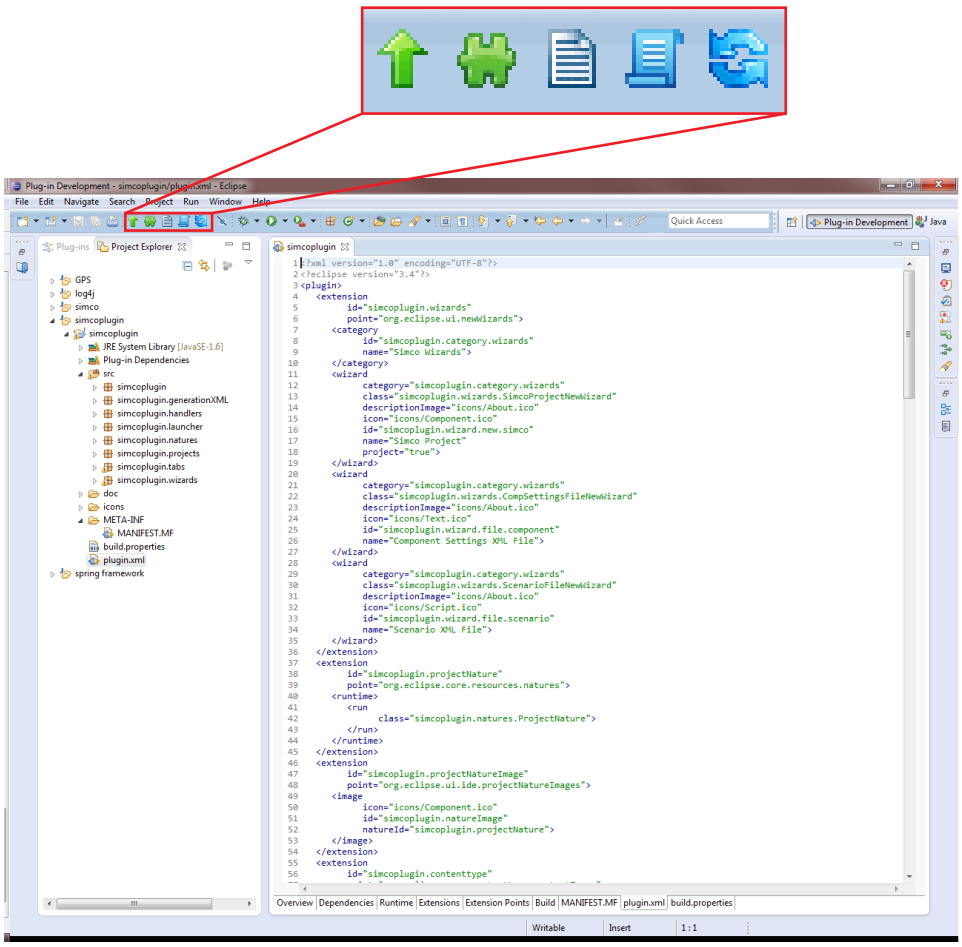
Výpis 12 – Definice launcheru

8.2.6 Nástrojová lišta (Toolbar)

Nyní vytvořené wizardy a commandy zpřístupníme uživateli v *toolbaru* (nástrojové liště). K tomu využijeme extension point `org.eclipse.ui.menus` a přidáme element `menuContribution`, kde definujeme nový toolbar. Umístění toolbaru je dáno atributem `locationURI` zde: `toolbar:org.eclipse.ui.main.toolbar`. To znamená, že se toolbar bude zobrazovat v hlavní nástrojové liště Eclipse workbench. V ukázce je vytvořen toolbar s jedním tlačítkem napojeným na command, který jsme vytvořili v kapitole 8.2.4. Styl tlačítka je nastaven v atributu `style`, icona v atributu `icon` a tooltip (nápověda po umístění kurzoru myši nad prvek) v atributu `tooltip`. Podobným způsobem se tlačítko napojí i na wizard. V elementu `command` přibyde element `parameter` s atributy `name` s hodnotou `newWizardId` a atributem `value`, jehož hodnota bude id wizardu. Výsledný vzhled Simco nástrojové lišty je na obrázku 31.

```
<extension
          point="org.eclipse.ui.menus">
       <menuContribution
             locationURI="toolbar:org.eclipse.ui.main.toolbar?after=additions"
                >
          <toolbar
                id="simcoplugin.toolbar"
                label="Simco Toolbar Commands">
             <command
                   commandId="simcoplugin.commands.workspaceCommand"
                   icon="icons/Raise.ico"
                   label="Switch Workspace"
                   style="push"
                   tooltip="Switch Workspace">
             </command>
</extension>
```

Výpis 13 – Definice toolbaru



Obrázek 31 – Nástrojová lišta Simco pluginu

8.2.7 Manifest.mf

Po přečtení kapitoly 5.2.1 je obsah samotného manifestu je očekávatelný. Je zde především výčet všech závislostí, které plugin potřebuje ke svému spuštění. Plugin se tedy vůbec nespustí a nezobrazí v toolbaru Eclipse, pokud nebudou k dispozici bundly Simco frameworku.

8.3 Balíky

V této kapitole projdeme všechny balíky aplikace a popíšeme důležité funkčnosti jejich tříd, nejvýznamnější metody a některé řešené problémy.

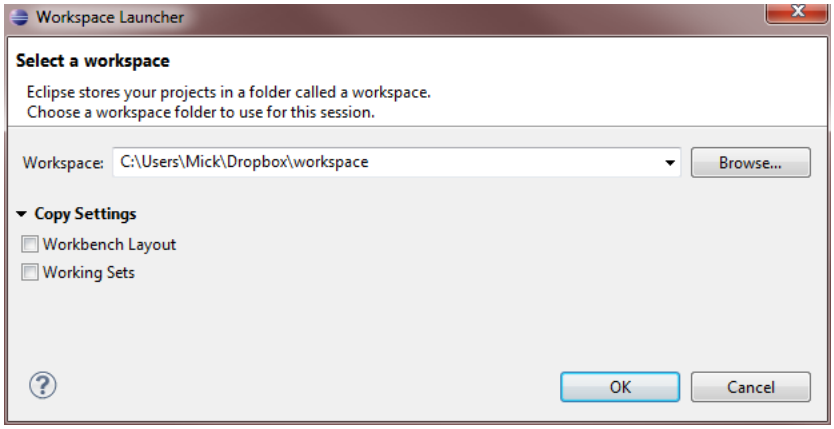
8.3.1 simcoplugin

Obsahuje pouze aktivační třídu, jejíž metody jsou prostředím Eclipse volány při startu a ukončení pluginu. Také umožňuje přístup ke grafickým ikonám.

8.3.2 simcoplugin.handlers

Obsahuje dvě třídy, které obě dědí od `AbstractHandler` z balíku `org.eclipse.core.commands`, která implementuje rozhraní `IHandler`. Obě třídy implementují metodu `execute`, která provede vykonání commandu.

Přepnutí workspace `SwitchWorkspaceHandler` je handler pro command `simcoplugin.commands.workspace`. V metodě `execute` vyvolá dialogové okno pro přepnutí workspace využitím interní třídy Eclipse `OpenWorkspaceAction`.



Obrázek 32 – Dialogové okno pro přepnutí workspace

Refresh projektů Handler pro command `simcoplugin.commands.refreshCommand`. Projde všechny projekty v adresáři workspace a importuje je do pracovního prostoru Eclipse. Tato funkčnost je nutná, protože po překopírování do adresáře workspace projekty nejsou vidět ve vývojovém prostředí Eclipse, dokud se takto neimportují. Tato funkce je vytvořená jako command, aby byla dostupná v toolbaru aplikace, protože může být užitečná nejen pro Simco plugin, ale i v jiných případech.

8.3.3 simcoplugin.natures

Zde je pouze třída `ProjectNature`, která implementuje rozhraní `IProjectNature` a umožňuje vytvořit nový typ projektu. ID nature tohoto Simco projektu je `simcoplugin.projectNature`.

8.3.4 simcoplugin.projects

V tomto balíku jsou třídy zajišťující import projektů do workspace.

Import komponent Důležitou třídou je `SimcoProjectSupport`. Metody této třídy jsou volány po dokončení wizardu pro import projektů do workspace a vytvoření Simco projektu. Metoda `createProject` nejdříve vytvoří bázi projektu, a poté je mu přiřazeno project nature z

třídy `ProjectNature`. Metoda `copyProjectsToWorkspace` překopíruje adresáře do workspace, nastaví cesty k těmto souborů a vyvolá command `simcoplugin.commands.refreshCommand` zajišťující obnovení projektů tak, aby byly viditelné ve workspace Eclipse. Samotné kopírování adresářů do workspace zajišťuje třída `CopyProjects`. Dále je zde metoda `addToProjectXMLfile`, která hledá XML soubor nastavení komponenty v kořenovém adresáři projektu. Pokud takovýto projekt není k dispozici, existují dvě možnosti. Komponenta je jiného typu než reálná či simulovaná a nebo nebyl vytvořen tento soubor. První případ lze vyřešit manuálním nastavením komponenty, a to upravením XML souboru scénáře. Druhý případ lze vyřešit vytvořením kostry XML souboru komponenty pomocí připraveného wizardu a dopsáním potřebných nastavení.

8.3.5 simcoplugin.tabs

Tento balík obsahuje třídy, které vytvářejí podporu pro záložky Run configurations Simco projektu.

Skupina záložek run configurations Každá Run configuration má svojí skupinu záložek. Testované komponenty a framework Simco potřebují ke spuštění OSGi, proto třída `SimcoLaunchGroup` dědí od `OSGiLauncherTabGroup`, přebírá záložky z této skupiny a přidává záložku `SimcoProjectTab` vytvořenou v rámci Simco pluginu.

Záložka Simco projektu Třída reprezentující novou záložku se jmenuje `SimcoProjectTab` dědí od třídy `AbstractLaunchConfigurationTab` a překrývá některé její metody. Tato záložka, zobrazuje všechny validně vytvořené simulační scénáře pro Simco framework. Tyto scénáře mají vlastní content type a nacházejí se v Simco projektu. Proto se nejdříve projdou pouze projekty s nature Simco projektu a vyberou se jen ty z content type Simco scenario. Data o každém XML scénáři jsou uloženy ve třídě `XMLFileData`. Uživatel může zvolit pouze jeden z nabízených scénářů, avšak tlačítka nemohou být obsahem tabulek (v tomto případě by se hodily přepínací tlačítka) ani jiné SWT komponenty. K zobrazení dat je použita JFace komponenta `CheckboxTableViewer`, která poskytuje zaškrtávací tlačítka. Možnost označení pouze jednoho z nich je ošetřena programově v listeneru.

V záložce Simco Project Tab vzniknul problém s uložením konfigurace. Konfigurace záložek musí být uložena, aby mohla být příště vyvolána v nezměněném nastavení. Konfigurace používá rozhraní `ILaunchConfigurationWorkingCopy`, které poskytuje metodu `setAtributte` pro nastavení (uložení) atributů záložky s omezenými možnosti uchovávání objektových typů. Pro naše účely, kdy potřebujeme uchovat identifikátor řádku a příznak výběru, využíváme mapu, kde klíčem je identifikátor řádku a hodnota „true“ nebo „false“. Běžně bychom použili pro klíč `integer` a pro hodnotu `boolean`, ale zde máme k dispozici pro obě hodnoty pouze `string`. Proto při uložení musí dojít k přetypování a při inicializaci taktéž. Celá akce uložení je provedena v metodě `performApply`, která se provede po stisknutí tlačítka „Apply“. Incializace

uložené konfigurace je implementována v metodě `initializeFrom`, která se spouští při otevření této spouštěcí konfigurace. Při prvním vytvoření konfigurace je automaticky zaškrtnut první scénář ze seznamu.

8.3.6 simcoplugin.launcher

Zde je implementována akce po spuštění Run configuration.

Spuštění Akce po spuštění, tedy po stisku tlačítka Run v Simco spouštěcí konfiguraci, obstarává třída `Launcher`. Tato třída dědí od `OSGiLaunchConfigurationDelegate`, protože spuštění samotného Simco frameworku zajišťuje OSGi. Přesto však překrývá metodu `launch`, kde získá z atributu spouštěcí konfigurace cestu ke zvolenému scénáři. Nastavení této cesty zajišťuje třída `XMLConfig`, která uloží tuto cestu do souboru `pathToScenario` do konfiguračního adresáře `simco.config`, odkud ho poté Simco framework načte.

8.3.7 simcoplugin.wizards

Zde jsou umístěny wizardy a jejich jednotlivé stránky, dále také třídy podporující jejich funkce. Jsou zde implementovány tři wizardy:

- wizard pro vytvoření kostry XML konfiguračního souboru komponenty,
- wizard pro vytvoření kostry XML konfiguračního souboru scénáře,
- wizard pro vytvoření nového Simco projektu.

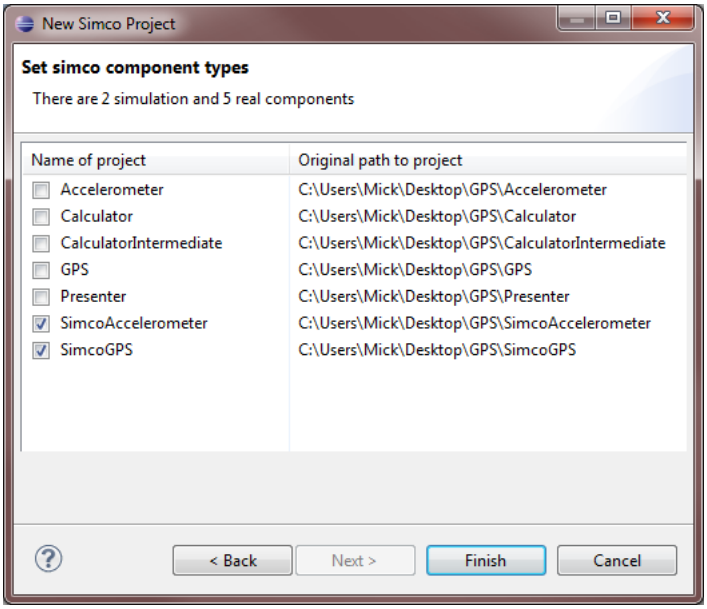
Vytvoření kostry XML konfiguračního souboru komponenty Hlavní třídou wizardu `ComponentSettingsFileNewWizard`, jemuž zajišťuje podporu třída `SimcoProjectNewFile`. `SimcoProjectNewFile`, je abstraktní třída dědící od `Wizard` a implementující rozhraní `INewWizard`, která poskytuje metody vykonané při incializaci a během ukončení wizardu vytvářejícího nový soubor. Jedinou stránkou wizardu je `CompSettingsFileNewPage` dědící od `WizardNewFileCreationPage`. V metodě `getInitialContents` jsou XML elementy kostry, která je obsahem nově vytvořeného souboru.

Vytvoření kostry XML konfiguračního souboru scénáře Analogicky jako v předchozím případě je vytvořena podpora pro kostru XML konfiguračního souboru scénáře. Hlavní třídou tohoto wizardu je `ScenarioFileNewWizard` a jeho stránkou `ScenarioFileNewPage`.

Vytvoření nového Simco projektu Hlavní třídou tohoto wizardu je `SimcoProjectNewWizard`, která nastavuje všechny stránky wizardu. Kontroluje, jestli je možné wizard, již ukončit a provádí akci při jeho ukočení. Tyto akce jsou především importování komponent a vytvoření scénáře, které zajišťují metody třídy `SimcoProjectSupport`.

První stránkou wizardu je `SimcoProjectNewProjectPage`, která dědí od `WizardPage`. Tato třída zajišťuje zadání jména nového projektu. Navíc je zde přidáno pole pro zadání názvu konfiguračního XML souboru scénáře, a také prohlížeč pro určení adresáře, ze kterého se budou importovat projekty do workspace.

Druhou stránkou wizardu je `SimcoProjectCompSettingsTypePage`, která dědí od abstraktní třídy `WizardPage` a používá komponetu JFace `CheckboxTableViewer` k zobrazení jednotlivých importovaných komponent. Uživatel označí ty, které jsou simulované a ostatní jsou považované za reálné. Souhrnné informace o projektech jsou ve třídě `Projects` a informace o jednotlivých projektech pro generování simulačního scénáře jsou ve třídě `ProjectData`. Jedním z problémů řešených v této stránce bylo, jakým způsobem zobrazit data. Všechny stránky wizardu jsou vytvořeny při spuštěním wizardu, avšak tato druhá stránka wizardu je závislá na datech (adresář z projekty), které jsou zadány v první stránce. Tento problém byl vyřešen použitím zmíněné JFace komponety, kdy je možné vytvořit komponentu, ale vstupní data nastavit později. Data se nastaví v metodě `setVisible`, kde se kontroluje, jestli je tato stránka již aktivní. Pokud ano, uživatel musel úspěšně projít stránkou první a data o projektech jsou k dispozici.



Obrázek 33 – Ukázka druhé stránky wizardu Simco projekt

8.3.8 simcoplugin.generationXML

Zde je třída `XMLFile` starající se o generování XML konfiguračního souboru scénáře na základě dat uložených ve třídě `ProjectData`.

8.4 Změny v Simco frameworku

V samotném Simco frameworku došlo k několika menším změnám, které jsou popsány v následujících podkapitolách.

8.4.1 Přechod k novější verzi OSGi

Nejdříve byla odstraněna chyba, která vznikla přechodem k novější verzi Eclipse a OSGi. Simco framework byl vyvinut v Eclipse Europa (verze 3.3.2) a OSGi Realease 4 verze 4.1. Zde byl pro třídu `Event` pouze jeden konstruktor skládající se z objektů `String` a `Dictionary`. Od OSGi Realease 4 verze 4.2 a Eclipse Galileo (verze 3.5)má třída `Event` dva konstruktory. K původnímu navíc přibyl konstruktor `String` a `Map`. Ve frameworku Simco, konkrétněji v projektu `framework.scheduler`, je používán konstruktor `String` a druhým parametrem je `Hashtable`. Vzhledem k tomu, že `Hashtable` dědí od `Dictionary` a zároveň implementuje rozhraní `Map`, vznikla zde dvojznačnost, kde není zřejmé, který konstruktor použít. Tento problém byl vyřešen implicitní konverzí na `Map`. Tyto změny byly realizovány v třídách `EventsInvoker` a `SimcoSimulation` v projektu `framework.scheduler` a jsou okomentovány v programovém kódu.

8.4.2 Změna konfiguračních cest

Konfigurační soubory byly umístěny vždy v kořenovém adresáři daného projektu. Po přechodu instalujeme jar soubory Simco tak, že je umístíme do adresáře `Eclipse/plugins`. Byl vytvořen adresář `simco.config`, který je ve stejném adresáři jako jar soubory frameworku. Do tohoto adresáře se přesunou veškeré konfigurační soubory Simco. V kódu Simco tedy musely být změněny tyto cesty. V programovém kódu jsou tyto místa označená a původní přístup je zakomentován. Změny proběhly v projektu `framework.core` a ve třídě `CoreImpl` a v projektu `framework.gui` ve třídě `Configuration`.

8.4.3 Načtení simulačního scénáře

Dříve se načítal simulační scénář pouze přes menu grafického prostředí a byl to první krok realizovaný uživatelem. Simco plugin přenesl tuto volbu při vytváření do prostředí Eclipse, kde si uživatel vytvoří svojí spouštěcí konfiguraci, a zde nastaví simulační scénář. Cesta ke zvolenému scénáři je nastavena do konfiguračního adresáře `simco.config`, odkud si ho grafické rozhraní Simco frameworku načítá při spuštění. V samotném programovém kódu byla přidána metoda `readPathToScenarioFile` ve třídě `GuiMainFrame`, která načte cestu ke scénáři. Tato cesta je předána metodě `setScenario` ve třídě `FileActionsListener`, která předá scénář jádru frameworku. Tyto akce jsou provedeny po vykreslení hlavního okna aplikace v metodě `startWindow`. Všechny zmíněné změny byly provedeny v projektu `framework.gui`.

8.5 Prostředí

Aplikace byla vyvíjena a testována s použitím následjících nástrojů a prostředí:

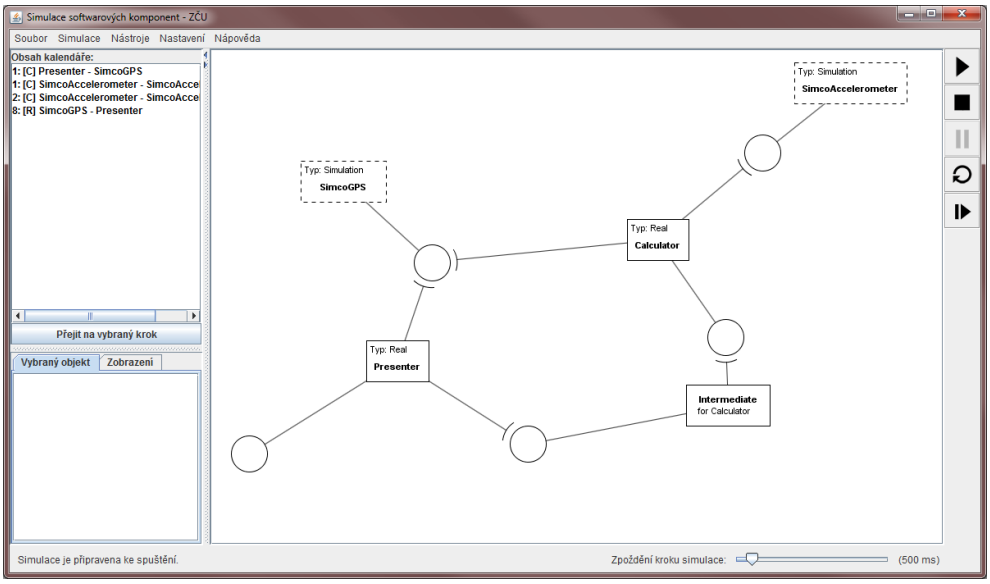
- Java verze 1.7,
- Windows 7 Professional 32-bit,
- Eclipse for RCP and RAP Developers verze 4.2.1.

9 Testování

Aplikace nemá žádné speciální požadavky na hardwarové vybavení počítače. Všechny kroky potřebné k instalaci a spuštění Simco pluginu a Simco frameworku naleznete v příloze A. V této kapitole popíšeme testování naimplementovaného pluginu v jednotlivých krocích, které uživatel projde při jeho použití, abychom otestovali funkčnost celého řešení a jeho provázanost se Simco frameworkem. Pro testování použijeme speciální aplikaci vyvinutou v rámci diplomové práce [12]. Zde o ní naleznete také podrobnější informace. Pro naše účely postačí znalost faktu o tom, že se skládá ze softwarových komponent, které jsou připravené jako projekty a můžeme je jednoduše importovat.

Scénář testování krok po kroku:

1. Spuštění vývojového prostředí Eclipse s předinstalovaným Simco frameworkem a Simco pluginem a splněnými všemi potřebnými závislostmi.
2. Použití tlačítka na přepnutí workspace.
   (a) Nastavena cesta k novému workspace a potvrzení jeho otevření tlačítkem "OK". Aktuální stav workbench Eclipse se uloží a zavře. Poté se znovu otevře již v novém workspace.
3. Použití tlačítka na vytvoření Simco projektu. Otevře se wizard nového Simco projektu.
4. První stránka wizardu nového Simco projektu.
   (a) Nastavení názvu nového projektu.
   (b) Nastavení jména simulačního scénáře.
   (c) Zvolení adresáře v souborovém systému, který obsahuje komponenty testovací aplikace.
   (d) Stisk tlačítka „Next".
5. Druhá stránka wizardu nového Simco projektu.
   (a) Výběr simulačních komponent.
   (b) Po stisku tlačítka „Finish" se vytvoří nový Simco projekt ve workspace a v něm XML konfigurační soubor scénáře.
6. Úprava XML simulačního scénáře v editoru Eclipse.
7. Přechod do dialogového okna Run configurations.
   (a) Vytvoření spouštěcí konfigurace s nastavením všech bundlů testovací aplikace v záložce „Bundles".



Obrázek 34 – GUI Simco s vykreslenými komponenty definovanými ve scénáři

(b) Zaškrtnutím simulačního scénáře v záložce „Simco Project Tab".

(c) Uložení vytvořené konfigurace stisknutím tlačítka „Apply".

(d) Spuštění konfigurace tlačítkem „Run".

8. Otevřeno grafické prostředí Simco frameworku s načteným scénářem a vykreslení definovaných komponent viz obrázek 34.

10 Závěr

Tato práce se zabývá návrhem a realizací převedení testovacího nástroje Simco do Eclipse pluginu. V průběhu realizace práce byly nejprve zkoumány postupy implementace pluginu do Eclipse s využítím nástrojů, které toto prostředí pro vývoj pluginů poskytuje. Po prozkoumání a analýze Simco bylo rozhodnuto, že se funkcionalita pluginu soustředí především na přínosy propojení nástroje Simco a vývojového prostředí Eclipse. Uskutečné změny v samotném nástroji Simco byly minimální.

Celý plugin byl zasazen do Eclipse IDE tak, aby práce s ním byla z uživatelského hlediska intuitivní. Všechny uživatelské prvky pluginu jsou umístěny na místech, kde by je uživatel tohoto vývojového prostředí očekával. Veškeré komponenty frameworku Simco byly integrovány do Eclipse, takže se uživatel může soustředit pouze na samotné testované komponenty a vytváření konfiguračních XML souborů, pro které plugin také zajišťuje podporu. Implementace byla otestována spuštěním testovací aplikace s připraveným simulačním scénářem.

Všechny body zadání byly splněny. Nabízejí se zde možnosti rozšíření aplikace. Jedním z možných vylepšení je například automatické označení testovaných komponent definovaných v simulačním scénáři v záložce „bundles“ spouštěcí konfigurace, aby uživatel tuto akci nemusel provádět manuálně.

Seznam obrázků

Seznam výpisů

Seznam tabulek

Seznam zkratek

IDE (Integrated Development Enviroment) - vývojové prostředí

HTML (Hypertext Markup Language) - značkovací jazyk pro tvorbu webových stránek

RCP (Rich Client Platform) - nástroj podporující integraci nezávislých softwarových komponent

OSGi (Open Services Getaway initiative) - modulární systém pro Javu

SWT (Standard Widget Toolkit) - grafický nástroj platformy Java

API (Application Program Interface) - programové rozhraní aplikace

SDK (Software Development Kit) - balíček nástrojů umožňující tvorbu jistých aplikací

CBSE/CBD (Component-Based Software Engineering/Component-Based Development) - komponentové programování

JEE (Java Enterprise Edition) - Java platforma především pro vývoj podnikových systémů

XML (Extensible Markup Language) - značkovací jazyk

JVM (Java Virtual Machine) - virtuální stroj pro zpracování Java bytcode

JAR (Java Archive) - formát na komprimaci a distribuci Java souborů

Reference

[1] *Eclipse documentation* [online]. [cit. 2012-12-10]. Dostupné z: http://help.eclipse.org

[2] Eclipse 4 RCP - Tutorial. *Vogella* [online]. [cit. 2013-01-20]. Dostupné z: http://www.vogella.com/articles/EclipseRCP/article.html

[3] CLAYBERG, Eric a Dan RUBEL. *Eclipse plug-ins*. 3rd ed. Upper Saddle River: Addison-Wesley, c2009, xlv, 878 s. The eclipse series. ISBN 978-0-321-55346-1.

[4] The Apache Ant Project [online]. [cit. 2013-01-20]. Dostupné z: http://ant.apache.org/

[5] Lazy Activation Policy. *OSGi Alliance* [online]. [cit. 2013-01-24]. Dostupné z: http://www.osgi.org/Design/LazyStart

[6] THE OSGI ALLIANCE. *OSGi Service Platform Core Specification version 4.2* [online]. 4. vyd. 2009 [cit. 2013-04-21]. Dostupné z: http://www.osgi.org/download/r4v42/r4.core.pdf

[7] BACHMANN, Felix. *Technical Concepts of Component-Based Software Engineering* [online]. 2ndEdition. Pittsburgh: CarnegieMellon University, 2000 [cit. 2013-04-16]. VolumeII. Dostupné z: http://www.sei.cmu.edu/reports/00tr008.pdf

[8] *Introduction to Spring Framework* [online]. [cit. 2013-04-28]. Dostupné z: http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/overview.html

[9] COGOLUEGNES, Arnaud, TEMPLIER Thierry, PIPER Andy, Spring Dynamic Modules in Action. Manning Publication Co., září 2010. ISBN 9781935182306

[10] *The OSGi Architecture* [online]. 2013 [cit. 2013-04-28]. Dostupné z: http://www.osgi.org/Technology/WhatIsOSGi

[11] *Blueprint Container* [online]. [cit. 2013-05-10]. Dostupné z: http://static.springsource.org/spring-osgi/snapshot-site/reference/html/blueprint.html

[12] KABÍČEK, Tomáš. *Simulační systém softwarových komponent založený na komponentách.* Plzeň, 2011. 87 s. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky

[13] PROKOP, Matěj. *Vizualizace simulace komponent.* Plzeň, 2011. 100 s. Diplomová práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky

[14] *Migrate your Swing application to SWT* [online]. [cit. 2013-05-10]. Dostupné z: http://www.ibm.com/developerworks/java/tutorials/j-swing2swt/section5.html

Přílohy

A Uživatelská příručka

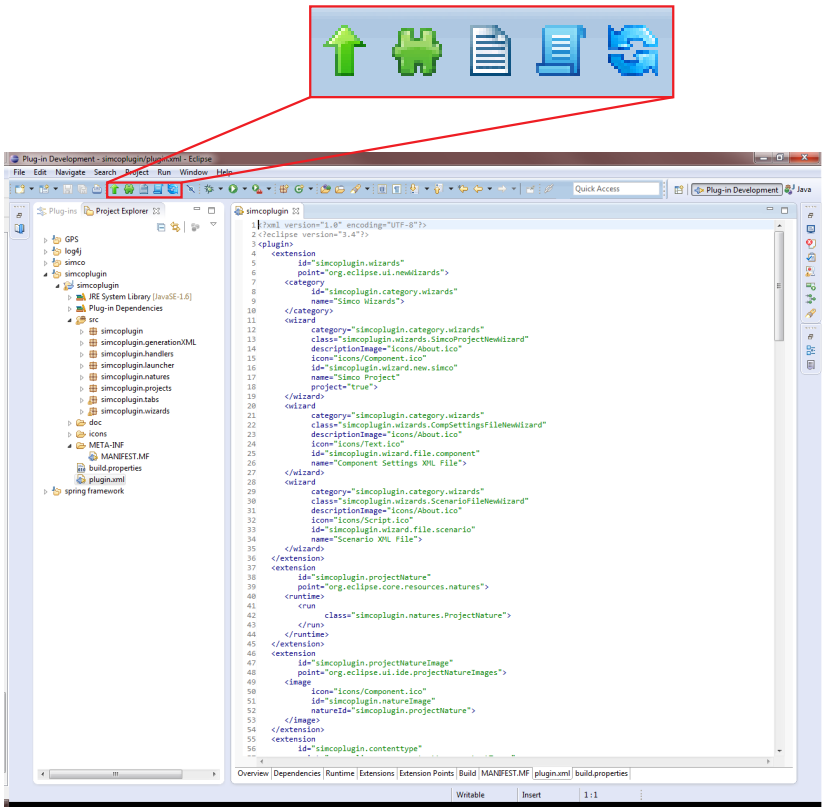
Instalace

Prvním krokem je pořízení vývojového prostředí Eclipse. Níže popsaný byl otestován a měl by bezproblémů fungovat na všech verzích Eclipse Juno. Všechny potřebné soubory, včetně vývojového prostředí Eclipse, jsou k dispozici na přiloženém CD.

1. Překopírování všech knihoven z adresáře `Aplikace/knihovny` do adresáře `Eclipse/Plugins`.
2. Překopírování jar souborů frameworku Simco a adresáře `simco.config` z adresáře `Aplikace/simco` do adresáře `Eclipse/Plugin`.
3. Překopírování Simco pluginu z adresáře `Aplikace/simco plugin` do adresáře `Eclipse/Plugins`.
4. Spuštění Eclipse.

Ovládání

Ovládání programu ukážeme na postupu, který uživatel prochází při importu a spuštění aplikace. Pokud všechny kroky instalace proběhly úspěšně, bude po spuštění Eclipse na hlavní nástrojové liště toolbar Simco pluginu, který je zobrazen na obrázku 35.

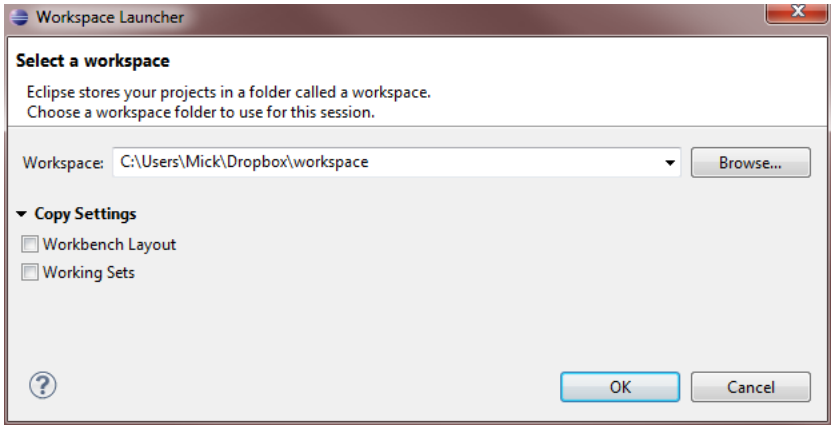


Obrázek 35 – Toolbar Simco pluginu

Popis tlačítek (a klávesová kombinace) z obrázku 35 zleva:

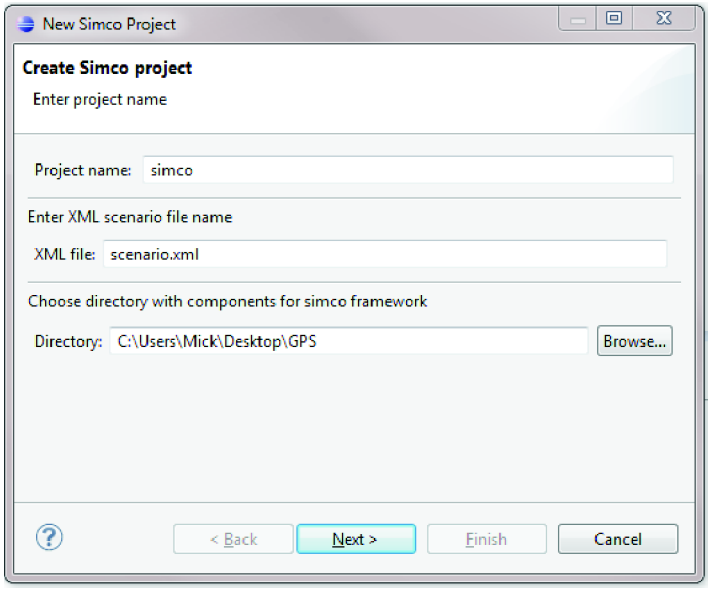
- tlačítko pro přepnutí workspace (CTRL+ALT+1),
- tlačítko pro vytvoření nového Simco projektu (CTRL+ALT+2),
- tlačítko pro vytvoření kostry XML konfiguračního souboru komponenty (CTRL+ALT+3),
- tlačítko pro vytvoření kostry XML konfiguračního souboru scénáře (CTRL+ALT+4),
- tlačítko pro import všech projektů z workspace, aby byly viditelné v Eclipse (CTRL+ALT+5).

Pokud máme nějaké projekty ve workspace je lepší přepnout do jiného „čistého“ workspace. Tuto akci provedeme stisknutím prvního tlačítka a nastavením workspace viz obrázek 36.



Obrázek 36 – Změna workspace

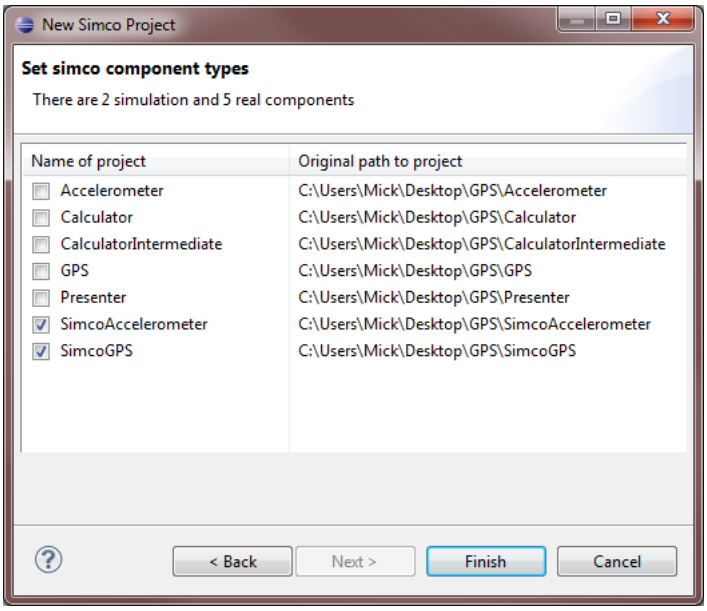
Další krokem bude vytvoření nového projektu. Stisknutím druhého tlačítka nástrojové lišty Simco pluginu se zobrazí první stránka průvodce viz obrázek 37. Zde je potřeba nastavit jméno projektu, název konfiguračního souboru scénáře, a také cestu k adresáři, který obsahuje projekty, které se budou importovat. Jméno projektu nesmí být shodné s žádným názvem importovaného projektu, protože by zde vznikla kolize. Jestli je nějaký název projektu shodný s tím, který vytváříte, lze to snadno zjistit z druhé stránky průvodce, kde je seznam všech importovaných komponent i s jejich názvy, není tedy problém se o krok vrátit a název nového projektu změnit.



Obrázek 37 – Nový Simco projekt - první stránka průvodce

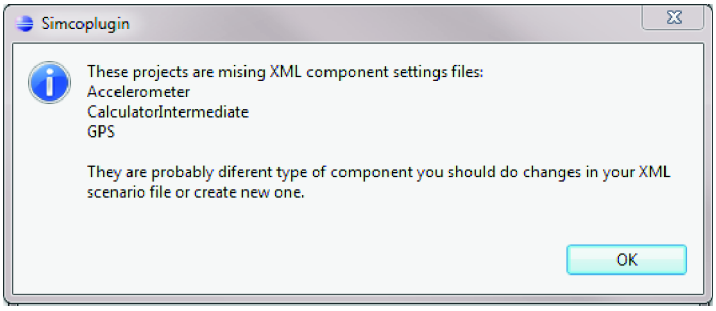
Na druhé stránce wizardu je nutné zaškrtnout ty projekty, které představují simulované

komponenty ostatní budou automaticky nastaveny jako reálné.



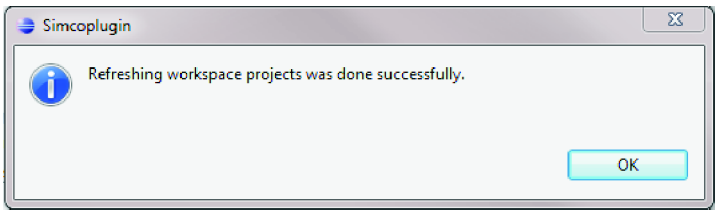
Obrázek 38 – Nový Simco projekt - druhá stránka průvodce

Po stisknutí tlačítka „Finish“ proběhne nejdříve vytváření scénáře, hledání konfiguračních souborů komponent. Uživatel je upozorněn na komponenty, kterým tento soubor chybí. Tyto komponenty jsou pravděpodobně jiného typu nebo jim tento soubor nebyl vytvořen.



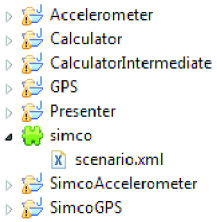
Obrázek 39 – Projekty, kterým chybí konfigurační soubor

Po potvrzení předchozího informačního dialogu, proběhne import, který při větším množství komponent může trvat několik sekund. Počkejte tedy na informační dialogové okno s hlášením, že import proběhl.



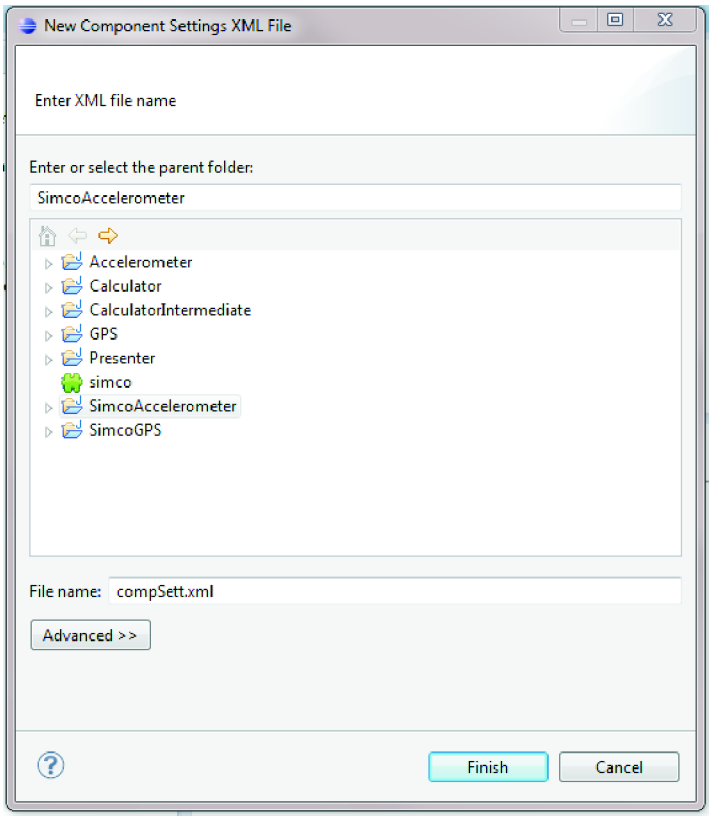
Obrázek 40 – Import proběhl úspěšně

Ve workspace je vytvořen nový Simco projekt obsahující vygenerovanou část simulačního scénáře a importované projekty.



Obrázek 41 – Workspace se Simco projektem

Dalším krokem může být vytvoření nového konfiguračního souboru komponenty. Použijeme třetí tlačítko nástrojové lišty Simco pluginu. Zvolíme název XML souboru a umístíme ho do kořenového adresáře projektu pro který tento soubor vytváříme. Vygeneruje se kostra, kterou můžeme libovolně upravit.

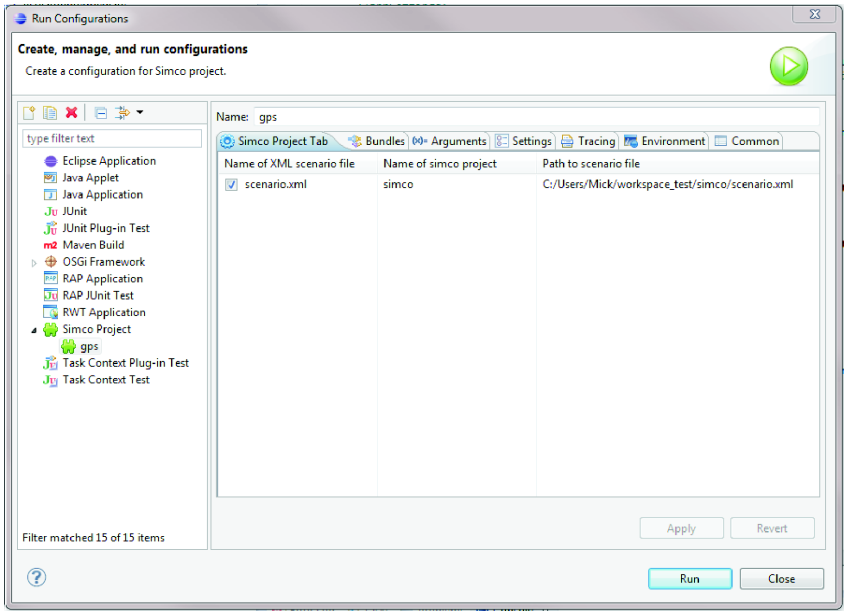


Obrázek 42 – Stránka průvodce pro vytvoření konfiguračního souboru komponenty

Podobně vytvoříme nový simulační scénáře použitím čtvrtého tlačítka zleva nástrojové lišty Simco pluginu. Nový scénář je nutné umístit do Simco projektu, aby byl viditelný v Run configurations.

Poslední tlačítko nástrojové lišty Simco pluginu slouží pro import všech projektů, které se nacházejí ve workspace, ale nejsou viditelné v Eclipse. Jeho funkcionalita je využita při importu komponent. Toto tlačítko se může hodit například pokud uživatel smaže projekt, ale neoznačí jeho smazaní jako nezvratné. Projekt zůstane ve workspace a tímto tlačítkem se jednoduše importuje zpět do Eclipse.

Po editaci a konfiguraci XML souboru scénáře spustíme testování. Pro testovací aplikaci Simco, kterou lze nalézt na přiloženém kompaktním disku, je přiraven simulační scénář, který můžete importovat do Simco projektu a pouze pozměnit cesty k jednotlivým konfiguračním souborům komponent. Před spuštěním je nutné nejdříve vytvořit Run configuration pro Simco projekt. Ve workspace je vytvořen pouze jeden scénář, takže tento scénář bude automaticky označen v záložce Simco project. Pokud existuje více scénářů v Simco projektu (projektech) budou zde všechny zobrazeny.



Obrázek 43 – Simco project tab - Run configuration

V záložce bundles se nastavují bundly, které se budou spouštět. Implicitně jsou označeny všechny bundly ve workspace i v platformě Eclipse. Není vhodné spouštět všechny bundly, když velké množství z nich k běhu nepotřebujeme. Necháme však vyberné všechny bundly ve workspace, protože tam máme jen testované komponenty a bundle Simco projektu se zde nezobrazuje. Odznačíme políčko Target Platform, čímž se zde odznačí všechny bundly a stiskneme tlačítko „Add Required Bundles“ v pravé části dialogového okna. Většina potřebných bundlů se nám automaticky označí. Existuje však několik, které musíme označit ručně. Při jejich hledání si můžeme pomoci zadáváním jejich jmen do filtrovací textového pole (type filter text). Označíme tyto bundly:

- framework.core,
- framework.gui,
- org.eclipse.zest.layouts,
- org.springframework.osgi.extender,
- org.springframework.osgi.catalina.osgi.

Konfiguraci uložíme tlačítkem „Apply“ a spustíme tlačítkem „Run“. Nic by nemělo bránit ke spuštění testovacích výpisů do konzole Eclipse a po chvíli se spustí i hlavní okno Simco frameworku s vykreslenými komponenty podle nastaveného simulačního scénáře.

B Obsah kompaktního disku

Přiložený kompaktní disk obsahuje tyto adresáře

- Aplikace
  - GPS
    - komponenty testovací aplikace
    - `scenario.xml` - připravený simulační scénář pro testovací aplikaci (nutné přenastavit cesty k projektů)
  - knihovny
    - knihovny potřebné ke spuštění aplikace - knihovny log4j, spring frameworku a zest
  - simco framework
    - `jar` soubory frameworku simco - obsahuje i zdrojové texty programů
    - adresář **simco.config** obsahující soubor `configuration.xml` s vychozím nastavením pro grafické rozhraní simco
  - simco plugin
    - `jar` soubor simco pluginu - obsahuje také zdrojový kód pluginu a javadoc dokumentaci
- Eclipse
  - Eclipse for RCP and RAP developers verze 4.2.1. (Juno), ve kterém byla aplikace vyvíjena a testována
- Text
  - text diplomové práce ve formátu PDF
  - adresář **Lyx** - obsahující zdrojové soubory textu (.lyx) psaného v programu LyX verze 2.0.5.