Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

GUI pro testování komponentových aplikací

Plzeň 2014 Lukáš Rostás

Prohlášení

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

V Plzni dne 26. června 2014

Lukáš Rostás

Poděkování

Rád bych poděkoval Ing. Richardu Lipkovi, PhD. za vstřícnost, ochotu a cenné rady při vedení této práce.

Abstract

The goal of this bachelor's thesis is to design and implement a library that creates a graphical user interface which makes it easier to control testing of simulated components. This GUI will be generated dynamically and will contain elements based on the programmer's requierements. The theoretical part of this paper describes the OSGi, the Spring framework and the principles of component-based development. The second part aims to describe the design and implemenation of the library.

Obsah

1 Úvod

Cílem této práce je implementovat knihovnu, která vytvoří grafické uživatelské rozhraní usnadňující ovládání testování simulovaných komponent. Toto GUI bude generované dynamicky a bude obsahovat prvky na základě požadavků programátora komponent. Knihovna umožní nastavovat parametry jednotlivých komponent v simulaci a zároveň sledovat proměnné a parametry komponent jiných a zobrazovat je v grafické či textové podobě.

V první, teoretické, části práce je text zaměřen na vysvětlení základních pojmů komponentového programování a seznámení se s frameworky OSGi a Spring. Dále bude představen simulátor komponent Simco, který je vyvíjen na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni. Realizační část bude zaměřena na samotnou aplikaci vytvořenou v rámci této bakalářské práce.

2 Komponentové programování

Komponenty, komponentové programování a komponentový software v posledních letech nabývají na důležitosti. Základní myšlenka komponentově orientovaného softwarového inženýrství (CBSE)[1] je vytváření aplikací propojováním jednotlivých na sobě nezávislých částí, tzv. komponent. Tento přístup umožňuje urychlit vývoj aplikací, jejich dlouhodobou udržitelnost a v neposlední řadě zlevnit ceny software díky znovupoužitelnosti již existujících komponent.

2.1 Komponenty

2.1.1 Definice a vlastnosti

Definic softwarové komponenty existuje mnoho, pro účely této práce jsem vybral tyto dvě: Podle Clemense Szyperského [1] je softwarová komponenta jednotka kompozice skládající se ze smluvně specifikovaných rozhraní a explicitních kontextových závislostí. Softwarová komponenta může být nasazena nezávisle a je předmětem kompozice třetích stran. Jiná definice [6] říká, že komponenta je nezávisle dodávaná jednotka, která zapouzdřuje služby za veřejné rozhraní a která může být skládána s dalšími komponentami.

Obecně pro komponenty platí [3]:

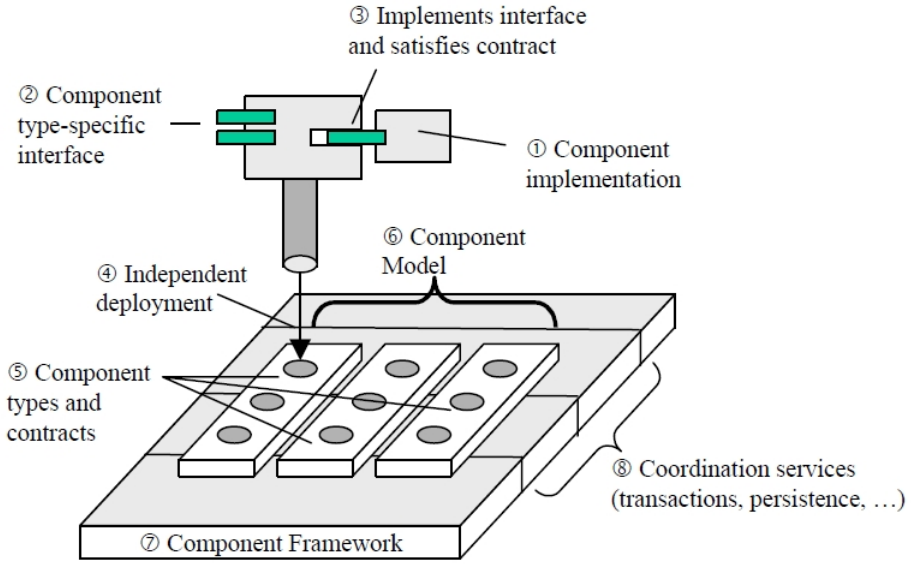
- neveřejná (skrytá) implementace funkcionality,
- jsou používány třetí stranou (third-party),
- odpovídají komponentovému modelu.

[1] Component-based software engineering nebo také CBD – component-based development.

Základní vlastnost komponent je skrytá implementace před okolím. Jediným způsobem, jak může třetí strana využít komponentu, je přes její rozhraní. Tento princip je znám jako „black-box“ model. Komponenta nemá přístup k implementaci ostatních komponent, ale pouze k jejich veřejnému rozhraní. Tím je odstraněna závislost na konkrétní implementaci a komponenta je snadno nahraditelná.

Z hlediska vývoje komponent a maximalizace možné znovupoužitelnosti je důležité brát v potaz, jak velkou funkcionalitu by měly komponenty zapouzdřovat. Lze říci, že příliš jednoduché komponenty vlastně nemají smysl a komponenty, které se snaží zahrnout příliš komplexní funkčnost, jsou zase málo flexibilní.

Na obrázku 2.1 jsou znázorněny vztahy mezi komponentami, komponentovým modelem a komponentovým frameworkem. Tyto pojmy budou vysvětleny dále.



Obrázek 2.1: Komponenta, model, framework a jejich vztah [3]

Komponenta (1) je softwarová implementace, která může být spuštěna na fyzickém nebo logickém zařízení. Komponenta může implementovat jedno

nebo více požadovaných rozhraní (2). Díky rozhraní může být mezi dvěma komponentami navázána komunikace. Povinnosti a závazky, které musí komunikující komponenty plnit, specifikují tzv. kontrakty (3). Kontrakty zajišťují, že se nezávisle vyvíjené komponenty budou řídit určitými pravidly tak, aby interakce komponent probíhaly předvídatelným způsobem a aby mohly být nasazeny do standardních běhových prostředí (4). Komponentově orientované systémy jsou založeny na malém počtu různých komponentových typů, z nichž každý z nich má jedinečnou roli (5) a je popsán rozhraním (2). Komponentový model (6) je soubor komponentových typů, jejich rozhraní a specifikací přípustných interakcí mezi komponentovými typy. Komponentový framework pak poskytuje různé služby (8) podporující a vynucující komponentový model. V mnoha ohledech lze komponentový framework přirovnat ke speciálnímu operačnímu systému [3].

Výhody komponentového přístupu:

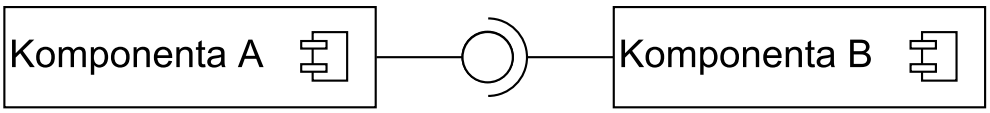
- nezávislý vývoj a nasazení,
- nezávislá rozšíření a snadná nahraditelnost komponent,
- znovupoužitelnost,
- kratší doba uvedení produktu na trh,
- úspora času a prostředků při vývoji,
- dlouhodobá udržitelnost systému.

2.1.2 Komponentové rozhraní

Jak již bylo zmíněno, jediný možný přístup ke službám komponenty je skrze její rozhraní. Je tedy důležité, aby toto rozhraní pro interakci s ostatními komponentami bylo jasně definované a zdokumentované. Stabilní rozhraní zajistí logické oddělení jednotlivých komponent a zamezí nežádoucím přístupům k vnitřní implementaci. V ideálním případě by rozhraní mělo být definováno na základě toho, co komponenta poskytuje a vyžaduje od ostatních

komponent [4]. Komponenta obvykle ke své funkčnosti požaduje rozhraní jiné komponenty, které značíme jako *required*. Rozhraní, která komponenta poskytuje, jsou označována *provided*.

Na obrázku 2.2 je znázorněno komponentové rozhraní mezi dvěma komponentami, kde komponenta A poskytuje požadované vlastnosti komponentě B a komponenta B tyto vlastnosti vyžaduje.



Obrázek 2.2: Rozhraní mezi komponentami

2.1.3 Komponentový model

Komponentový model je definice standardů pro implementaci komponenty, její dokumentaci a nasazení. Komponentový model zajišťuje, aby nezávisle vyvíjené komponenty mohly být bez problémů nasazeny ve společném běhovém prostředí. Model také určuje, jak by měla být definována rozhraní a jaké prvky by tato definice měla obsahovat. Všechny komponenty musí splňovat požadavky komponentového modelu.

Mezi tyto požadavky patří [3]:

- *Komponentové typy* – komponentový typ lze definovat na základě rozhraní, které komponenta implementuje. Pokud implementuje tři různá rozhraní X, Y a Z, tak komponenta je typu X, Y a Z a může libovolně zastávat roli X, Y nebo Z. Komponentový model vyžaduje, aby komponenta implementovala jedno nebo více rozhraní. Pak může model definovat jeden nebo více komponentových typů. Různé komponentové typy pak mohou v systému zastávat odlišné role a mohou být zapojeny v různých schématech interakce.

- *Schémata interakcí komponent* – komponentový model specifikuje použité komunikační protokoly a způsob dosažení požadované kvality služeb, jako je zabezpečení nebo transakce. Dále model popisuje, jak komponenty komunikují mezi sebou nebo s komponentovým frameworkem. Interakční schémata mohou být společná pro všechny komponentové typy, nebo jedinečná pro jednotlivé z nich.

- *Navázání zdrojů* – proces skládání komponent je záležitost navazování komponent na jeden nebo více zdrojů. Zdrojem je myšlena buď služba poskytovaná komponentovým frameworkem nebo jiná komponenta nasazená v daném frameworku. Komponentový model popisuje zdroje dostupné komponentám a také to, jak a kdy jsou komponenty navazovány na tyto zdroje. Naopak, framework považuje komponenty za zdroje, které je třeba řídit.

Komponentových modelů existuje celá řada, následuje výčet několika z nich. Kapitola 2.2 je pak celá věnována komponentovému model OSGi použitému při vývoji aplikace vytvářené v rámci této práce.

Příklady komponentových modelů:

- *Enterprise JavaBeans (EJB)* – komponentová architektura, sloužící pro realizaci aplikační vrstvy informačního systému. EJB komponenty jsou objekty implementované vývojářem, které zajišťují vlastní aplikační logiku systému. Jedná se o součást platformy Java EE[2] [7].

- *Component Object Model (COM)* – platformě nezávislý, distribuovaný, objektově orientovaný systém pro vytváření binárních softwarových komponent, které jsou schopny spolupracovat. Byl uveden společností Microsoft v roce 1993.

- *Common Object Request Broker Architecture (CORBA)* – standard definovaný konsorciem OMG (Object Management Group) tvořící ucele-

[2]Java Platform, Enterprise Edition – standardizovaná platforma určená pro vývoj přenositelných, robustních, škálovatelných a bezpečných serverových aplikací v jazyce Java.

nou architekturu pro podporu tvorby distribuovaných objektově orientovaných aplikací. Model je nezávislý na operačním systému, programovém vybavení nebo programovacím jazyku.

- *OSGi Service Platform* – je specifikace dynamického modulárního systému pro programovací jazyk Java.

2.1.4 Komponentový framework

Komponentový framework je implementací komponentového modelu, respektive služeb, které daný komponentový model vynucuje [3]. Komponentový framework řídí životní cyklus komponent a zprostředkovává jejich vzájemné interakce. Jeden komponentový framework může implementovat více komponentových modelů.

Komponentový framework lze přirovnat k malému operačnímu systému. V této analogii jsou komponenty pro framework to samé, co jsou procesy pro operační systém. Komponentový framework řídí zdroje sdílené komponentami, poskytuje základní mechanizmy umožňující komunikaci mezi komponentami a řídí životní cyklus komponent [3].

2.1.5 Kompozice

Kompozice je termín popisující, jak jsou skládány systémy při komponentově orientovaném vývoji.

Z obrázku 2.1, na kterém je znázorněn princip komponentově orientovaného systému, lze identifikovat dva subjekty, které jsou součástí kompozice: komponenty a frameworky. Díky tomu existují tři hlavní kategorie interakcí vznikajících v CBD [3]:

- *Komponenta-komponenta* – Kompozice, která umožňuje interakce mezi komponentami. Tyto interakce dodávají aplikační funkcionalitu. Kon-

trakty specifikující tyto interakce jsou klasifikovány jako kontrakty aplikační úrovně.

- *Framework-komponenta* – Kompozice, která umožňuje interakce mezi komponentovým frameworkem a jeho komponentami. Tyto interakce dovolují frameworku řídit komponentové zdroje. Kontrakty specifikující tyto interakce jsou klasifikovány jako kontrakty systémové úrovně.
- *Framework-framework* – Kompozice, která umožňuje interakce mezi frameworky. Tyto interakce umožňují kompozici komponent nasazených v různých frameworcích. Tyto kontrakty jsou klasifikovány jako kontrakty mezioperační úrovně.

2.2 Platforma OSGi

V následující části je popsána technologie OSGi. Většina textu vychází ze: [4] a [10].

2.2.1 Modularita v Javě

Již bylo řečeno, že v dnešní době je kladen velký důraz na modularitu aplikací. V programovacím jazyce Java jsou však tyto možnosti značně omezené. Aplikace vytvořené v Javě jsou typicky distribuovány v JAR[3] archivech, což je souborový formát založený na ZIP kompresi, který umožňuje sloučit několik souborů do jednoho. Jsou v něm obsaženy především soubory typu `class`, metadata a případné další zdroje pro účely aplikace jako obrázky nebo textové dokumenty.

JAR soubory obvykle poskytují jednu knihovnu nebo pouze část funkcionality aplikace. Jen zřídka aplikaci tvoří pouze jeden JAR archiv, rozsáhlé

[3]Souborový formát JAR – akronym vytvořený z „Java archive“. Soubory mají příponu `.jar`. Více na [8].

aplikace se mohou skládat z desítek až stovek těchto souborů [4]. Je zřejmé, že kompozice tak velkého rozsahu je náročná na správu a řízení, avšak Java sama o sobě nenabízí uspokojivé nástroje a technologie ke splnění požadavků komponentově orientovaného programování tak, jak jsme je popsali v předchozích kapitolách.

Zde je přehled nejzásadnějších nedostatků[4] spojených s použitím JAR souborů jako jednotek nasazení [4]:

- Nedostatky načítání zdrojů z classpath, možnosti konfliktů v případech, kdy se v aplikaci vyskytuje více tříd se stejným jménem. Classloader vždy načte první verzi třídy daného jména, na kterou narazí.
- Neobsahují metadata ukazující jejich závislosti.
- Slabá podpora verzování, několik různých verzí nemůže být spuštěno současně.

Z výše uvedených důvodu vznikla pro platformu Java specifikace OSGi, která tyto problémy odstraňuje.

2.2.2 Architektura OSGi

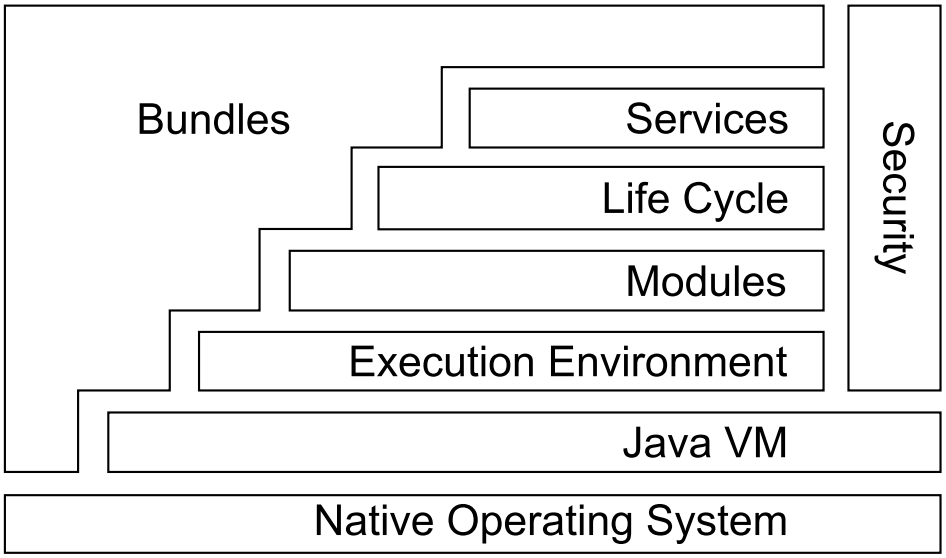
Technologie OSGi je specifikace dynamického modulárního systému pro platformu Java, která implementuje dynamický komponentový model. Standard je definován a udržován mezinárodním konsorciem OSGi Alliance založeným v roce 1999. Původně zkratka OSGi znamenala Open Services Gateway initiative, dnes se však již nepoužívá a „OSGi“ se stalo ochrannou známkou.

OSGi umožňuje instalaci, spouštění, aktualizaci a odebírání modulů za běhu, tedy bez nutnosti zastavení JVM (Java Virtual Machine). Dále pak definuje životní cyklus modulu a nabízí infrastrukturu pro spolupráci modulů

[4]Tyto nedostatky jsou tak značné, že vzniklo pojmenování „JAR hell“.

skrze služby. Implementací specifikace tohoto modulárního systému je OSGi framework, viz 2.2.5.

Na obrázku 2.3 je znázorněna architektura OSGi a její jednotlivé vrstvy.



Obrázek 2.3: Vrstvy v OSGi [9]

Následuje výčet vrstev s jejich popisem [10]:

- **Bundles** – bundly jsou OSGi komponenty vytvořené programátory.
- **Services** – vrstva služeb dynamicky propojuje vytvořené bundly – zajišťuje komunikaci bundlů.
- **Life Cycle** – vrstva starající se o životní cyklus bundlů – instalaci, spuštění, zastavení, update a jejich odinstalování.
- **Modules** – je vrstva starající se o zapouzdření bundlů a definující, jak může bundle exportovat a importovat kód.
- **Execution Environment** – definuje, jaké metody a třídy konkrétní platforma nabízí.
- **Security** – vrstva zabývající se aspekty bezpečnosti.

2.2.3 Komponenty v OSGi

V souvislosti se specifikací OSGi nehovoříme o komponentě nebo modulech, ale o tzv. *bundle*. Bundle je reprezentován JAR souborem, který musí obsahovat manifest s informacemi o bundlu. V manifestu se nacházejí speciální hlavičky obsahující informace, které framework potřebuje k úspěšné instalaci a spuštění bundlu. Manifest se nachází v JAR archivu na obvyklé cestě: `META-INF/MANIFEST.MF`. Prostřednictvím manifestu lze také nastavit, jaké služby budou dostupné ostatním bundlům. Následující seznam obsahuje některé typické hlavičky manifestu OSGi bundlu:

- *Bundle-Name*: Jméno bundlu (srozumitelné člověku).
- *Bundle-SymbolicName*: Jedinečný identifikátor bundlu. Je to jediná povinná hlavička manifestu.
- *Bundle-Description*: Krátký popis funkcionality bundlu.
- *Bundle-ManifestVersion*: Určuje verzi OSGi, která se má použít.
- *Bundle-Version*: Verze bundlu.
- *Export-Package*: Seznam balíků (packages), které jsou viditelné a k dispozici ostatním bundlům.
- *Import-Package*: Seznam balíků, které bundle využívá a vyžaduje pro svoji funkčnost.

Příklad jednoduchého manifestu se nachází ve výpisu 2.1.

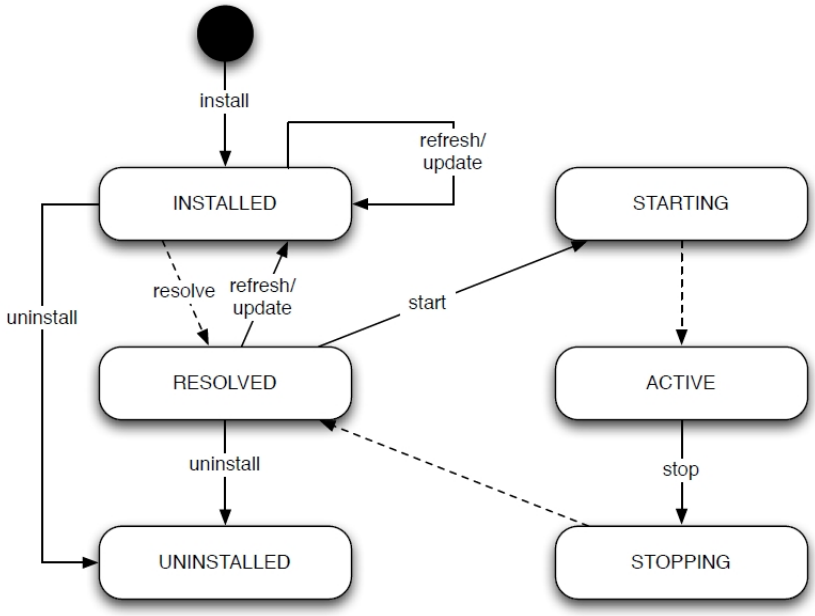
Důležitou vlastností bundlů je, že každý má svůj vlastní zavaděč tříd (classloader), což přispívá k vzájemné izolaci bundlů a tedy modularitě celé aplikace.

```
Bundle-Name: Hello World
Bundle-SymbolicName: cz.zcu.kiv.helloWorld
Bundle-Description: Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Export-Package: cz.zcu.kiv.helloWorld;version="1.0.0"
Import-Package: org.osgi.framework;version="1.3.0"
```

Výpis 2.1: Manifest OSGi bundlu

2.2.4 Životní cyklus bundlu

Vrstva Life Cycle se stará o životní cyklus bundlu, který je zobrazen na obrázku 2.4.



Obrázek 2.4: Životní cyklus bundlu [4]

Stavy, ve kterých se může bundle nacházet:

- *INSTALLED* – bundle byl úspěšně nainstalován.
- *RESOLVED* – bundle je buď připraven ke spuštění (všechny závislosti jsou splněny a zdroje k dispozici), a nebo byl zastaven.
- *STARTING* – spouštění bundlu, je zavolána metoda `start()` rozhraní `BundleActivator`.
- *ACTIVE* – bundle byl úspěšně aktivován a běží, metoda `start()` proběhla úspěšně.
- *STOPPING* – zastavování bundlu. Je zavolána metoda `stop()` rozhraní `BundleActivator`.
- *UNINSTALLED* – bundle byl odinstalován, konečný stav.

2.2.5 OSGi framework

V současné době existuje několik na sobě nezávislých open source implementací standardu OSGi [4].

- *Equinox* – je zřejmě nejrozšířenější OSGi framework [4], je součástí vývojové platformy Eclipse. Equinox implementuje specifikaci OSGi 4.1 a je licencován pod Eclipse Public License (EPL).
- *Knopflerfish* – další populární implementace OSGi, která implementuje jak OSGi 3, tak OSGi 4. Framework je vyvíjen společností Makewave a je licencován pod BSD licencí. Existuje i komerční verze Knopflerfish Pro.
- *Apache Felix* – malá a velmi kompaktní implementace OSGi 4.x. Felix je vyvíjen neziskovou společností Apache Software Foundation a licencován pod Apache License Version 2.0.

- *Concierge* – je velmi kompaktní a vysoce optimalizovaná implementace OSGi Release 3, která je určena zejména pro platformy s omezenými zdroji, jako jsou mobilní telefony nebo embedded zařízení. Concierge je licencován pod BSD licencí.

2.2.6 Služby

Moduly spolu mohou komunikovat kromě sdílení Java balíků, které je realizováno nastavováním viditelnosti v manifestu (hlavičky `Export-Package` a `Import-Package`), i pomocí služeb. Přístup ke službě je definován pomocí Java rozhraní, které představuje kontrakt mezi poskytovatelem a klientem. Poskytovatel služby registruje službu do registru služeb. Naopak klient využívá registr k vyhledání požadované služby a její následné využití. O zajištění tohoto mechanizmu se stará vrstva *Service Layer*.

2.3 Spring

Spring Framework je Java platforma, která poskytuje komplexní infrastrukturu podporující vývoj Java aplikací. Framework zajišťuje infrastrukturu a programátor se tak může soustředit na samotnou implementaci. Hlavním účelem je usnadnit vývoj v oblasti Java EE aplikací. Spring je modulární framework organizovaný do dvaceti modulů a umožňuje použít jen ty části, které programátor skutečně použije [11].

Jádro Springu je tvořeno Spring kontejnerem. Objekty, se kterými pracuje kontejner, se nazývají tzv. *beany* (anglicky beans). V terminologii komponentového programování beany lze nazvat komponentami. Framework využívá konceptu Inversion of control, viz dále.

2.3.1 Dependency Injection

Návrhový vzor Inversion of Control usnadňuje správu objektů a závislostí tím, že přenáší odpovědnost za vytváření instancí tříd, inicializaci a provázání objektů z aplikace na framework [11]. Jednotlivé objekty tříd si samy nevytvářejí ostatní objekty, na kterých závisí, ale udělá to za ně Spring. Ve Springu je tento přístup znám jako *Dependency Injection*, tedy volně přeloženo jako vkládání závislostí. Výsledkem je větší větší kontrola nad objekty, větší znovupoužitelnost aplikace a její lepší udržitelnost. Existují dva hlavní způsoby Dependency Injection:

1. *Konstruktorem* – při inicializaci je Springem zavolán konstruktor s parametrem, který je v elementu `constructor-arg`. Atribut `ref`[5] je reference na vkládaný objekt.

2. *Setterem* – vložení se provádí setterem. Parametr je definován v elementu `property`, přičemž setter musí být pojmenován ve tvaru `setName`, kde `Name` je hodnota atributu stejného jména. Reference na vkládaný objekt se nachází opět v atributu `ref`.

Oba způsoby jsou ukázány ve výpisu 2.2, jenž zobrazuje část konfiguračního `XML` souboru. Tento soubor obsahuje definice beanů a je nezbytnou součástí každé aplikace využívající Spring.

```
1 <beans>
2   <bean id="foo" class="x.y.Foo">
3       <constructor-arg ref="bar"/>
4       <property name="baz" ref="baz"/>
5   </bean>
6
7   <bean id="bar" class="x.y.Bar"/>
8   <bean id="baz" class="x.y.Baz"/>
9 </beans>
```

Výpis 2.2: Dependency Injection ve Springu

[5]Lze také vložit parametr přímo pomocí atributu `value`.

2.3.2 Spring DM (Blueprint)

V předchozích kapitolách byly popsány technologie OSGi a Spring. Spojením těchto dvou technologií vznikl komponentový model Spring DM, který dovoluje využívat výhod obou z nich.

V současné době je technologie Spring DM součástí OSGi[6] pod názvem Blueprint. Blueprint vychází z modelu Spring DM, rozdíly mezi nimi jsou nevelkého významu. Změny, které se týkají především konfiguračního souboru, jsou popsány v [12].

Spring DM, respektive Blueprint je tvořen několika OSGi bundly. Tyto bundly jsou zodpovědné za vytváření tzv. aplikačního kontextu, který obsahuje a spravuje beany. Při spojení obou technologií vnesena do aplikace nová modularita. V běžné Spring aplikaci totiž existuje pouze jeden aplikační kontext, který je sdílený pro celou aplikaci. Oproti tomu Spring DM vytváří pro každý OSGi bundle aplikační kontext svůj vlastní. To znamená, že beany v jednom bundlu nemohou přistupovat k beanům v ostatních bundlech [5].

Aby bylo v aplikaci budované s použitím OSGi možné využívat Spring, je nutné, aby bundle obsahoval Spring konfigurační `XML` soubor. Tento soubor je třeba umístit do adresáře `META-INF/spring`[7]. Druhou variantu je specifikovat umístění konfiguračního souboru přímo v manifestu, a to hlavičkou `Spring-context`. V tomto případě může být umístění libovolné [5].

Významným přínosem Spring DM je ulehčení práce se službami OSGi použitím jmenného prostoru (namespace) `osgi`. Prostřednictvím konfiguračního souboru lze nastavit, jaké služby bude bundle využívat a jaké poskytovat.

I když aktuálněji se jeví použití kontejneru Blueprint, v realizační části této práce je použit Spring DM, a proto i ukázka konfiguračního souboru ve výpisu 2.3 je uvedena dle konvencí Spring DM.

[6] A to od specifikace OSGi 4.2.

[7] V případě Blueprintu se jedná o umístění `OSGI-INF/blueprint`.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xmlns:osgi="http://www.springframework.org/schema/osgi"
5   xsi:schemaLocation="http://www.springframework.org/schema/beans
6   http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
7   http://www.springframework.org/schema/osgi
8   http://www.springframework.org/schema/osgi/spring-osgi-1.0.xsd">
9
10  <osgi:service id="settings"
11     ref="control" interface="gui.ISettings" />
12  <osgi:reference id="messanger"
13     interface="org.osgi.service.event.EventAdmin" />
14
15  <bean id="control" class="gui.Control">
16    <constructor-arg ref="messanger"> </constructor-arg>
17  </bean>
18
19 </beans>
```

Výpis 2.3: Spring DM konfigurační soubor

V ukázce 2.3 je na řádcích 10 a 11 pomocí elementu `osgi:service` zaregistrována služba `gui.ISettings`. Tato služba je tedy poskytnuta ostatním bundlům. Oproti tomu element `osgi:reference` zajišťuje, že bundle může využívat službu `EventAdmin`. Instance této služby je předána pomocí mechanizmu Dependency Injection konstruktorem.

3 Simulátor Simco

Simco je framework, který vznikl na Katedře informatiky a výpočetní techniky Fakulty aplikovaných věd Západočeské univerzity v Plzni v rámci diplomových prací Tomáše Kabíčka [13] a Matěje Prokopa [14] v roce 2011, kdy první jmenovaný vytvořil jádro frameworku a druhý pak jeho grafické rozhraní. Název Simco je odvozen z anglického *Simulation of Component*. Celý simulátor je vytvořen ze softwarových komponent, což vypovídá o jeho modulárnosti.

Účelem simulátoru je poskytnout nástroj k testování reálných softwarových komponent v simulačním prostředí bez nutnosti vytvářet jejich modely, které by mohly být potenciálně chybné. Testování probíhá za použití principu diskrétní událostní simulace s předem daným scénářem. Aplikace, která je na tento framework napojena se nazývá Simco aplikace.

Framework je tvořen čtyřmi základními stavebními prvky [13]:

- Komponenty – objekty reprezentující komponenty.
- Události – objekty reprezentace událostí simulace běhu Simco aplikace.
- Kalendář – ukládá a spravuje události, obsahuje jejich seznam, zajišťuje kontrolu nad během celé Simco aplikace. Obsahuje simulační čas, který se inkrementuje po každém kroku simulace a určuje, která událost má být vykonána.
- Simulace – objekt, kterým se manipuluje s celou simulací.

3.1 Komponenty

Jak bylo řečeno, Simco aplikace se skládá z komponent, konkrétně ze Spring DM modulů. Framework pracuje s několika druhy komponent, které je třeba rozlišit. Jednotlivé typy jsou uvedeny v následujícím výčtu [13]:

- `SIMULATION` – simulační komponenta. Neprovádí reálný výpočet, její běh je pouze nasimulovaný. Zastupuje reálný systém a vrací data, která strukturou připomínají data reálná. Uživatel může prostřednictvím konfiguračních souborů nastavit délku simulovaného výpočtu a hodnoty, které komponenta vrátí.
- `REAL` – reálná komponenta, která provádí reálný výpočet a která je testována. Neměla by vědět o tom, že je napojena na Simco framework, nedochází tedy k úpravám jejího kódu.
- `INTERMEDIATE` – komponenta typu prostředník. Je potřebná k tomu, aby mohl framework sledovat komunikaci dvou reálných komponent.
- `FRAMEWORK` – komponenty, které obsahují implementaci frameworku Simco.

3.2 Události

Hlavní princip fungování simulátoru spočívá v tom, že každý krok, který se v Simco aplikaci provede, je spojen s událostí ve frameworku. Pokud probíhá komunikace mezi komponentami, Simco framework ji zachytí, vytvoří událost a vloží ji do kalendáře. Vložená událost může způsobit vznik dalších událostí. V simulátoru mohou vzniknout následující typy události [13]:

- `REAL_CALL` – metoda reálné komponenty 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` – periodicky vyskytující se událost.

- `CASUAL` – událost, u které lze nadefinovat, s jakou pravděpodobností se bude vyskytovat.

Komponenty třetích stran lze napojit na Simco framework prostřednictvím OSGi bundlu `EventAdmin`. Ten umožňuje publikovat události mezi jednotlivými bundly. Historie průběhu simulace se ukládá do souboru. Samotné jádro frameworku se skládá z šesti komponent, jejich podrobný popis lze nalézt v [13].

3.3 Konfigurace

Simulaci lze nakonfigurovat pomocí `XML` souboru. Prostřednictvím konfiguračního souboru je možno nastavit tzv. scénář průběhu simulace. Ve scénáři lze nadefinovat události typu `CASUAL`, `REGULAR`, seznam komponent Simco aplikace s jejich vlastnostmi a nastavení Simco aplikace. Ve výpisu 3.1 je uveden příklad jednoduchého scénáře.

Kořenový element `XML` souboru má název `simCoScenario`. Seznam komponent se nachází v elementu `components`. Každá komponenta má definovaný svůj typ, jedinečné jméno[8] a cestu ke svému konfiguračnímu souboru bundlu.

V elementu `events` jsou definovány události. U každé události je uveden její typ (`REGULAR` nebo `CASUAL`), zdroj události, rozhraní, přes které je volaná služba registrována, jméno volané metody a parametry této metody. V elementu `eventDetails` je nadefinováno, jak často se má daná událost opakovat. Podle typu události je nastavena buď perioda, a nebo náhodná veličina. Více viz [13].

V elementu `appSettings` lze nastavit zpoždění kroku simulace, a to buď v milisekundách nebo v sekundách.

[8]Shodné s obsahem hlavičky `Bundle-SymbolicName` v manifestu komponenty.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <simCoScenario>
3    <components>
4      <component type="SIMULATION">
5        <symbolicName>SimcoGPS</symbolicName>
6        <settingsFile> C:/SimcoGPS/gpsSettings.xml </settingsFile>
7      </component>
8    </components>
9    <events>
10     <event type="REGULAR">
11       <source>SimcoAccelerometer</source>
12       <serviceName>simco.application.IAccelerometer</serviceName>
13       <methodName>setVector</methodName>
14       <methodParameters>
15         <parameter dataType="java.lang.Integer" value="10" />
16       </methodParameters>
17       <eventDetails>
18         <detail key="period" val="10" />
19       </eventDetails>
20     </event>
21   </events>
22   <appSettings>
23     <stepLenght unit="msec">500</stepLenght>
24   </appSettings>
25 </simCoScenario>
```

Výpis 3.1: Scénář simulace

4 Analýza

4.1 Požadavky na aplikaci

Cílem této práce je vytvořit komponentu, pomocí které bude možné vytvořit dynamicky generované grafické uživatelské rozhraní. Toto GUI bude sloužit k usnadnění ovládání testování komponent. Vyvinutá komponenta bude mít dvě základní funkce:

1. nastavování komponent,
2. sledování komponent.

Nastavovací část umožní odeslat data do simulace, a tím nastavovat jejich parametry. Druhá část funkcionality spočívá v možnosti sledovat data komponent. Zde bude vytvářená komponenta data přijímat, zobrazovat a ukládat.

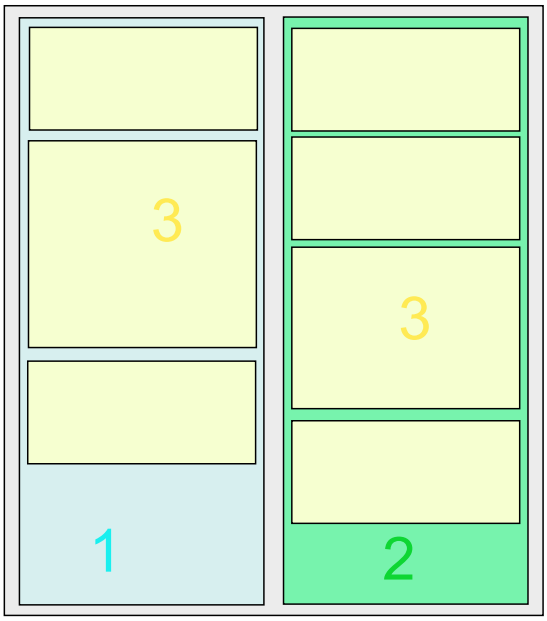
4.2 Použité prostředky

Vytvářená komponenta bude nasazena jako samostatný modul do simulátoru Simco (viz kapitola 3). Simulátor je složen ze Spring DM bundlů. Z toho vyplývá, že vytvářená komponenta bude implementována v programovacím jazyce Java. Dále bude využit framework OSGi a Spring, respektive technologie využívající výhod obou z nich – Spring DM.

K implementaci bude použito vývojové prostředí Eclipse Java EE 4.2, framework Spring DM 3.1, programovací jazyk Java 1.7 a operační systém Windows 7 Professional.

4.3 Návrh prostředí

Okno aplikace je rozděleno na dvě části. V levé části (1) se nachází panel určený k odesílání dat do simulace. V pravé části (2) jsou pak umístěny sledovací prvky. Obě části jsou umístěny do okna za použití layoutu `BoxLayout`. Tento layout řadí komponenty podle nastavení vedle sebe do řádku, a nebo pod sebe do sloupce. V našem případě je použita horizontální konfigurace. Jednotlivé nastavovací nebo sledovací elementy (3) jsou pak uvnitř panelů (1) a (2) rozmist'ovány pod sebe pomocí layoutu `GridBagLayout`. Rozvržení okna je znázorněno na obrázku 4.1.



Obrázek 4.1: Rozvržení grafického uživatelského rozhraní

4.4 Dynamické GUI

Z výše uvedeného vyplývá, že GUI nebude mít pokaždé stejnou podobu, ale bude obsahovat prvky, o které si programátor komponent zažádá. Definování toho, co má GUI obsahovat, se provádí prostřednictvím API, které tato komponenta poskytuje.

Vytvářená komponenta umožní sledované komponentě zaregistrovat si proměnné, které budou sledovány. Tato data bude možné zobrazovat jak v textové podobě, tak v grafické ve formě grafu. Dále bude umožněno přijímat zprávy informující o průběhu simulace, třídit je podle druhů, ukládat je do souborů. Oproti tomu nastavovací část API poskytne služby, kterými budou data odesílána do simulace, a tím bude umožněno komponentu řídit, respektive nastavovat její parametry.

5 Implementace

5.1 Poskytované API

Středobodem komunikace s vnějšími komponentami je rozhraní `ISettings`. Rozhraní poskytuje programátorovi simulované komponenty sadu metod, pomocí kterých se vytvoří okno nastavovacího modulu a jeho jednotlivé elementy. Zajišťuje základní komunikaci mezi uživatelským rozhraním a komponentami. Prostředků API je potřeba jednak k vytvoření ovládacího a sledovacího GUI, a jednak k nastavování dat uvnitř řízené komponenty, tedy k získání uživatelem zadané hodnoty.

5.1.1 Vytvoření okna

public ControlGUI createWindow(String windowName);

První metoda, kterou programátor komponenty použije. Touto metodou je spuštěna inicializace prázdného okna. Oken je z principu možné vytvořit více (pro více komponent), je tedy důležité rozlišit, v jakém okně se prvek má vytvořit. K tomu slouží návratová hodnota této metody, která je použita jako parametr v dalších metodách tohoto rozhraní. Jediný parametr metody je celočíselné ID panelu, které si určí programátor. To slouží jednak k rozlišení jednotlivých oken uživateli – ID je vidět v záhlaví okna, a jednak při přijímání dat z komponenty, která okno používá.

5.2 Nastavovací prvky

5.2.1 Nastavení celočíselné proměnné

```
public void createIntSet(ControlGUI refGui, String label,
        int messageType, int min, int max, int step);
```

Metoda, která v okně vyrobí blok, ve kterém je možné nastavit celočíselnou proměnnou. Nejdůležitějším prvkem bloku je Swing komponenta `JSpinner`. Spinner slouží k přijetí celočíselné od uživatele, přičemž parametry `min` a `max` je omezen rozsah spinneru. Parametr `step` určuje krok spinneru. Současně je vytvořeno tlačítko, kterým se zadaná hodnota odešle řízené komponentě. Hodnota parametru `messageType` je druh (kód) zprávy, která je odeslána do simulace po stisknutí tlačítka „OK“. Parametr `label` obsahuje název proměnné, která je zaregistrována, a konečně parametr `refGui`, který zajistí vykreslení jednotlivých prvků do správného okna.

```
public void createIntSet(ControlGUI refGui, String label,
        int messageType);
```

Metoda je přetížená, je možné ji zavolat bez parametrů `min, max` a `step`.

5.2.2 Nastavení reálné proměnné

```
public void createDoubleSet(ControlGUI refGui, String label,
    int messageType, double min, double max, double step);
```

Metoda, která v okně vyrobí blok, ve kterém je možné nastavit reálnou proměnnou. Význam jednotlivých parametrů je totožný s metodou `createIntSet`, pouze se zde pracuje s datovým typem `double`.

```
public void createDoubleSet(ControlGUI refGui, String label,
        int messageType);
```

Opět je možné využít přetížení metod a volat metodu bez nastavení spinneru.

5.2.3 Nastavení textové proměnné

```
public void createTextSet(ControlGUI refGui, String label,
        int messageType);
```

Metoda umožňuje nastavit a odeslat proměnnou typu `String`. Parametry jsou reference na správné okno, popisek zobrazený v okně a kód proměnné.

5.2.4 Nastavení proměnné typu pole

```
public void createFieldSet(ControlGUI refGui, String label,
        int messageType);
```

Tato metoda umožňuje nastavit a odeslat více hodnot najednou. Po jejím zavolání je v okně vytvořena tabulka se dvěma sloupci, do které lze vkládat jak číselné, tak řetězcové hodnoty. Zároveň je v tomto bloku vyrobeno tlačítko, jehož stiskem je přidán do tabulky řádek. Odeslaná data reprezentuje objekt třídy `DefaultTableModel`. Parametry metody jsou: odkaz na okno, kde se má blok vykreslit, popisek bloku a kód generované zprávy při odeslání dat z okna.

5.3 Sledovací prvky

5.3.1 Sledování proměnné

```
public void observeVariable(ControlGUI refGui, String name,
        int code);
```

Metoda vytvoří v okně blok pro přijímání hodnot registrované proměnné z řízené komponenty. Sledovací blok obsahuje dvě needitovatelná textová pole, která jsou reprezentována Swing komponentou `JTextArea`. V prvním poli je zobrazována aktuálně přijatá hodnota z řízené komponenty, ve druhém pak celá její historie. Dále jsou zde umístěna dvě tlačítka. První tlačítko „GET“ slouží uživateli k aktivnímu zažádání komponenty o aktuální hodnotu sledované proměnné. Stiskem tlačítka je vygenerována zpráva typu *žádost o zaslání*

hodnoty. Je však na uživateli, zda ve svém kódu tuto zprávu sleduje a jestli na ni reaguje. Druhé tlačítko „GRAPH“ zobrazí graf z historie přijatých hodnot.

Parametr `refGui` je odkaz na okno, ve kterém se má sledovací blok vykreslit. Popisek sledovacího bloku zobrazený v GUI, je nastavený parametrem `name`. Celočíselná hodnota `code`, je jedinečný kód proměnné, kterou si uživatel zaregistruje. Podle hodnoty tohoto parametru je zjištěno, kterému sledovacímu elementu je přijatá hodnota z řízené komponenty určena. Mechanizmus, kterým se posílají data mezi komponentami a okny, je popsán v kapitole 5.4.

5.3.2 Logovací komponenta

```
public void createLog(ControlGUI refGui, String[] levels,
                      int code);
```

Metoda vytvoří textovou oblast, která funguje jako přijímač zpráv o běhu simulace z řízené komponenty. Zobrazení dat je realizováno Swing komponentou `JTable`. Při vytváření této komponenty uživatel definuje možné úrovně zpráv, které chce rozlišovat. Zpráva odesílaná z řízené komponenty by měla obsahovat čas odeslání, úroveň zprávy a samotný text zprávy.

Logovací komponenta dále obsahuje tlačítko „Save“, kterým lze jeho obsah uložit do souboru. Součástí je také panel, který obsahuje zaškrtávací pole, tzv. *checkboxy*. Uživatel může těmito přepínači filtrovat obsah tabulky podle úrovně zprávy. Parametr `refGui` je odkaz na okno, do kterého se logovací blok vykreslí, pole řetězců `levels` obsahuje možné úrovně zpráv. Parametr `code` je opět kód, pod kterým si řízená komponenta zaregistrovala tento sledovací prvek.

5.4 Validace okna

```
public void done(ControlGUI refGui);
```

Poté, co si uživatel voláním výše uvedených metod nakonfiguruje požadované GUI, je nutné jako poslední zavolat metodu `done(ControlGUI)`. Voláním této metody je okno zvalidováno a správně vykresleno. Parametr je reference okna, které má být zobrazeno.

5.5 Komunikace

V předchozí kapitole bylo popsáno, jakým způsobem je generováno grafické rozhraní. Nyní bude vysvětleno, jak probíhá vlastní komunikace mezi komponentou a vygenerovaným oknem, tedy zasílání nastavených dat do řízené komponenty a přijímání sledovaných dat z druhé strany.

Existují dva základní způsoby komunikace mezi komponentami – volání metod rozhraní, které komponenta poskytuje, a využití správce událostí, který dokáže předat zprávy registrovaným komponentám. Voláním metod rozhraní API je vytvořeno okno grafického rozhraní. Odesílání nastavovacích hodnot do simulace a příjem sledovaných parametrů však probíhá druhým způsobem.

5.5.1 Zasílání zpráv

K tomu, aby komponenta mohla využívat mechanizmus předávání zpráv prostřednictvím správce událostí, je nutné mít k dispozici službu `EventAdmin`, která je součástí bundlu *org.eclipse.equinox.event*. Událost je objekt třídy

`Event`, který sestává ze dvou atributů:

- `topic` – topic je předmět zasílané zprávy. Díky tomuto atributu je možné sledovat pouze ten druh událostí, které chceme.
- `properties` – prostřednictvím tohoto atributu jsou přenášená data. Data jsou uložena v asociativním poli `HashMap`. Prvek tvoří dvojice klíč – hodnota. Právě klíč obsahuje kód proměnné nebo prvku uživatelského rozhraní, kterému je zpráva určena.

V případě naší knihovny existují tři předměty zpráv:

1. `Setting_topic` – zprávy tohoto typu přenášejí z vygenerovaného okna uživatelem odeslané hodnoty do sledované komponenty. Kód nastavované proměnné a její hodnotu obsahuje atribut `properties`.
2. `Value_request_topic` – z vygenerovaného okna je odeslána komponentě žádost o zaslání hodnoty sledované proměnné. Kód žádané proměnné a panelu je přenášen atributem `properties`.
3. `Value_send_topic` – simulovaná komponenta odesílá hodnotu sledované proměnné k zobrazení.

5.5.2 Konfigurační soubor

Aby bylo možné využívat API, je třeba zajistit aby komponenty obdržely reference na příslušné služby. Aby mohla simulovaná komponenta vzdáleně vygenerovat nastavovací GUI, je třeba do jejího konfiguračního souboru uvést následující řádek:

<osgi:reference id=”Interface” **interface**=”service.ISettings” />

Reference na rozhraní je startovací metodě komponenty předáno pomocí Springu a jeho dependency injection. Komponenta poté může využívat služeb rozhraní `ISetttings`.

Následující část konfiguračního souboru říká, že komponenta bude naslouchat událostem s topicem „Setting_topic“. Objekt `EventHandler`, který to umožňuje, je opět předán s využitím DI.

```
<osgi:service id="Listen1" ref="InstanceSimul"
        interface="org.osgi.service.event.EventHandler">
        <osgi:service-properties>
                <entry key="event.topics" value="Setting_topic" />
        </osgi:service-properties>
</osgi:service>
```

Příklad celého konfiguračního souboru je uveden v příloze A.1.

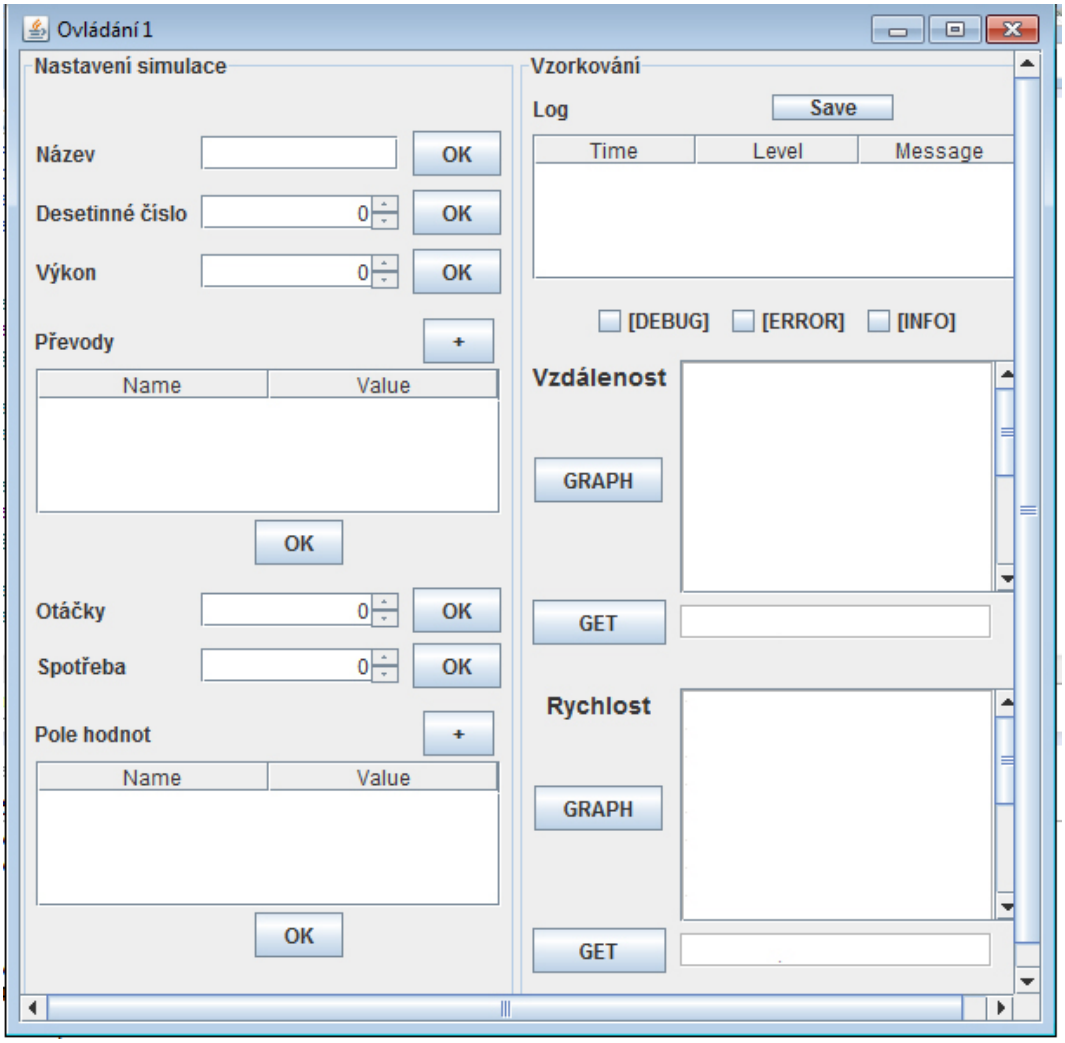
5.6 Ukázka použití

Na následujícím výpisu je příklad použití vytvářené knihovny. Tento úsek kódu vygeneruje GUI, které je zobrazeno na obrázku 5.1.

```
1  ControlGUI control1 = refInterface.createWindow(1);
2
3  refInterface.createTextSet(control1, "Nazev", 1);
4  refInterface.createDoubleSet(control1,
5          "Desetinne cislo", 2, -5.00, +5.00, 0.05);
6  refInterface.createDoubleSet(control1, "Vykon", 3);
7
8  refInterface.createFieldSet(control1, "Prevody", 4);
9  refInterface.createIntSet(control1, "Otacky", 5);
10 refInterface.createDoubleSet(control1, "Spotreba", 5);
11 refInterface.createFieldSet(control1, "Pole hodnot", 6);
12
13 refInterface.createLog(control1, urovneZprav, 7);
14
15 refInterface.observeVariable(control1, "Vzdalenost", 8);
16 refInterface.observeVariable(control1, "Rychlost", 9);
17
18 refInterface.done(control1);
```

Výpis 5.1: Příklad vytváření GUI

V rámci zpřehlednění kódu je vhodné nadefinovat sadu konstant, které budou reprezentovat jednotlivé signály.



Obrázek 5.1: Příklad vygenerovaného okna

6 Závěr

V rámci této práce byla implementována knihovna, která na základě požadavků programátora vytvoří grafické uživatelské rozhraní, jehož prostřednictvím je umožněno nastavovat parametry jednotlivých komponent v simulaci a zároveň sledovat proměnné a parametry jiných komponent a s těmito daty dále manipulovat.

Vítaným vylepšením funkčnosti by byla možnost zobrazovat sledovaná data „online“ pomocí dynamického grafu.

Přehled zkratek

GUI – Graphical user interface
OSGi – Open Service Gateway initiative
CBSE – Component-based software engineering
EJB – Enterprise JavaBeans
Java EE – Java Platform, Enterprise Edition
COM – Component Object Model
CORBA – Common Object Request Broker Architecture
OMG – Object Management Group
CBD – Component-based development
JAR – Java archive
JVM – Java Virtual Machine
BSD – Berkeley Software Distribution
XML – Extensible Markup Language
Spring DM – Spring Dynamic Modules
API – Application Programming Interface

Literatura

[1] SZYPERSKI, Clemens. *Component software: beyond object-oriented programming*. 1st ed. repr. Harlow, England: Addison-Wesley, 1998. ISBN 0-201-17888-5.

[2] KIV Wiki. *Úvod do komponent* [online]. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra informatiky a výpočetní techniky. 2011 [cit. 3/2014]. Dostupné z WWW: `http://wiki.kiv.zcu.cz/UvodDoKomponent/HomePage`

[3] BACHMANN, Felix et al. *Volume II: Technical concepts of component-based software engineering* [online]. Carnegie Mellon University, Software Engineering Institute. 2000 [cit. 3/2014]. Dostupné z WWW: `http://resources.sei.cmu.edu/asset_files/TechnicalReport/2000_005_001_13715.pdf`

[4] BARTLETT, Neil. *OSGi In Practice* [online]. Amazon.com, 2009. Dostupné z WWW: `http://freecomputerbooks.com/OSGi-In-Practice.html`

[5] WALLS, Craig. *Modular Java: creating Flexible Applications with OSGi and Spring*. 1st edition. USA: The Pragmatic Bookshelf, 2009. ISBN 1-934356-40-9.

[6] D'SOUZA, Desmond. *Objects, Components, and Frameworks with UML: The Catalysis^{TM}Approach* [online]. Addison-Wesley Professional. 1998 [cit. 3/2014]. Dostupné z WWW: `http://www.catalysis.org/publications/Benelux-new.pdf`

[7] FI WIKI. *Enterprise JavaBeans* [online]. Fakulta informatiky Masarykovy univerzity. 2013 [cit. 4/2014]. Dostupné z WWW: http://kore.fi.muni.cz/wiki/index.php/Enterprise_JavaBeans

[8] ORACLE. *JAR File Specification* [online]. Dostupné z WWW: http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html

[9] The OSGi Alliance. *The OSGi Architecture* [online]. Dostupné z WWW: http://www.osgi.org/Technology/WhatIsOSGi

[10] The OSGi Alliance. *OSGi Core Release 5* [online]. 2012 [cit. 4/2014]. Dostupné z WWW: http://www.osgi.org/download/r5/osgi.core-5.0.0.pdf

[11] JOHNSON, Rod et al. *Spring Framework Reference Documentation* [online]. 2013 [cit. 5/2014]. Dostupné z WWW: http://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/

[12] SpringSource. *OSGi 4.2 Blueprint Container* [online], [cit. 6/2014]. Dostupné z WWW: http://docs.spring.io/osgi/docs/2.0.0.M1/reference/html/blueprint.html

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

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

A Přílohy

A.1 Konfigurační soubory

```
<?xml version=”1.0” encoding=”UTF−8”?>
<beans xmlns=”http://www.springframework.org/schema/beans”
  xmlns:xsi=”http://www.w3.org/2001/XMLSchema−instance”
    xmlns:osgi=”http://www.springframework.org/schema/osgi”
        xsi:schemaLocation=”http://www.springframework.org/schema/beans
            http://www.springframework.org/schema/beans/spring−beans−2.0.xsd
        http://www.springframework.org/schema/osgi
        http://www.springframework.org/schema/osgi/spring−osgi−1.0.xsd”>

   <osgi:service id=”Interface” ref=”Control” interface=”service.ISettings” />
   <osgi:reference id=”Messanger” interface=”org.osgi.service.event.EventAdmin”/>

        <bean id=”Control” name=”ControlWindow” class=”tvorbaGUI.Control”>
                <constructor−arg ref=”Messanger”> </constructor−arg>
        </bean>

        <osgi:service id=”Listening_to_value” ref=”Control”
                interface=”org.osgi.service.event.EventHandler”>
                <osgi:service−properties>
                        <entry key=”event.topics” value=”Value_send_topic” />
                </osgi:service−properties>
        </osgi:service>
</beans>
```

Výpis 1: Konfigurační soubor ovládací komponenty

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:osgi="http://www.springframework.org/schema/osgi"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
      http://www.springframework.org/schema/beans/spring-beans-2.0.xsd
      http://www.springframework.org/schema/osgi
      http://www.springframework.org/schema/osgi/spring-osgi-1.0.xsd">


  <osgi:reference id="Interface" interface="service.ISettings" />
  <osgi:reference id="Messanger" interface="org.osgi.service.event.EventAdmin"/>

       <bean id="InstanceSimul" name="SimulKomponenta"
                  class="simulKomponenta.Sim" init-method="start">

            <property name="refInterface" ref="Interface" />
            <property name="messanger" ref="Messanger" />
       </bean>

       <osgi:service id="Listen1" ref="InstanceSimul"
            interface="org.osgi.service.event.EventHandler">
            <osgi:service-properties>
                 <entry key="event.topics" value="Setting_topic" />
            </osgi:service-properties>
       </osgi:service>

       <osgi:service id="Listen2" ref="InstanceSimul"
            interface="org.osgi.service.event.EventHandler">
            <osgi:service-properties>
                 <entry key="event.topics" value="Value_request_topic" />
            </osgi:service-properties>
       </osgi:service>
</beans>
```

Výpis 2: Příklad konfiguračního souboru sledované komponenty