VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

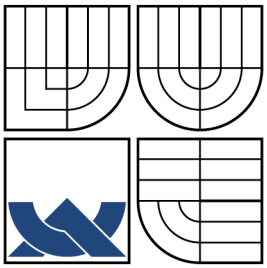
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

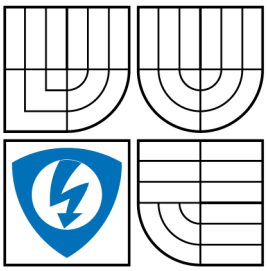
ROZHRANÍ PRO VZDÁLENÝ PŘÍSTUP K PROSTŘEDÍ MATLAB

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. JIŘÍ ŠTEFEK
AUTHOR



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

ROZHRANÍ PRO VZDÁLENÝ PŘÍSTUP K PROSTŘEDÍ MATLAB

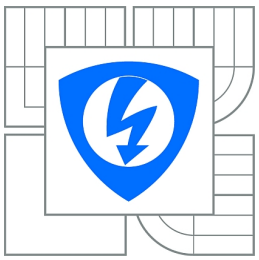
INTERFACE FOR REMOTE ACCESS TO MATLAB ENVIRONMENT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. JIŘÍ ŠTEFEK
AUTHOR

VEDOUCÍ PRÁCE — ING. JIŘÍ MEKYSKA
SUPERVISOR

BRNO 2011



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student: Bc. Jiří Štefek ***ID:*** 78724
Ročník: 2 ***Akademický rok:*** 2010/2011

NÁZEV TÉMATU:

Rozhraní pro vzdálený přístup k prostředí MATLAB

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je rozbor aktuální problematiky komunikace programovacího jazyka Java s prostředím MATLAB a následný návrh a implementace systému vzdálené komunikace s tímto prostředím. Systém bude pomocí webového rozhraní zpracovávat požadavky uživatelů, spouštět patřičný kód na serveru či distribuovaných stanicích a vracet výsledky v textové a grafické podobě či v podobě binárních souborů. Systém na serveru bude možné rozšiřovat pomocí modulů určených ke konkrétnímu zpracování multimediálních dat.

DOPORUČENÁ LITERATURA:

[1] WALLS, Craig; BREIDENBACH, Ryan . Spring in Action: Second Edition. [s.l.]: Manning Publications Co., 2008. ISBN 1-933988-13-4.
[2] BAUER, Christian; KING, Gavin. Java Persistence with Hibernate: Second Edition of Hibernate in Action. [s.l.]: Manning Publications Co., 2006. 880 s. ISBN 1-932394-88-5.
[3] HEROUT, Pavel. Učebnice jazyka Java: Třetí rošířené vydání. [s.l.]: KOPP, 2008. ISBN 987-80-7232-323-4.

Termín zadání: 7.2.2011 ***Termín odevzdání:*** 26.5.2011

Vedoucí práce: Ing. Jiří Mekyska

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Cílem této práce je rozbor komunikačních možností programovacího jazyka Java s prostředím MATLAB a následný návrh a implementace systému využívajícího tuto komunikaci. Práce se nejdříve zaměřuje na zpřehlednění a srovnání metod přístupu k prostředí MATLAB. Dále je zde navržena webová aplikace zprostředkující vzdálené výpočty v prostředí MATLAB využívající nejefektivnější spojovací metodu z minulého bodu. Následně je uveden krátký popis aplikačního rámce Spring, kterého je využito při implementaci aplikace. Poslední část se věnuje popisu možností, instalací a konfigurací systému.

KLÍČOVÁ SLOVA

MATLAB, Java, Spring, JMatlink, MatlabControl, JAMAL.

ABSTRACT

The goal of this thesis is to analyse communication possibilities between Java programming language and MATLAB enviroment and then to design and implementate a system using this communication. First of all this thesis focuses on overview and comparison of methods accesing MATLAB enviroment. In the next step there is a design of the system that mediates remote computations in MATLAB enviroment using the most effective method from previous step. Next chapter leaves a short description of Spring framework, which is used in implementation of application. The last step folows description of possibilities, instalation and configuration of the system.

KEYWORDS

MATLAB, Java, Spring, JMatlink, MatlabControl, JAMAL.

ŠTEFEK, Jiří *Rozhraní pro vzdálený přístup k prostředí MATLAB*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2011. 60 s. Vedoucí práce byl ING. Jiří Mekyska

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Rozhraní pro vzdálený přístup k prostředí MATLAB“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

Brno ....

(podpis autora)

Poděkování

Hluboce děkuji vedoucímu diplomové práce Ing. Jiřímu Mekyskovi za důkladné a věcné konzultace, rady při zpracování diplomové práce a vždy trpělivý přístup. Dále děkuji všem lidem za rady a opravy při zpracování této práce.

V Brně dne: . . . .

(podpis autora)

OBSAH

Úvod **10**

1 Spojení programovacího jazyka Java a prostředí MATLAB **11**
1.1 Úvod . . . 11
1.2 Volání prostředí MATLAB z jazyka Java . . . 11
1.2.1 Využití třídy Java *Runtime* . . . 11
1.2.2 Využití JNI a MATLAB C-Engine . . . 13
1.2.3 Využití JMI . . . 15
1.2.4 MATLAB Builder JA . . . 16

2 Srovnání metod **17**
2.1 Rozhraní IMatlabConnection . . . 17
2.2 Projekt JMatLink . . . 18
2.3 Projekt MatlabControl . . . 19
2.4 Projekt JAMAL . . . 20
2.5 Testování metod . . . 20

3 Návrh systému **23**
3.1 Uživatelé . . . 24
3.2 Webové rozhraní . . . 24
3.3 Výpočetní servery . . . 25

4 Aplikační rámec Spring **26**
4.1 Úvod . . . 26
4.2 Spring Webflow . . . 28
4.3 Acegi Security . . . 29

5 Implementace systému **30**
5.1 Instalace a konfigurace . . . 30
5.1.1 Webový server . . . 31
5.1.2 Výpočetní servery . . . 33
5.2 Popis funkcí . . . 35
5.2.1 Tvorba modulů . . . 37
5.2.2 Vstupní a výstupní formát čísel . . . 38
5.2.3 Jazyková lokalizace . . . 39
5.2.4 Spuštění první analýzy . . . 39

6 Závěr **42**

Literatura **44**

Seznam symbolů, veličin a zkratek **46**

Seznam příloh **47**

A Obsah CD **48**

B Adresářová struktura projektu **49**

C Zdrojové kódy použité pro testování **50**

C.1 JAMAL 50
C.2 MatlabControl 52
C.3 Měření uplynulého času 54
C.4 Rozhraní pro analýzy 55
C.5 Třída pro analýzu 55
C.6 Element analýzy 56
C.7 Vstupní a výstupní elementy 56
C.8 Spojovací třída 56
C.9 Měření úkonů 58
C.10 Testovací analýzy 59
C.11 Testovací třída 60
C.12 Analýza Dvojnasobek 60

SEZNAM OBRÁZKŮ

1.1 Část výpisu metod rozhraní Java-to-Matlab Interface. 15
2.1 Srovnání metod. 21
3.1 Schéma aplikace. 23
5.1 JmcServerGui po inicializaci konektoru. 37
5.2 Webové rozhraní – výběr analýzy. 40
5.3 Schéma běhu analýzy. 41

SEZNAM TABULEK

1.1 Argumenty pro spuštění MATLABu. 13
1.2 Příkazy knihovny MATLAB C-Engine. 14
3.1 Přehled rolí a jejich pravomocí. 24
4.1 Přehled stavů ve Webflow. 28
5.1 Přehled významu adresářů. 33
5.2 Přehled atributů konektorů. 34
5.3 Přehled atributů kořenového elementu. 36
5.4 Přehled atributů vstupních a výstupních elementů. 36

ÚVOD

Prostředí MATLAB ([11]) zajišťuje podporu komunikace s různými programovacími jazyky, programy a rozhraními. Tím se nabízí tvorba komplexnějšího systému zprostředkující jakkoliv přijaté, či dodatečně upravené výsledky získané výpočetním jádrem MATLABu. Toto prostředí umí komunikovat s různorodým rozhraním a to přes sériový port až například k ethernetovému rozhraní. Některé druhy komunikace, však mohou být trochu těžkopádné a proto je nabízena spolupráce s jinými programovacími jazyky. Konkrétně umí využívat jazyky C, C++, Java , Fortran, ActiveX a .NET.

Programovací jazyk Java ([4], [6]) patří k jednomu z nejpoužívanějších objektově orientovaných programovacích jazyků. U programátorů si získal oblibu zejména pro svou jednoduchost a robustnost. Neméně významnou složkou je i podpora vláknových aplikací, podpora vzdáleného volání procedur a dokonce i spolupráce s jinými programovacími jazyky.

Cílem této práce je přiblížit a popsat komunikaci programovacího jazyka Java s prostředím MATLAB a navrhnout systém, který tyto dva elementy spojí do jednoho. Práce se zabývá jednotlivými rozhraními pro komunikaci a snaží se popsat jejich kladné i záporné stránky. Cílem tohoto porovnání je vybrat nejvhodnějšího kandidáta na nasazení do systému, který dokáže zpracovávat uživatelské potřeby spojené s vykonáním patřičné analýzy v prostředí MATLAB a prezentovat takto dosažené výsledky.

Sjednocené poznatky o možnostech zkoumané komunikace jsou k nalezení v první kapitole. Práce se odvolává na nalezené druhy komunikace a u všech typů odhaluje jejich potenciál k tvorbě potřebné aplikace.

Následující kapitola se zabývá porovnáváním těchto druhů komunikací. Vybrané racionální druhy komunikace se podrobí jednoduchým testům. K testovacím účelům a pro různorodost zkoumaných aplikací je navrženo jednotné rozhraní a hned několik tříd pro analýzu jejich výkonu a jiných potenciálů.

Samotný návrh systému je situován ve třetí kapitole. S ohledem na minimální požadovanou konfiguraci klientských stanic systému, možností distribuování výpočtů a jednoduché rozšiřování o moduly je zde navrženo schéma aplikace.

V další sekci je zběžně rozebraný aplikační rámec Spring. Tento rámec je příliš komplikovaný pro podrobný popis, proto je zde uveřejněn jen informativní popis základních funkcí tak, aby čtenář pochopil alespoň z části funkcionalitu konfiguračních souborů použitých ve výsledné aplikaci.

Poslední část se snaží poskytnout popis systému. Mezi záběr této kapitoly patří instalace a konfigurace systému a popis některých důležitých funkcí implementované aplikace. Je zde také uveden návod na první spuštění aplikace.

1 SPOJENÍ PROGRAMOVACÍHO JAZYKA JAVA A PROSTŘEDÍ MATLAB

1.1 Úvod

Programovací jazyk Java patří k jednomu z nejpoužívanějších programovacích jazyků. Jedná se o jazyk objektově orientovaný. Svou oblibu si získal především robustností, jednoduchostí, podporou pro distribuovanost a snadnou implementací vláknových aplikací a jejich využití k paralelizaci výpočtů. Důležitou vlastností je také podpora vzdáleného volání metod (RMI – Remote Method Invocation, [13]) a spolupráce s jinými programovacími jazyky (JNI – Java Native Interface).

MATLAB je programové prostředí a skriptovací programovací jazyk pro vědeckotechnické výpočty, simulace, zpracování signálů a další. Klíčovou datovou strukturou v tomto prostředí jsou matice (angl. Matrix), od toho také nese svůj název MATrix LABoratory.

Spojením těchto dvou elementů lze vytvořit dynamické aplikace nabízející na jedné straně výpočetní vlastnosti systému MATLAB a na straně druhé zpracování získaných výsledků jazykem Java do jakékoliv formy, například zprostředkování formou webového rozhraní, kterou se tato práce mimo jiné zabývá.

Vzájemná komunikace ve směru prostředí MATLAB a jazyk Java, tedy prostředí MATLAB volá programy jazyka Java je dostatečně zdokumentována už v nápovědě prostředí MATLAB, ovšem popis komunikace opačným směrem je již záležitostí téměř nezdokumentovanou a tato práce se tyto způsoby snaží přiblížit.

1.2 Volání prostředí MATLAB z jazyka Java

Úvod do této problematiky s možnostmi jeho řešení lze nalézt např. v [10]. Z této publikace vyplývá, že existují nejméně dva způsoby komunikace v tomto směru, kterými se budou zabývat podsekce 1.2.1 a 1.2.2. Ve skutečnosti existuje více způsobů, kupříkladu jsou to: nezdokumentované rozhraní JMI v části 1.2.3 a zakoupitelná součást MATLABu v části 1.2.4.

1.2.1 Využití třídy Java *Runtime*

S pomocí třídy Java *Runtime* lze spustit proces a odchytávat proudy bitů v obou jeho směrech. Konkrétně lze zachytit vstupní, výstupní a chybový proud. Následující kód (výpis 1.1) představuje spuštění procesu s MATLABem (*proces*) a uložení ukazatelů na jeho proudy (*in, out, err*). Proces se spustí příkazem (*prikaz*), jehož

Výpis 1.1: Využití třídy Java Runtime.

```
String prikaz = "matlab -nosplash -noFigureWindows -nodesktop -nojvm";
String prikaz2 = "matlab -automation";//alternativa
proces = Runtime.getRuntime().exec(prikaz);
out = new BufferedWriter(new OutputStreamWriter(proces.getOutputStream
    ()));
in = new BufferedReader(new InputStreamReader(proces.getInputStream()))
    ;
err = new BufferedReader(new InputStreamReader(proces.getErrorStream())
    );
```

argumenty jsou s největší pravděpodobností závislé na konkrétní verzi distribuce MATLABu[1]. Argumenty jsou jen volitelné, ale zrychlují spuštění i běh programu. Argument *-nosplash* zakazuje úvodní „splash screen“, *-noFigureWindows* zakazují vykreslování oken *figure* pro kreslení grafů a jiných grafických objektů a *-nodesktop* zakazuje otevření klasické plochy MATLABu, vyvolá se jen příkazové okno. Argument *-nojvm* zruší podporu Java programů volaných z MATLABu. Argument *-automation* by měl zajišťovat podobnou funkcionalitu. Pro výčet všech argumentů konkrétní distribuce je vhodné použít, mimo manuálu, příkazovou řádku s argumentem pro vypsání nápovědy *matlab -h*. Přehled vybraných argumentů je umístěn v tab. 1.1.

Pro komunikaci pak jen[2] stačí ovládat proudové čtečky a zapisovače *out, in*. Tento způsob komunikace je však pouze teoretický. Na OS Windows XP nelze provést takové odchytávání, protože MATLAB zde běží přes grafické rozhraní a nemá implementovány žádné výstupy nebo vstupy přes proudy bitů.

Jedním z myslitelným a zároveň multiplatformním řešení za pomocí této třídy je použití příkazové řádky s argumentem *-r*, což nám dá možnost spustit skript při náběhu programu. Dodáním potřebných vstupních a výstupních procedur tomuto skriptu, lze teoreticky uskutečnit veškerou funkcionalitu. Např. skript načte externí soubory s daty, transformuje tyto data na data MATLABu blízké. Provede výpočet a výsledky uloží do souborů, které může například dodatečně přenést pomocí FTP protokolu. Otázkou tohoto řešení je rychlost takové implementace, protože se operace pohybují na úrovni souborů[3].

[1] Uvedené platí pro verzi 2008a pro OS Windows XP, u ostatních verzí se mohou lišit, nebo nemusí být podporovány.

[2] Není to úplně pravda, viz dále.

[3] V současné době (2010) dosahuje teoretická rychlost čtení dat z paměti RAM typu DDR3 zhruba 12 GBps, teoretická rychlost rozhraní SATA revize 3 pevného disku pak kolem 600 MBps. Výsledek porovnání těchto teoretických hodnot udává asi 20 krát pomalejší přenos dat diskem, ovšem v praxi bude výsledek jistě horší.

Tab. 1.1: Argumenty pro spuštění MATLABu.

Argument	Význam
-nosplash	Nezobrazí „splash screen“ během startu.
-nodesktop	Nezobrazí standardní gui MATLABu, místo toho jen jednoduché příkazové okno. Spustí JVM.
-nojvm	Vypne podporu veškerých Java programu uvnitř MATLABu. Spustí se jednoduché příkazové okno („MATLAB Command Window“).
-r MATLABPrikaz	Spustí MATLAB a vykoná příkaz „MATLABPrikaz“. Použitý M-file musí být v pracovním adresáři.
-noFigureWindows	Zakáže vykreslování oken pro grafy (zakáže okna typu „figure“).
-automation	Spustí minimalizované příkazové okno MATLABu, bez „splash screen“.
-sd adresar	Nastaví pracovní adresář.

Výhody:

- platformní nezávislost,
- spolupráce s budoucími verzemi MATLABu s minimálním zásahem[4].

Nevýhody:

- přístup přes soubory (nebo proudy bitů v některých OS a distribucích MATLABu, ovšem v tomto případě to nebude s rychlostí také nijak závratné, protože se bude muset tento výstup ještě velice složitě upravovat na hodnotnější datové struktury.),
- většina logiky v M-souborech a z toho plynoucí složitější ladění aplikace,
- dodatečná úprava výsledků v jazyku Java jen přes soubory.

1.2.2 Využití JNI a MATLAB C-Engine

JNI (Java Native Interface) je rozhraní programovacího jazyka Java pro spolupráci s jinými programy a knihovnami napsanými v jiných programovacích jazycích (C, C++, ...). Lze tak dosáhnout obousměrného spojení mezi nativním kódem a kódem napsaným v Javě. MATLAB C-Engine je knihovna umožňující spouštět a využívat MATLAB v programovacích jazycích C, C++ a Fortran. Na daném PC, kde se bude takový program spouštět musí být přítomen MATLAB. Možné příkazy lze najít v tabulce 1.2. Tato knihovna je umístěna v hlavičkovém souboru *engine.h* u každé

[4]Úpravou argumentů příkazové řádky. Vhodným řešením by mohlo být např. uchovávání argumentů v konfiguračním souboru, a pro jiné distribuce tento soubor jen přenastavit.

Tab. 1.2: Příkazy knihovny MATLAB C-Engine.

Funkce	Význam
engOpen	Otevře příkazovou řádku MATLABu.
engClose	Ukončí příkazovou řádku MATLABu.
engGetVariable	Získá matici z MATLABu.
engPutVariable	Pošle matici do MATLABu.
engEvalString	Spustí příkaz definovaný řetězcem uvnitř MATLABu.
engOutputBuffer	Vytvoří paměť k uložení výstupního textu z prostředí MATLAB.
engOpenSingleUse	Otevře příkazovou řádku MATLABu pro jedno použití.
engGetVisible	Zjistí viditelnost příkazové řádky MATLABu.
engSetVisible	Schovat nebo zobrazit příkazovou řádku MATLABu.

distribuce MATLABu. Postup tvorby aplikace je poněkud těžkopádný. Nejdříve se vytvoří zdrojový soubor v jazyku Java. Zde se nadefinují nativní metody. Tento soubor se poté zkompiluje. Ze zkompilovaného souboru se pomocí překladače *javah* s argumenty *-jni balik.jmenoSouboru* vytvoří hlavičkový soubor pro část kódu v jazyku C. V novém zdrojovém souboru v jazyku C se tato hlavička naimportuje spolu s knihovnou *jni.h*, která je součástí každého JSDK (Java Software Development Kit – Balíček pro tvorbu Java programů) a naimportuje se i *engine.h*, který je zase součástí distribuce MATLABu. Využije se hlaviček z generovaného souboru k vytvoření metod, které budou obsahovat požadované volání a režii nad daty. Tento zdrojový kód se pak zkompiluje překladačem pro jazyk C a vytvoří se z něj *.dll* soubor. Tato *.dll* knihovna se pak umístí na vhodné místo[5] a v Java programu se načte pomocí metody *System.loadLibrary("nazevKnihovny")*. Podrobný popis lze nalézt v [10].

Tomuto přístupu se věnuje open source projekt JMatLink ([12]) od Steffana Müllera. V současné verzi (v1.3) je program navržen pro podporu obsluhy více vláken. Nástavbou nad toto rozhraní je rovněž open source projekt, MATLAB Server Pages (MSP, [9]), který spojuje JMatLink s webovým rozhraním a umožňuje jak vzdálené, tak lokální volání MATLABu. MSP je komplexní aplikace zahrnující i vlastní JSP značky a ošetřování výjimek a navíc je snadně použitelná ve své základní verzi. Nevýhoda tohoto projektu tkví v implementaci staré verze JMatLinku (v1.0), která nepodporovala umísťování do Java balíčků (package), což může způsobovat problémy s kompilací a údržbou.

[5]Ve Win XP například do kořenového adresáře Windows. Možno umístit i jinam, ale aby cesta byla uvedena v proměnné PATH.

Výhody:

- teoreticky rychlý přístup k datům přes ukazatele,
- jednoduché přetypování mezi datovými typy MATLABu a Javy.

Nevýhody:

- platformně závislé jádro, nutnost kompilace „.c“ souboru na „.dll“ pro danou platformu,
- těžké ladění, obsahuje kódy z více programovacích jazyků,
- nutné ošetřit uvolňování prostředků v nativní části.

1.2.3 Využití JMI

JMI (Java-to-Matlab Interface) je Javovské rozhraní pro komunikaci s MATLABem přímo z dílny Mathworks. Toto rozhraní je dodáváno v každé distribuci MATLABu. Na první pohled se může zdát, že je to nejideálnější a nejschůdnější cesta k implementaci komunikace Java-MATLAB, ovšem je tu jedno úskalí. Největším nedostatkem tohoto rozhraní je nulová dokumentace a ze strany Mathworks rovněž nulová podpora. Největším zdrojem informací mohou sloužit pouze neoficiální materiály. Částečně zdokumentované rozhraní a mimo jiné i ostatní nedostatky MATLABu jsou uvedeny na blogu [1].

Metody obsažené uvnitř tohoto rozhraní lze vypsat pomocí příkazu *methodsview('com.mathworks.jmi.Matlab')* uvnitř MATLABu. Výsledkem na takový dotaz může být obr. 1.1. Jak už bylo zmíněno, neexistuje žádná oficiální dokumentace a z [1] vyplývá, že některé metody vykazují asynchronní chování. To může mít za následek nevyzpytatelné stavy, jestliže se metody spustí v krátkém sledu po sobě a jedna metoda bude závislá na výsledku druhé, musí se tato skutečnost ošetřit. Na

Methods for class com.mathworks.jmi.Matlab

Qualifiers	Return Type	Name	Arguments
static	void	LMKeepAlive	( )
		Matlab	( )
static	void	addListener	(com.mathworks.jmi.MatlabListener)
	boolean	equals	(java.lang.Object)
	void	eval	(java.lang.String,com.mathworks.jmi.CompletionObserver)
	void	eval	(char)
	void	eval	(java.lang.String,com.mathworks.jmi.MatlabListener,int)
	void	eval	(java.lang.String,com.mathworks.jmi.MatlabListener)
	void	eval	(java.lang.String,com.mathworks.jmi.CompletionObserver,int)
	java.lang.String	eval	(java.lang.String)
	void	evalConsoleOutput	(java.lang.String,com.mathworks.jmi.CompletionObserver,int)
	void	evalConsoleOutput	(java.lang.String,com.mathworks.jmi.CompletionObserver)
	void	evalConsoleOutput	(java.lang.String,com.mathworks.jmi.MatlabListener)
	void	evalConsoleOutput	(java.lang.String)

Obr. 1.1: Část výpisu metod rozhraní Java-to-Matlab Interface.

důkaz funkčnosti tohoto rozhraní s ošetřením asynchronnosti existuje více projektů. Z těch nejznámějších open source projektů je to JAMAL – JAva MAtlab Linking ([8]) a MatlabControl ([7]). U obou projektů je implementována podpora vzdáleného volání MATLABu a tedy správná půda pro realizaci distribuovanosti. JAMAL je ovšem poněkud chudší bratr MatlabControlu a realizuje jen jednu metodu z JMI. Ovšem i s ní lze uskutečnit veškerou komunikaci, i když někdy za cenu těžkopádnosti a menší efektivity.
Výhody:

- platformní nezávislost,
- převod mezi datovými typy Javy a MATLABu.

Nevýhody:

- oficiálně nezdokumentováno,
- diskutabilní kooperace s novými verzemi MATLABu.

1.2.4 MATLAB Builder JA

MATLAB Builder JA je zakoupitelná součást MATLABu. Tento produkt umožňuje převádět „.m“ soubory na Java třídy, se kterými lze libovolně manipulovat a uzpůsobovat je konkrétním aplikacím (web i desktop). Tyto třídy lze volně spouštět na jakémkoliv PC a platformě podporující MATLAB. Ke spuštění je nutné mít nainstalovaný MCR (MATLAB Compiler Runtime), jedná se o prostředí pro spouštění zkompilovaných souborů z MATLABu, které je volně dostupné.
Výhody:

- možnost tvorby interaktivních grafů (AJAX a Servlety),
- rozhraní pro automatický převod mezi datovými typy Javy a MATLABu,
- podpora.

Nevýhody:

- další placená součást,
- pracná a časově nepříznivá kompilace a vytváření aplikace.

2 SROVNÁNÍ METOD

Na základě předchozího výčtu různých metod se práce zaměří na jejich testování. Konkrétně na testování JMatLinku (kap. 1.2.2) a projektů JAMAL a MatlabControl (kap. 1.2.3). Přístup z kapitoly 1.2.4 byl zavržen z důvodu složité implementace nových výpočetních schopností navrhovaného systému. Přístup z kapitoly 1.2.1 byl zavržen pro práci na úrovni souborů a pro nutnost úpravy výpočetních skriptů, čímž by se narušil návrh na snadnou rozšiřitelnost o nové moduly.

Pro testování tří jmenovaných projektů bylo navrženo rozhraní *IMatlabConnection* (výpis 2.1). Nutno podotknout, že toto rozhraní není v konečné formě a bude dále vyvíjeno a upravováno. Takto uvedené je prozatím v dostačující formě pro testování. Bude doplněno o případnou implementaci distribuovaných výpočtů, ovšem ne všechny projekty toto umožňují v základní formě[1]. Realizaci tohoto řešení ve jmenovaných projektech lze vidět ve výpisech 2.2, C.1 a C.2.

Veškeré testování proběhlo na PC s konfigurací: procesor AMD DualCore 4800+ (2x2,6GHz), 4 GB RAM 667MHz, 7200 rpm HDD, operační systém Windows XP SP3, MATLAB R2008a.

2.1 Rozhraní IMatlabConnection

Z názvů metod by mělo vyplynout, co jaká metoda představuje. Pro kompletnost však bude uveden jejich výčet. Koncovka MC představující zkratku MatlabConnection, byla dodána pro ujištění jedinečnosti názvu metody. Myslitelnou datovou strukturou pro získávání a posílání dat jsou prozatím matice reálných čísel a jejich podmnožiny (vektory, skaláry).
Metody rozhraní:

- *sendCmdMC(String cmd)* – pošle a spustí řetězec s příkazem specifikovaným v proměnné *cmd* v příkazovém okně MATLABu,
- *sendVariableMC(String varName, double[][] data)* – pošle a uloží matici dat (*data*) pod názvem *varName* v MATLABu,
- *double[][] receiveVariableMC(String varName)* – přijme matici dat z MATLABu specifikovanou názvem proměnné *varName*,
- *long createConnectionMC()* – vytvoří nové spojení s MATLABem a vrátí identifikátor tohoto spojení,
- *destroyConnectionMC(long identifier)* – zruší spojení specifikované předešle získaným identifikátorem *identifier*.

[1] JMatLink ve své současné verzi distribuci výpočtů nepodporuje. Je nutné vytvořit nad ním nástavbu.

Výpis 2.1: Rozhraní pro testování různých druhů přístupů.

```
package cz.vutbr.fekt.utko.xstefe05.matlabconnector;
public interface IMatlabConnection {
    public void sendCmdMC(String cmd);
    public void sendVariableMC(String varName, double[][] data);
    public double[][] receiveVariableMC(String varName);
    public long createConnectionMC();
    public void destroyConnectionMC(long identifier);
}
```

2.2 Projekt JMatLink

Jedná se o OSS (Open Source Software – Software s otevřeným zdrojovým kódem). V současnosti je ke stažení ve verzi 1.3 (z roku 2005) z [12]. Obsahuje již zkompilovanou knihovnu *dll* pro operační systémy Windows a makefile skript pro kompilaci pod operačním systémem Linux. *Dll* se podle návodu umístí do kořenového adresáře Windows. Dále je potřeba přidat do systémové proměnné *Path* absolutní cestu k adresářům MATLABu: cestu k adresáři *bin* a k adresáři *bin/win32*. A nakonec systém restartovat. Od předchozí verze je Javovská část programu umístěna v balíčku a lze jí zkompilovat i novější verzí kompilátoru (předtím jen Java v1.3).

Aplikace je navržena pro mnohonásobnou komunikaci s MATLABem, ovšem je ošetřena díky omezením Windows XP a MATLABu docela neefektivním způsobem. Příkazové řádky MATLABu jsou volány jen z jednoho vlákna a každá vyvolaná metoda čeká na zpracování a brzdí tím vlákna ostatní, tím se výhoda vícevláknovosti příliš neprojeví. Realizaci implementace navrženého rozhraní pro testování lze nalézt ve výpisu 2.2. Negativum tohoto projektu je dlouhá doba bez žádné aktualizace[2] a lze se tedy domnívat, že projekt je nyní v mrtvém stavu.

Výpis 2.2: Implementace v projektu JMatLink.

```
public class JMatLink implements cz.vutbr.fekt.utko.xstefe05.
    matlabconnector.IMatlabConnection {
...
    public void sendCmdMC(String cmd) {
        engEvalString(cmd);
    }
    public long createConnection() {
        long l = engOpenSingleUse();
    }
    public void destroyConnection(long identifier) {
        engClose(identifier);
```

[2]Naposledy 28-12-2005.

```
    }
    public double[][] receiveVariableMC(String varName) {
        double[][] arr = engGetArray(varName);
        return arr;
    }
    public void sendVariableMC(String varName, double[][] data) {
        engPutArray(varName, data);
    }
}
```

2.3 Projekt MatlabControl

Tento OSS, momentálně (05-2011) ve verzi 3.1.0, nabízí Javovské API (Application Programming Interface – Rozhraní pro programování aplikací) pro ovládání a interakci lokální nebo vzdálené komunikace s MATLABem. Lokální komunikace je taková, kde je Java program spuštěn přímo z MATLABu, vzdálená komunikace znamená nezávislé spuštění Java programu a MATLABu a jejich kooperaci pomocí RMI (Remote Method Invocation – Vzdálené volání metod). Toto API spoléhá na JMI (Java-to-MATLAB Interface – Java rozhraní pro přístup k programu MATLAB), které není oficiálně zdokumentováno a ze strany Mathworks není podporováno. Pro testování této aplikace byla navržena nová třída, lze nalézt ve výpisu C.2. Během testování tato aplikace vykazovala asynchronní chování. Pro nápravu tohoto chování byla třída doplněna ne zcela dostačujícím způsobem o krátké časové úseky pro uspání vlákna. Před každou metodou s komunikací s MATLABem je tedy doplněna krátká časová prodleva[3] definovaná konstantou *SLEEPTIME*. Pro funkčnost tohoto API bylo taktéž nutné nastavit cestu k souboru *matlab.exe* v metodě *getMatlabLocation()* třídy *Configuration*.

Pro navržené rozhraní je nutno výstup z MATLABu přetypovat, protože výsledky z původní metody jsou uloženy v Java objektu. Dalším problémem[4] je rozdílná reprezentace dvourozměrných polí v MATLABu a v Javě a výsledkem na dotaz o matici z MATLABu bude jen jeden vektor uložený v Java objektu. Pro účely této práce a s návrhem na řešení pouze v oboru matic reálných čísel byla realizována jednoduchá metoda (výpis 2.3), která nejdříve získá rozměry požadované matice a poté rozloží získaný vektor na dvourozměrné pole pomocí metody *objectToDouble*.

[3]Tuto prodlevu by bylo vhodné umístit do konfiguračního souboru pro její jednodušší změnu. Na jiném PC nebo pro složitější analýzy by nemusela dostačovat.

[4]Tento problém se vyskytuje i u projektu JAMAL a byl vyřešen stejným způsobem.

Výpis 2.3: Metoda pro získání matice z programu MATLAB, projekt MatlabControl.

```
private double[][] receiveVariable(String varName) throws
    MatlabInvocationException {
        double[][] result = null;
        Object variable = null, dimension = null;
        variable = this.proxy.returningEval(varName, 1);
        dimension = this.proxy.returningEval("size(" + varName + ")",
            1);
        result = objectToDouble(variable, dimension);
        return result;
}
```

2.4 Projekt JAMAL

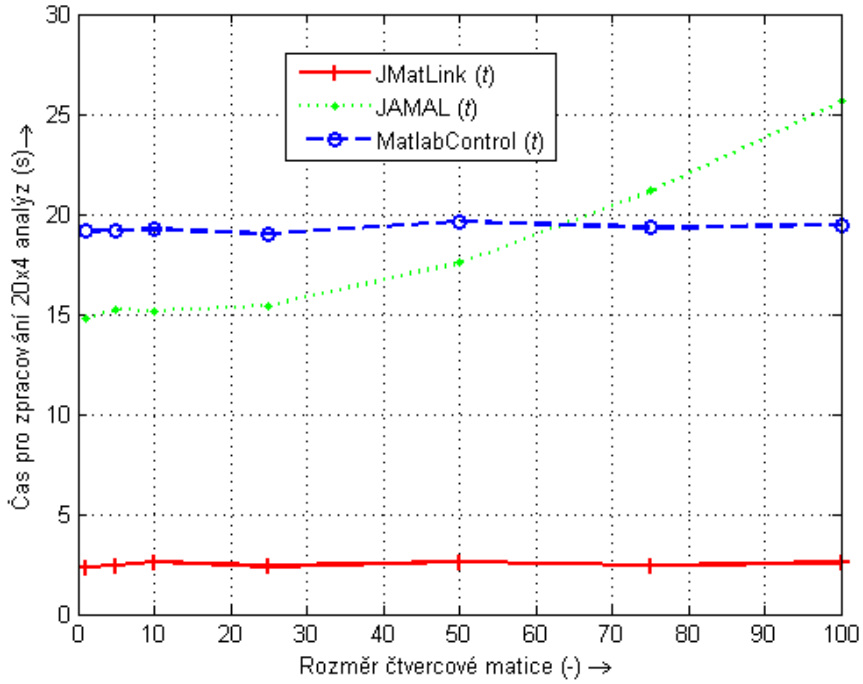
JAMAL je rovněž OSS. Projekt je velice podobný projektu MatlabControl. Ke komunikace Java-MATLAB také využívá RMI. V současné době (05-2011) funguje ve verzi 2.1. Skládá se rovněž ze dvou částí (server a klient). Nabízí však pouze jednu metodu přístupu k MATLABu, ovšem i s ní lze vytvořit veškerá logika. Na rozdíl od projektu MatlabControl zde dochází k problému uváznutí příkazového okna MATLABu a nelze se do něj přepnout. Tento fakt může způsobit velkou obtíž při ladění aplikace. Co se instalace týče, je potřeba nastavit cestu k MATLABu v rozhraní *MatlabCaller*. Dále je nutné přidat řádek s úplnou cestou k jeho JAR (Java ARchive – Souborový archív pro programovací jazyk Java) souboru do souboru *classpath.txt*, ze kterého čte program MATLAB cesty. Uvnitř MATLABu lze tento konfigurační soubor editovat jednoduchým příkazem „`edit classpath.txt`“.

Realizaci implementace navrženého rozhraní pro testování tohoto projektu lze nalézt ve výpisu C.1. Byla vytvořena nová třída *MatlabClientImpl*, odvozená od originálu *MatlabClient*, protože bylo nutné upravit konstruktor, aby v něm implicitně nedocházelo k navázání komunikace. Z důvodu jediné metody pro přístup k MATLABu, bylo nutné změnit metodu pro odesílání proměnných (*sendVariableMC*). Vstupní pole se transformuje do textového řetězce a poté se jí pomocí funkce *eval* uvnitř programu MATLAB přiřadí jméno. Tato transformace má za následek výrazné zpomalení u matic velkých rozměrů. V porovnání s projektem MatlabControl však při testování nebyl objeven problém s asynchronností metod.

2.5 Testování metod

Pro účely testování a porovnání předešlých metod přístupu bylo vytvořeno několik tříd. Konkrétně to jsou třída pro navázání spojení, předávání příkazů a získávání

výsledků z jednotlivých typů spojení *MatlabConnector* (výpis C.8), třída pro měření vykonávaných úkonů *ConnectionTester* (výpis C.9), třída s jednoduchými analýzami *TestAnalyses* (výpis C.10) a třída *Test* (výpis C.11), která spojuje všechny tyto programy do jedné aplikace. Měření časových úseků vykonává třída *Stopky* (výpis C.3). Rozhraní pro analýzy *IAnalysis* (výpis C.4) a třída pro ukládání jednotlivých analýz *Analysis* (výpis C.5). Každá analýza je rozčleněna na vstupní a výstupní elementy (výpis C.6 a výpis C.7).



Obr. 2.1: Srovnání metod.

Celý test byl vykonán tak, jak je doložen ve výpisu C.11. Každá z analýz (výpis C.10) byla provedena dvacetkrát pro každé vytvořené spojení, tedy 80 analýz pro každé ze tří druhů testovaných spojení, navíc byl každý takový test prováděn s různými rozměry vstupních matic. Výsledkem měření je graf na obr. 2.1, který vyjadřuje časovou náročnost výpočtu všech 80 analýz pro dané spojení s daným rozměrem matic. Do uvedených časových hodnot není započten čas pro vytvoření spojení, protože test byl především zaměřen na měření rychlosti zpracování jednotlivých metod.

Z výsledku vyplývá, že metoda JMatLink dosahuje nejlepšího výkonu. V celém průběhu, tedy pro jakékoliv rozměry matice, má konstantní zpoždění a navíc je nejrychlejší z implementovaných. MatlabControl vykazuje rovněž konstantní časové nároky, ovšem je asi sedmkrát pomalejší než JMatLink. Pomalost této metody je způsobena přidanými časovými prodlevami částečně negujícími asynchronnost metod, viz kapitola 2.3. JAMAL je sice o něco rychlejší než projekt MatlabControl při

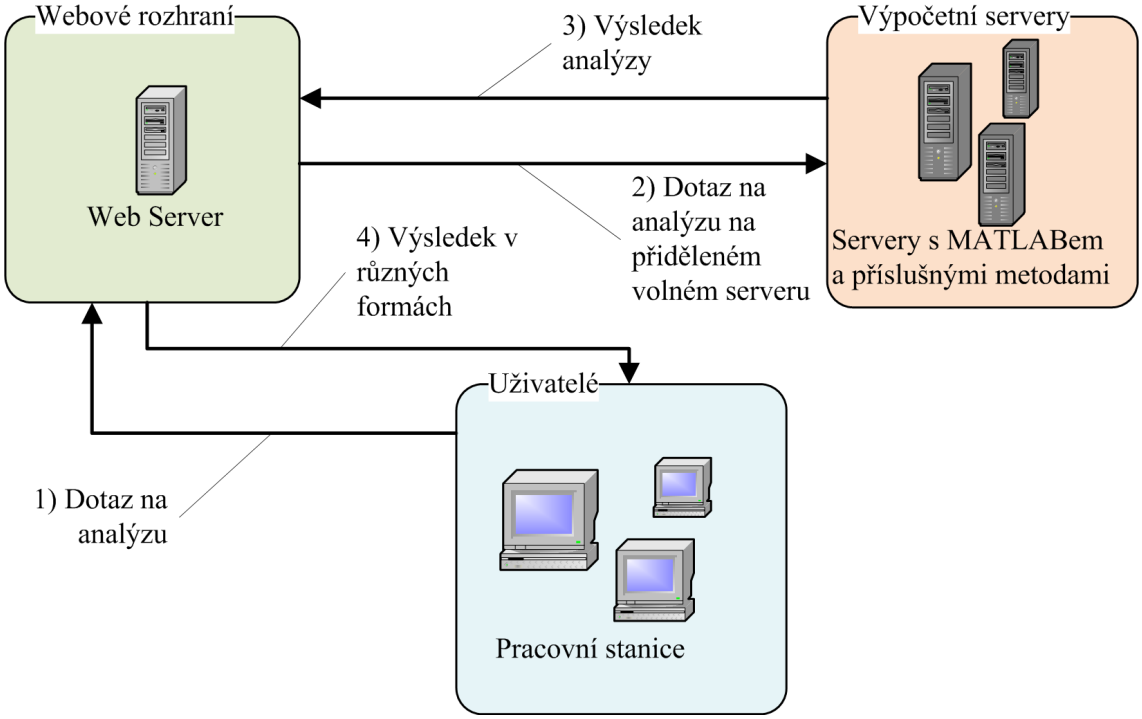
malých rozměrech matic, ale na druhou stranu jde vidět, že navržená implementace pro vkládání proměnných do MATLABu není ideální a vykazuje značný nárůst výpočetní kapacity při rostoucích rozměrech vkládaných matic.

3 NÁVRH SYSTÉMU

Ze srovnání metod vyplývá, že nejlépe v ohledu na rychlost je na tom metoda JMatLink, ovšem co se schopností týče, mírně zaostává za ostatními. Tato metoda nepodporuje RMI, a proto je možné ji nasadit bez dodatečných úprav pouze na stejné PC jako web server. Ostatní metody (JAMAL a MatlabControl) podporují RMI a mohou být nasazeny na vzdálené servery bez dodatečného zásahu do jejich kódu.

Díky vlastnostem objektově orientovaného jazyka lze realizovat aplikaci využívající všechny uvedené metody. Důvod k tomuto kroku může být opodstatněný tím, že projekty JAMAL a MatlabControl jsou poměrně nové a stále vyvíjené, a je vysoce pravděpodobné, že se dočkají nových aktualizací, což může vést ke zvýšení efektivity jejich práce a na druhé straně projekt JMatLink, který má sice omezené možnosti, ovšem je momentálně mnohonásobně rychlejší.

Strukturu aplikace lze vysledovat z obrázku 3.1. Systém je rozdělen do tří bloků. Definice každého z nich je uvedena v příslušných sekcích.



Obr. 3.1: Schéma aplikace.

Tab. 3.1: Přehled rolí a jejich pravomocí.

Role	Pravomoce
ROLE_ADMIN	Inicializovat spojení s Matlabem. Přidávat a editovat analýzy. Prohlížet logovací soubor. Mazat soubory ostatních uživatelů. A všechna práva běžného uživatele (ROLE_USER).
ROLE_USER	Spustit analýzu. Prohlédnout si výsledky analýzy. Mazat své soubory a nahrávat soubory na server.

3.1 Uživatelé

S ohledem na co nejmenší požadavky na stranu klientů systému bylo vytvořeno webové rozhraní. Pro přístup k tomuto rozhraní tedy stačí uživatelům webový prohlížeč. Pro využití appletu k nahrávání souborů na server je zapotřebí mít nainstalované JRE (Java Runtime Enviroment – Prostředí pro spuštění programů napsaných v jazyku Java). Žádné jiné zvláštní programové vybavení není zapotřebí.

Uživatel přistupuje do systému pod účtem s heslem a bude mít s účtem svázána určitá privilegia. Tyto privilegia umožňují uživatelům ovlivňovat prostředky systému a zároveň jim omezovat přístup k některým funkcím. Přehled rolí a práv je umístěn v tabulce 3.1.

3.2 Webové rozhraní

Tato část zastupuje jedinou možnou komunikaci výpočetního jádra systému s uživateli. Dále tato vrstva obsahuje FTP klienta pro distribuci a výměnu souborů s výpočetními servery. Konkrétní úkoly a možnosti této vrstvy jsou uvedeny v následujícím výčtu.
Úkoly:

- přístup k systému pomocí uživatelských účtů a hesel,
- spuštění analýz,
- prezentace výsledků,
- uchovávání přehledu o výpočetních serverech a podle těchto statistik vybírat nejvhodnější server pro uživatelské požadavky,
- správa souborů a modulů a jejich distribuce ke konkrétnímu výpočetnímu serveru.

Rozličností jednotlivých metod pro přístup a pro snadnější zpracování programem je vhodné opatřit každé spojení i analýzy svým popiskem. Byl vybrán popis pomocí XML (Extensible Markup Language – Rozšiřitelný značkovací jazyk, [19])

dokumentu, protože forma takového souboru má velkou vypovídající hodnotu a je plně přenositelná. Takto vytvořené popisky lze navíc validovat proti DTD (Document Type Definition – Definice typu dokumentu) nebo XSD (XML Schema Definition – Schéma popisující strukturu XML souboru) šabloně[1] ([5]) a zajistit tak jejich správnost[2]. Každé spojení i analýza bude tímto dokumentem pevně definována.

Provoz webové aplikace bude zajišťovat webový server OSS Servlet a JSP (Java Server Pages – Webové stránky na bázi Javy) kontejnerem Apache Tomcat ([15]), který si získal nemalou oblibu pro svou jednoduchou instalaci a konfiguraci. Formát běhu bude řízen specifikami aplikačního rámce Spring, který je zběžně vysvětlen v kapitole 4. Komplexní zabezpečení systému je realizováno systémem Acegi Security System for Spring. Šifrovaný přenos je zaručen přes protokol HTTPS (Hypertext Transfer Protocol Secure – Zabezpečený protokol HTTP).

Samotné webové stránky budou ve formě *jsp* ([2]) souborů a kódovány ve striktním *xhtml* formátu využívající kaskádových stylů ([3]).

3.3 Výpočetní servery

Tyto servery jsou výpočetním jádrem celého systému. Každý výpočetní server bude vybaven: metodou pro přístup k MATLABu (viz metody v kap. 2), samotným výpočetním prostředím MATLAB, Java programem pro RMI komunikaci s webovým serverem a FTP Serverem pro obousměrnou datovou distribuci.

[1] Samozřejmě ještě existují jiné druhy šablon, ale uvedené patří k nejpoužívanějším.

[2] Pouze částečně, lze totiž validovat pouze zápis, ovšem jeho sémantická správnost je věcí druhou.

4 APLIKAČNÍ RÁMEC SPRING

4.1 Úvod

Rámec Spring je jedním z mnoha aplikačních rámců pro programovací jazyk Java s otevřeným zdrojovým kódem používaných mimo jiné i ve webových aplikacích (Spring MVC – Model-View-Controller). Aplikační rámec, někdy též označovaný kontejner, se snaží běžně se opakující funkce aplikací implementovat ve svých vlastních třídách a poskytuje tím zjednodušení vývoje aplikací. Programátorovi tím odpadne větší množství práce díky takovému kontejneru. Ulehčení práce není jediný cíl, rámec učiní práci lehčeji udržovatelnou. Navíc se kontejner stará o životní cyklus aplikace a přesouvá se tím kontrola běhu aplikace více do rámce. Mezi jeho vlastnosti patří:

- intuitivní konfigurace přes XML dokumenty,
- snadná spolupráce a kompozice s jinými aplikačními rámci,
- podpora AOP (Aspect Oriented Programming – Aspektově orientované programování),
- knihovna značek pro JSP,
- „vnucení závislostí“ (dependency injection, inversion of control),
- open source.

Vnucení závislosti je jedna z hlavních náplní rámce Spring. Znamená to, že lze přesunout odpovědnost za vytváření a provázání objektů z aplikace do aplikačního rámce. Objekty jsou tedy vytvářeny a provázány v konfiguračních souborech. Tím se dá snadno zaměňovat referencí a snadno tím měnit např. starou implementaci nějaké logiky za novou nebo jinou. Pro ukázku je uveden krátký příklad ve výpisu 4.1. Kód je součástí konfiguračního souboru výsledného programu z kapitoly 5. Příklad uvádí tvorbu jednoho objektu s názvem *distributionService*, který je instancí třídy *DistributionServiceImplFTP* a je mu „vnuceno“ nebo také injektováno do referenční proměnné na rozhraní pro distributora souborů (*FTPFileProvider*) konkrétní instance třídy *FTPFileProviderImpl* (tato třída implementuje potřebné rozhraní *FTPFileProvider*) a dále je mu injektována instance *directoryService*, která byla vytvořena dříve a je sdílená mezi více objekty. Pro kompletnost je zde uveden i výsek zmiňované třídy ve výpisu 4.2. Tento výsek je příkladem tzv. „setterové injekce“[1]. Více informací o rámci Spring lze nalézt například v [14] a [17].

[1] Pro správné fungování je potřeba vybavit třídy metodami *set*

Výpis 4.1: Ukázka vnucování závislostí – konfigurační soubor.

```
...
<bean id="directoryService" class="cz.vutbr.fekt.utko.xstefe05.spring.
    service.DirectoryServiceImpl">
    <!-- vytvoření instance třídy DirectoryServiceImpl pod názvem
        directoryService -->
    ...
</bean>
...
<bean id="distributionService" class="cz.vutbr.fekt.utko.xstefe05.
    matlabconnector.DistributionServiceImplFTP">
    <property name="ftpFileProvider" > <!-- do referenční proměnné s
        názvem ftpFileProvider umístit referenci na instanci třídy ...
        -->
        <bean class="cz.vutbr.fekt.utko.xstefe05.matlabconnector.
            ftpprovider.FTPFileProviderImpl"/>
    </property>
    <!-- do referenční proměnné s názvem directoryService umístit
        referenci na instanci třídy directoryService vytvořenou dříve
        -->
    <property name="directoryService" ref="directoryService"/>
</bean>
...
```

Výpis 4.2: Ukázka vnucování závislostí – třída.

```
public class DistributionServiceImplFTP implements IDistributionService
    {
    private FTPFileProvider ftpFileProvider;
    private DirectoryService directoryService;
                ...
    public void setDirectoryService(DirectoryService directoryService)
        {
        this.directoryService = directoryService;
    }
    public void setFtpFileProvider(FTPFileProvider ftpFileProvider) {
        this.ftpFileProvider = ftpFileProvider;
    }
    ...
```

Tab. 4.1: Přehled stavů ve Webflow.

Stav	Element v XML souboru	Význam
Action	action-state	Stav pro využití ke zpracování požadavků, ukládání nebo získání dat z databáze.
Decision	decision-state	Stav, kdy je zapotřebí implementovat nějaké rozhodování.
End	end-state	Po dosažení konečného stavu je konverzace ukončena.
Start	start-state	Počáteční stav neobsahuje žádnou logiku, ale slouží jako odkazové místo na první stav v konverzaci.
Subflow	subflow-state	Tento stav umožňuje zahrnout do konverzace ještě další konverzaci. Využívá se hlavně pro rozkládání jedné složité konverzace na jednotlivé elementární.
View	view-state	Prohlížecí stav je většinou reprezentován formulářem a umožňuje uživateli se nějakým způsobem zapojit do konverzace.

4.2 Spring Webflow

Velice výkonným nástrojem rámce Spring je pokročilá tvorba konverzací mezi uživatelem a webovým rozhraním. Ukázka části konfiguračního souboru pro konverzaci u tvorby nové analýzy je ve výpisu 4.3. Každá konverzace musí mít jediný počáteční stav, ale může mít několik konečných stavů. Mezi počátkem a koncem konverzace většinou proběhne několik dalších stavů jejichž výpis a význam je umístěn v tabulce 4.1. V uvedeném případě začne konverzace ukázkou formuláře (*analysisCreationForm*[2]), po jeho vyplnění čeká kontejner na signál *submit*, který je spuštěn odesláním formuláře od uživatele. Tento čin posune konverzaci dále do akčního stavu *chechXMLActionState*. Jeho náplní je kontrola vstupních dat, tedy v prvé řadě validace XML souboru s novou analýzou. Pokud je vše v pořádku, přejde konverzace do konečného stavu a zobrazí uživateli stránku s názvem *result*[3]. Pokud při validaci nebo jiných ověřovacích mechanismech došlo k chybě, zobrazí se uživateli znovu první formulář, teď už s výpisem těchto chyb.

[2]Vyhledá se soubor *analysisCreationForm.jsp*.

[3]Opět jde o soubor *.jsp*. O správnou adresaci a překlad adres se stará tzv. Dispatcher Servlet.

Výpis 4.3: Webflow – konfigurační soubor *createAnalysis-flow.xml*.

```
...
<start-state idref="creationFormState" />
<view-state id="creationFormState" view="analysisCreationForm">
    <transition on="submit" to="checkXMLActionState" />
</view-state>
<action-state id="checkXMLActionState">
    <action bean="checkXMLAction" />
    <transition on="success" to="finish" />
    <transition on="fail" to="creationFormState" />
</action-state>
<end-state id="finish" view="result" />
...
```

4.3 Acegi Security

Acegi Security je aplikační rámec pro zabezpečení aplikací na bázi rámce Spring. Tento rámec nabízí komplexní zabezpečení v mnoha vrstvách: od autentizace a autorizace přes zajištění zabezpečeného kanálu přes HTTPS až po úroveň volání metod.

Jedna z mnoha implementovaných vrstev je ve výpisu 4.4. Jedná se jen o jednoduchý druh řízení přístupu[4] k odkazům na prostředky serveru (spuštění analýzy, mazání souborů...). Středem zájmu jsou značky *<authz:authorize>*. Z výpisu je zřejmé, že pokud má uživatel přidělena práva *ROLE_ADMIN* nebo *ROLE_USER* zpřístupní se mu odkazy na části systému. Pokud nemá ani jedno z práv zobrazí se mu jen odkaz na přihlášení.

Výpis 4.4: Zabezpečení pomocí ACEGI Security, soubor *navigator.jsp*.

```
...
<authz:authorize ifNotGranted="ROLE_USER, ROLE_ADMIN">
    <li><!-- Sekce pro uživatele bez přidělené role -->
        <a href="login.html"><spring:message code="navigator.login"/>
        </a>
    </li>
</authz:authorize>
<authz:authorize ifAnyGranted="ROLE_USER, ROLE_ADMIN">
<!-- Pouze pro přihlášené uživatele -->
...
</authz:authorize>
...
```

[4]Jen část metody řízení přístupu. Zbylé části jsou v konfiguračním souboru *WEB-INF/jmc-security.xml*.

5 IMPLEMENTACE SYSTÉMU

Projekt nese pracovní název **jmc**. Pod touto zkratkou si lze představit Java-Matlab Connecting. Veškeré zdrojové kódy projektu jsou uloženy ve stejnojmenném webovém archívu *jmc.war* (obsah archívu je umístěn v příloze B) a v Java archívu *jmc-ServerGui.jar*, kde je jednoduché grafické rozhraní pro správu (vytvoření a zničení) konektorů na straně výpočetních serverů. Práce čítá přes 100 zdrojových a konfiguračních souborů, proto není možné vysvětlovat význam všech. Souhrn vlastností systému lze nalézt v následujícím výčtu:

- Spojení pomocí metody JMatLink, která vykazovala nejlepší vlastnosti (kapitola 2).
- Webové rozhraní pro snadný přístup uživatelů bez nutnosti dodatečných instalací.
- Dynamická tvorba a úprava analýz.
- Zabezpečený přenos pomocí HTTPS, řízení přístupu.
- Dva testovací uživatelé, administrátor – účet: admin a heslo: admin a uživatel – účet: user, heslo: user. Nastaveni v souboru *jmc-security.xml*.
- Správa souborů: nahrávání na server pomocí appletu a mazání souborů pomocí webového rozhraní.
- Uchovávání výsledků spuštěných analýz pro každého uživatele.
- Distribuce výpočtu a souborů k volnému serveru.
- Podpora práce s komplexními čísly a maticemi a ucelený formát jejich zápisu.
- Veškeré konfigurace v XML souborech.
- Lokalizace webových stránek do 2 jazyků (čeština a angličtina).
- Několik ukázkových analýz. Např.: výpočet determinantů, ekvalizace histogramu, výpočet melovských kepstrálních koeficientů.

V následujících sekcích bude popsána instalace a konfigurace systém. Následně bude uveden popis vybraných funkcionalit systému.

5.1 Instalace a konfigurace

Instalace se skládá ze dvou částí. U klientů není potřeba žádný speciální software, stačí jen webový prohlížeč a JRE[1]. Konfiguracím a instalacím částí pro webový server a pro výpočetní servery se věnují příslušné kapitoly.

[1]Pro spuštění appletu pro nahrávání souborů na server.

Výpis 5.1: Zrušení nutnosti HTTPS – konfigurační soubor *jmc-security.xml*.

```
...
<bean id="channelProcessingFilter" class="org.acegisecurity.
    securechannel.ChannelProcessingFilter">
    <property name="filterInvocationDefinitionSource">
        <value>
            CONVERT_URL_TO_LOWERCASE_BEFORE_COMPARISON
            PATTERN_TYPE_APACHE_ANT
            /**=REQUIRES_INSECURE_CHANNEL <!-- pridat tento radek -->
            <!-- a tuto cast zakomentovat
            /login.html=REQUIRES_SECURE_CHANNEL
            /ccs.html=REQUIRES_SECURE_CHANNEL
            /upload.html=REQUIRES_SECURE_CHANNEL
            /flow.html=REQUIRES_SECURE_CHANNEL
            /j_acegi_security_check*=REQUIRES_SECURE_CHANNEL
            -->
        </value>
    </property>
    <property name="channelDecisionManager" ref="channelDecisionManager
        " />
</bean>
...
```

5.1.1 Webový server

Instalace a konfigurace Apache Tomcat

První úkol je instalace webového serveru Apache Tomcat. Z domovského webu tohoto software lze stáhnout distribuci programu buď ve formě instalačního souboru nebo ve formě komprimovaného souboru formátu zip. Dále je nutné pro správné fungování projektu vypořádat se s nastavením protokolu HTTPS. Buď lze toto zabezpečení vypnout nebo ho nastavit.

K vypnutí postačí zakomentovat několik řádků v konfiguračním souboru uvnitř projektu: *WEB-INF/jmc-security.xml* tak jak je ukázáno ve výpisu 5.1. Soubor lze nalézt po dekompresi ve webovém archívu *jmc.war* (viz bod 5.1.1).

V opačném případě je nejprve nutné získat certifikát. Pro testovací účely stačí self-signed certifikát, který se vytvoří příkazem: *%JAVA_HOME%/bin/keytool -genkey -alias tomcat -keyalg RSA* (viz [18]). Během generování klíče budou vyžádány uživatelské informace a heslo (heslo pro tomcat se použije v konfiguraci později). Certifikát se automaticky[2] uloží do domovského adresáře pod názvem

[2]Není potřeba ho nikam přemisťovat. Namapuje se v konfiguračním souboru.

Výpis 5.2: Podpora HTTPS – konfigurační soubor programu Tomcat *server.xml*.

```
...
<!-- Prenastaveni na port 80-->
<Connector port="80" protocol="HTTP/1.1"
        connectionTimeout="20000"
        redirectPort="443" />\
...
<!-- Odkomentovani a doplneni podpory HTTPS -->
<Connector SSLEnabled="true" clientAuth="false" maxThreads="150" port="
    443" protocol="HTTP/1.1" scheme="https"
     keystoreFile="${user.home}/.keystore" keystorePass="#heslo zadavane
         u tvorby certifikatu#"  secure="true" sslProtocol="TLS" />
...
```

.keystore. Tento certifikát je pak zapotřebí spojit se samotným protokolem v konfiguračním souboru Apache Tomcat. V konfiguračním souboru tohoto programu je nutné odkomentovat a přenastavit porty pro protokoly HTTP a HTTPS pro správné fungování systému, tak jako ve výpisu 5.2. Cesta k souboru: *%tomcat%/conf/server.xml*[3].

Instalace a konfigurace aplikace pro spojení s prostředím MATLAB

Po předchozím nastavení programu Tomcat se umístí webový archív *jmc.war* přímo do adresáře Tomcatu *%tomcat%/webapps*. Pro jednoduchou dekomprimaci lze pak jen spustit Tomcat (*%tomcat%/bin/startup.bat*) a jakmile se vše načte zase jej ukončit. Všechny následující konfigurace v této podsekci se budou týkat jen souborů uvnitř adresářů *jmc/WEB-INF* a *jmc/WEB-INF/cfg*.

V souborech *jmc-flow.xml* a *jmc-servlet.xml* se musí nastavit většina cest k fyzickým adresářům uvnitř služby *directoryService*[4]. Cesty mapují složky, kde se budou ukládat: soubory uživatelů, analýzy, m-fily apod. Dostačující bude upravit jen začátek cesty[5]. Hodnoty *usersFilesSuffix* a *usersResultsSuffix* není potřeba měnit. Význam všech složek je uveden v tab. 5.1.

Dalším úkolem je nastavení konektorů. Ve složce *cfg* se nalézají informace o konektorech. Soubor *connectorsCFG.xml* obsahuje veškeré informace o druhu a způsobu spojení s MATLABem. Tato spojení se pokusí vytvořit aplikace při řízené inicializaci přes webové rozhraní. Ve výpisu 5.3 je uveden obsah tohoto souboru. Tento příklad znázorňuje dva konektory. První (*jmatlinkLocal1*), jak už z názvu

[3]Namísto %tomcat% se dosadí absolutní cesta k tomuto programu, která je závislá od místa instalace, např.: *D:/Apache/Apache Tomcat 6.0.26/*

[4]V obou souborech se jedná o identickou službu. V druhém souboru lze tedy nahradit jen kopií první.

[5]Například z ***D:/javaprojekty/****jmc/web/analyses/* na ***C:/tomcat/webapps/****jmc/analyses*

Tab. 5.1: Přehled významu adresářů.

Název cesty	Význam
analysesPath	Cesta k adresáři s analýzami. Musí končit složkou *analyses*.
cfgpath	Cesta k adresáři s konfiguracemi.
mfilespath	Cesta k adresáři s m-fily.
userspath	Cesta k adresáři, ve kterém jsou uloženy adresáře s uživateli.
tmpAnalysespath	Adresář pro dočasně vytvořené analýzy (při tvorbě a editaci analýzy).
pathToLog	Cesta k logovacímu souboru. Implicitně u programu Tomcat v adresáři *%tomcat%/bin/logfile.log*. Jméno souboru je měnitelné v konfiguračním souboru *WEB-INF/classes/log4j.properties*.
pathToConnectorsConfig	Cesta ke konfiguračnímu souboru s konektory.
usersFilesSuffix	Mezilehlý adresář pro ukládání uživatelových souborů.
usersResultsSuffix	Mezilehlý adresář pro ukládání výsledků analýz.

a ip napovídá, je pro lokální spojení typu JMatLink. Druhý je zase konektor ke vzdálenému spojení. U obou konektorů je uvedeno uživatelské jméno a heslo pro přístup k FTP. FTP je využito pro distribuci souborů mezi webovým a výpočetním serverem u všech druhů spojení. Význam jednotlivých atributů lze nalézt v tab. 5.2.

5.1.2 Výpočetní servery

Výpočetní server musí být pochopitelně vybaven prostředím MATLAB a metodou pro spojení Java-MATLAB. Do operačního systému se tedy musí nejdříve nahrát knihovny aplikace JMatLink. Konkrétně knihovna *JMatLink.dll* podle pokynů z kapitoly 2.2 nebo podle [12].

Místem pro nastavení konektoru je opět konfigurační soubor[6], proto je nutné takovým souborem opatřit každý server[7]. Struktura tohoto souboru je podobná jako ve výpisu 5.3, ovšem pro serverovou část se využívá jen spojení typu *rmiServer* (ostatní se ignorují). Důležitými atributy jsou *id*, *connectorType*, *connectionType* a *port*, které musí souhlasit s klientskou částí a adresář *matlabDir*, do kterého se přesune pracovní plocha MATLABu ihned po vytvoření spojení. Není vhodné namapo-

[6]Ukázková konfigurace je k nalezení v souboru *jmc/WEB-INF/cfg/serverConnectorCFG.xml*.
[7]Pochopitelně každý se svou specifickou konfigurací korespondující s klientskou částí.

Tab. 5.2: Přehled atributů konektorů.

Atribut	Význam
id	Jednoznačný identifikátor spojení. Pokud se jedná o typ *rmiClient* musí být id ekvivalentní s id vzdáleného *rmiServer* spojení.
connectorType	Druh konektoru. Zatím je podporován jen konektor typu *jmatlink* (viz kap. 2.2).
connectionType	Druh spojení. Tři druhy: *local*, *rmiClient*, *rmiServer*. Pro klientskou část (webový server) se využívají první dva (třetí je ignorován). *Local* vytvoří lokální spojení s MATLABem. *RmiClient* se pokusí kontaktovat zadanou *ip* a vyhledat spojení typu *rmiServer* se stejným *id*.
port	Port na kterém běží RMI aplikace. Lze libovolně měnit. Musí být jak na straně serveru, tak na straně klienta stejná.
ip	Ip adresa RMI a FTP serveru. Nemá význam pro *rmiServer*.
matlabDir	Tento adresář se vytvoří a přepne se do něj pracovní plocha MATLABu při nově vytvořeném spojení typu *rmiServer* nebo *local*. Pro *rmiClient* nemá význam.
ftpDir	Po připojení k FTP serveru se klient přepne do tohoto adresáře. Nemá význam pro *rmiServer*.
ftpPass	Heslo k FTP serveru. Nemá význam pro *rmiServer*.
ftpUser	Přihlašovací jméno k FTP serveru. Nemá význam pro *rmiServer*.

Výpis 5.3: Konfigurační soubor konektorů *connectorsCFG.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<connectors xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="connectorsCFG.xsd" >
    <!-- První konektor -->
    <connector id="jmatlinkLocal1" connectorType="jmatlink"
        connectionType="local"
    port="1234" ip="127.0.0.1" matlabDir="C:/jmcTEMP/" ftpDir="jmcTEMP"
         ftpPass="hesloheslo" ftpUser="uzivatel"/>
    <!-- Druhý konektor -->
    <connector id="jmatlinkRMI1" connectorType="jmatlink"
        connectionType="rmiClient"
      port="1234" ip="192.168.14.101" matlabDir="-" ftpDir="jmcTEMP"
          ftpPass="hesloheslo" ftpUser="uzivatel"/>
</connectors>
```

vat tento adresář přímo do složky s prostředím MATLAB, protože se do této složky budou nahrávat a mazat soubory, mohlo by tedy dojít k nenávratným ztrátám. Na zbylých atributech, týkajících se FTP, nezávisí, ale je potřeba je vyplnit[8].

Každý výpočetní server musí být také vybaven FTP serverem. Pro testovací účely postačí například [16]. Vytvoří se účet pod heslem stejným, jako se vyplnilo v klientské části konfigurace (atributy: *ftpUser* a *ftpPass*). Domovský adresář tohoto účtu je potřeba nastavit tak, aby se klient po přesunutí do složky *ftpDir* ocitl ve stejném adresáři jako byla nastavena v konfiguraci předešlého bodu (*matlabDir* v konfiguraci konektoru na výpočetním serveru) a povolit v této složce práva na mazání, ukládání, stahování a vytváření adresářů a souborů.

Po těchto krocích lze jen spustit grafické rozhraní *jmcServerGui.jar* a přes menu *Main -> Start* vybrat pomocí dialogového okna konfigurační soubor s vytvořeným nastavením a spustit. Jako ověření by se mělo po chvíli vytvořit okno s příkazovým řádkem MATLABu a vypsat se kontrolní zpráva v textovém poli grafického rozhraní, viz obr. 5.1.

5.2 Popis funkcí

V této sekci bude popsáno několik vlastností, specifik a návodů týkajících se systému.

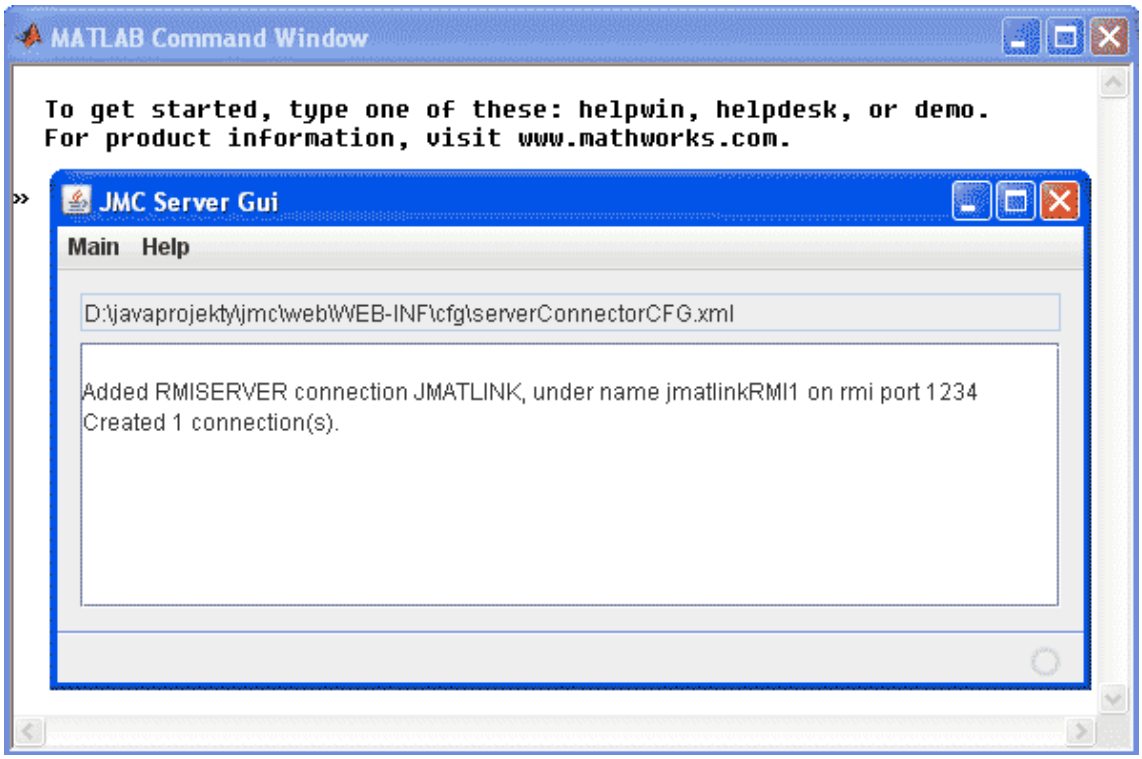
[8]Například pomlčkou nebo podle klientské části.

Tab. 5.3: Přehled atributů kořenového elementu.

Atribut	Význam
name	Jméno analýzy a zároveň i XML souboru. Bez mezer.
mfile	Název m-souboru pro tuto analýzu. Nemusí souhlasit s názvem funkce, ale je to doporučeno.
description	Popis metody. Používá se pro zobrazení detailů ve webovém rozhraní.
estimationTime	Odhadovaná doba běhu celé analýzy. Používá se k časovanému přesměrování na prohlížení výsledků u webového rozhraní.

Tab. 5.4: Přehled atributů vstupních a výstupních elementů.

Atribut	Význam
name	Jméno elementu. Ekvivalentní s názvem proměnné.
description	Popis elementu. Používá se pro zobrazení detailů ve webovém rozhraní.
type	Může nabývat hodnot *complex*/*real* – očekávaný vstup/výstup bude komplexní/reálné číslo. Tyto elementy budou přímo vystupovat ve funkci předávané do MATLABu ([vystup1, vystup2]=funkce(vstup1,vstup2)). U typu *file* je situace složitější a tento element není přímo předáván do MATLABu. Pokud se jedná o **vstupní** element, pak to znamená, že vstup od uživatele bude soubor (akceptovatelné formáty souboru můžou být upřesněny v atributu specifikátor). Pokud se jedná o **výstupní** element pak je zapotřebí **vždy nastavit specifikátor**. Pokud je specifikátor nastaven na některý z formátů pro obrázek (PS, EPS, SVG, PNG, PCX, EMF, JPG, TIF, BMP, PDF), pak se tento obrázek vytvoří automaticky (inkrementačně od 1 se pro každý takový element bude program snažit uložit okno *figure* se stejným číslem – je nutné dodržet tuto konvenci v m-filech). Pokud soubor nebude klasifikován jako obrázek, pak se program bude snažit získat soubor se jménem tohoto elementu a formátem daným specifikátorem (např. *vystup.wav*), předpokládá se tedy vznik souboru při běhu funkce.
value	Předdefinovaná hodnota, nepůjde změnit uživatelem. Pouze u vstupních elementů.
specifier	Specifikátor využívaný hlavně k definici souborového formátu u typu *file*. U jiných typů nemá význam.



Obr. 5.1: JmcServerGui po inicializaci konektoru.

5.2.1 Tvorba modulů

Každá analýza potřebuje svůj konfigurační soubor. Ukázka takového souboru je patrná z výpisu 5.4. Přehled atributů analýzy v tabulce 5.3. Tato ukázka je zaměřena na výpočet determinantu matice. Kromě hlavičky a hlavního elementu jsou důležité elementy *in* a *out*. Každý takto vnořený element znamená jeden vstup nebo výstup, upřesnění každého vstupu nebo výstupu je definováno jeho atributy. Přehled atributů je v tabulce 5.4. V souboru je nastaven jeden vstupní parametr s názvem *input*. Podle popisu si lze snadno představit k čemu tento vstupní element bude sloužit. Jedná se o vstupní matici reálných čísel od uživatele, ze které se bude počítat determinant. Druhý element s názvem *output* pak představuje výstup ve formě reálného čísla. Je nutno zmínit, že systém tuto proměnnou po návratu z MATLABu uloží automaticky do textového souboru specifického formátu, kterým se zabývá následující sekce.

Samotná tvorba analýzy přes webové rozhraní je pak díky těmto znalostem snadná. Jde totiž jen o editaci konfiguračního souboru analýzy a doplnění funkce pro MATLAB (m-file).

Výpis 5.4: Konfigurační soubor analýzy pro výpočet determinantů, soubor *determinants.xml*.

```
<?xml version="1.0" encoding="UTF-8"?>
<analysis xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="analysis.xsd"
 name="determinants" mfile="determinants" description="Computes
     determinant of matrix." estimationTime="10">
    <!-- Vstupní element -->
    <in name="input" type="real" description="Input matrix from which a
         determinant will be computed." />
    <!-- Výstupní element -->
    <out name="output" type="real" description="The determinant of
        matrix."/>
</analysis>
```

Výpis 5.5: Ukázka číselného formátu.

```
2x3;
1.1, 2.2, 3.3;
4.4, 5.5, 6.6;
```

5.2.2 Vstupní a výstupní formát čísel

Systém pracuje s podobným číselným formátem jako v prostředí MATLAB (styl anglické literatury). Hlavní specifikum tkví v psaní desetinných teček namísto čárek. Čísla je možné buď vyjádřit jako skaláry nebo je možné je psát do maticového formátu vytvořeného pro tento projekt. Ukázka tohoto formátu je umístěna ve výpisu 5.5. První řádek slouží k pevné definici rozměrů matice a k jejích kontrole programem. Následující řádky už jsou přímo řádky matice (vektory). Každý řádek zápisu je ukončen středníkem pro snadné rozpoznání programem. Jednotlivé hodnoty jsou odděleny čárkou.

Komplexní čísla

Aplikace obsahuje metody pro čtení komplexních čísel z textových řetězců. Označení imaginární jednotky může být reprezentováno písmeny j nebo i a to jak před číslem, tak i za číslem. Výstupní formát používá označení imaginární jednotky jako j a umísťuje se za číslo. Opět platí stejná pravidla jako v předchozím bodě – maximálně matice, desetinné tečky, oddělovače. Ukázka ve výpisu 5.6.

Výpis 5.6: Ukázka číselného formátu komplexních čísel.

```
2x3;
1.1j, 2.2i, -j3.3;
-i4.4, j, i;
```

5.2.3 Jazyková lokalizace

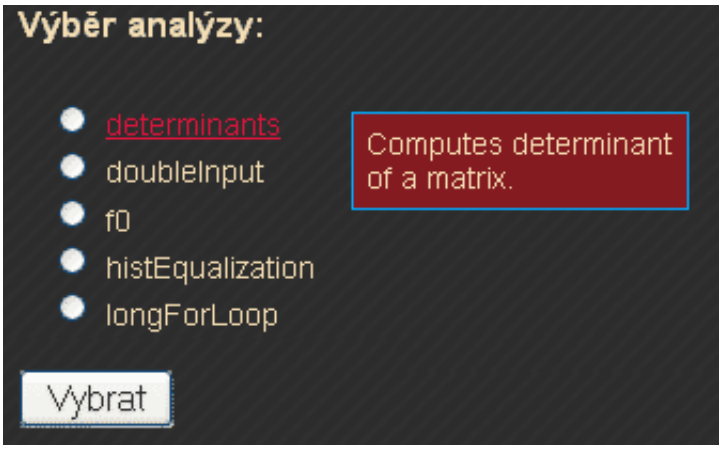
Projekt je lokalizován do 2 jazyků – angličtina a čeština. Veškerý text stránek, kromě chybových hlášení, které nejsou lokalizovány, je umístěn pro oba jazyky v souborech **.properties* uvnitř adresáře *WEB-INF/classes/ – messages_cs a messages_en*. Webový server by se měl automaticky postarat podle jazykového nastavení webového prohlížeče uživatele na jaký jazyk přepnout. Tuto techniku obstarávají značky uvnitř *jsp* souborů *<spring:message>*. Ve výpisu 4.4 lze vypozorovat jejich použití, kde hodnota atributu *code* odkazuje na text pod tímto klíčem uvnitř souborů *messages_*.properties*.

5.2.4 Spuštění první analýzy

Po nastavení a nainstalování všeho potřebného (kapitola 5.1) je možné přejít ke spuštění první analýzy. Nejdříve je samozřejmě nutné zapnout webový server Apache Tomcat (*%tomcat%/bin/startup.bat*). Ve webovém prohlížeči se zadá adresa < http://localhost/jmc/>. Pokud bylo vše řádně nastaveno, mělo by se zobrazit webové rozhraní. Nyní se musí inicializovat spojení, která jsou nastavena v konfiguračním souboru *jmc/WEB-INF/cfg/connectorsCFG.xml*. Pro počáteční otestování správné konfigurace aplikace je doporučeno mít v konfiguračním souboru pro konektory jen 1 lokální spojení (odpadá tvorba spojení typu rmiServer pomocí gui *jmcServerGUI*). K uskutečnění inicializace konektorů je zapotřebí mít administrátorská práva, proto se musí do systému vstoupit pod účtem s těmito právy – admin:admin. Přihlášením pod tímto účtem se zpřístupní menu *Inicializovat spojení s vzdálenými konektory*, kde lze vybráním nabídky *start* a příkazem *spustit* vytvořit tato spojení. Pokud se do 20s spojení nevytvoří (nevytvoří se okno s příkazovým řádkem MATLABu – *Matlab Command Window*) je vhodné zkontrolovat logovací soubor přes menu *Prohlédnout log*. Příčinou nevytvoření spojení může být chybná konfigurace konektoru (validovat soubor oproti XSD šabloně vhodným software) nebo nenačtení knihovny JMatLink (zkontrolovat umístění knihovny *JMatLink.dll* a restartovat PC). Tato inicializace je nutná pouze jedenkrát za běh aplikace.

Nyní je možné spouštět[9] analýzy. Přes menu *Spustit analýzu* je nabídnuta volba

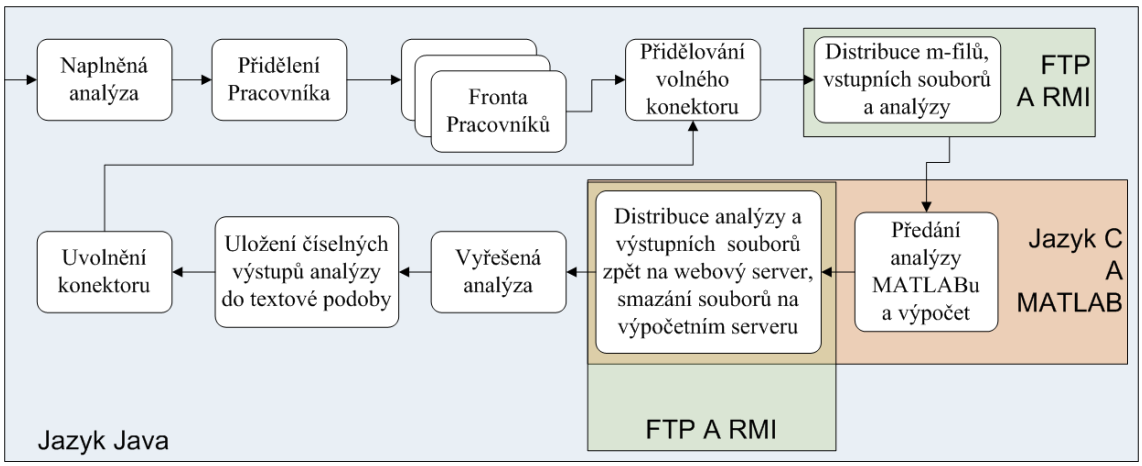
[9]Toto je možné i bez předchozí inicializace, ale po dokončení naplnění analýzy naskočí chybová zpráva o nutnosti inicializace. Analýza však bude bezpečně uložena a bude čekat na volný konektor.



Obr. 5.2: Webové rozhraní – výběr analýzy.

všech dostupných analýz umístěných v adresáři *jmc/analyses/*, náhled viz obr. 5.2. Výběrem této volby a potvrzením se provede ověření naplnění analýzy. Pokud má analýza vyplněny veškeré vstupní elementy přejde se k jejímu uložení do fronty pro zpracování. Pokud analýza není naplněna dojde k výzvě a výběru naplnění každého zbylého elementu. Pokud je vybraný element soubor, nabídne systém výběr souboru s koncovkou uvedenou ve specifikátoru tohoto elementu (např. analýza *histEqualization*). Jestli je vybraný element nějaký druh čísla (reálné nebo komplexní) je nabídnuto textové pole pro vložení čísla (nebo matice čísel podle formátu z kapitoly 5.2.2) s ošetřením výjimek.

Jak už bylo zmíněno, tak po naplněný všech vstupních elementů se analýza uloží do fronty a bude čekat na volný konektor. Uživateli je po tomto přidání vypsáno za jak dlouho bude analýza vypočtena (atribut analýzy *estimationTime*) a po uplynutí tohoto času se jeho prohlížeč automaticky přesune k výpisu výsledků. Jakmile je analýze přiděleno spojení, tak se konektor postará o veškerou distribuci souborů mezi výpočetním serverem a webovým serverem a o spuštění analýzy a navrácení provedené analýzy s výstupními daty. Číselné data pak jsou uloženy do textového souboru. Schéma postupu je znázorněno na obr. 5.3.



Obr. 5.3: Schéma běhu analýzy.

6 ZÁVĚR

Cílem práce bylo prozkoumat problematiku komunikace programovacího jazyka Java a prostředí MATLAB a navrhnout systém, který dokáže využívat těchto prostředků. Navíc systém připravit na uživatelské požadavky přes webové rozhraní, které bude jejich úkony zobrazovat v různých formách.

V první části práce je uveden rozbor možných druhů požadované komunikace. Byly popsány 4 druhy komunikací s ohledem na jejich aktuálnost a implementační schůdnost. Prvním typ, pomocí třídy *Runtime*, přistupuje k příkazovému řádku konkrétního systému a umožňuje spuštění jakéhokoliv programu, tedy i zmiňovaného MATLABu. Tento druh komunikace nabízí pouze práci na úrovni souborů. Druhým typem je komunikace pomocí nativních částí kódu, tedy kódů v jiných programovacích jazycích než je zadaný jazyk Java. Prostředí MATLAB totiž obsahuje knihovny pro interakci z programů v jazyku C a tento fakt je v této sekci popsán. Dalším druhem je využití nezdokumentovaného rozhraní Java-to-Matlab Interface (JMI), které obsahuje každá distribuce MATLABu přímo pro programovací jazyk Java. Posledním typem je tvorba kompletních Java tříd pomocí kompilátoru *MATLAB Builder JA*. Tento kompilátor převede jakýkoliv m-soubor na Java třídy, které lze spustit kdekoliv s nainstalovaným a volně šiřitelným prostředím *MATLAB Compiler Runtime*.

Druhá sekce se zabývá testování open source projektů, které využívají vybrané druhy komunikace. Mezi vybrané druhy komunikace patří pouze dva zmiňované, a to použití JNI a JMI. Z nabídky projektů byly vybrány projekty JAMAL, JMatLink a MatlabControl. Protože každý projekt má rozdílnou architekturu, bylo nutné navrhnout rozhraní, díky kterému se daly tyto metody otestovat. Srovnáním těchto projektů se dospělo k závěru, že nejlépe v ohledu rychlostního zpracování byla metoda využívající nativní části kódu – JMatLink, ovšem na druhou stranu projekt nepodporuje distribuované výpočty. Další dva projekty podporují tento faktor, ale mají rovněž svá omezení. Z tohoto důvodu byla ponechána možnost výběru mezi spojeními, ale dále je využívána pouze metoda JMatLink.

Následující kapitola navrhuje požadovaný systém. Celý systém byl rozdělen na 3 části – uživatele, webové rozhraní a výpočetní servery. Ke každé části jsou uvedeny návrhy na funkcionalitu a konfigurační požadavky. Tato sekce dokumentuje i návrh na jednotnou definici výpočetních modulů formou XML dokumentů, které si získaly popularitu svou vysokou informační hodnotou, rozšířitelností a přenositelností. Takové dokumenty lze navíc i ošetřovat pomocí validace a tím zamezit tvorbě chyb.

Čtvrtá část krátce představuje aplikační rámec Spring, který je použit při implementaci systému, a jednu pokročilou techniku při tvorbě webové aplikace. Veškeré informace v této sekci nezachází příliš do detailů, ale snaží se čtenáři tento rámec

přiblížit tak, aby vyjasnil smysl jeho nasazení a orientaci v konfiguračních souborech.

Poslední kapitola se zaměřuje na instalaci a konfiguraci vytvořeného systému a osvětlení některých základů práce a specifik této aplikace. Dále je zde uveden formát pro vyjadřování čísel uvnitř systému a možnost psaní matic a komplexních čísel. V neposlední řadě je zde znázorněn postup pro spuštění první analýzy.

Celá aplikace se zdrojovými kódy, potřebnými programy a elektronická forma tohoto dokumentu je umístěna v příloze na CD. Práce by se dala rozšířit o registraci nových uživatelů a s tím související i zlepšení zabezpečení, například využitím hashovacích funkcích pro uložení hesel (jen jejich otisků). Bezpochyby by se systému hodila i databáze uchovávající důležité informace ať už o uživatelích, tak například o analýzách.

LITERATURA

[1] ALTMAN, Yair. *Undocumented Matlab* [online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://undocumentedmatlab.com/>.

[2] BURD, Barry. *JSP: JavaServer Pages : Podrobný průvodce.* Praha : Computer Press, 2003. 416 s. ISBN 80-7226-804-X.

[3] CROFT, Jeff; LLOYD, Ian; RUBIN, Dan. *Mistrovství v CSS : Pokročilé techniky pro webové designéry a vývojáře.* Brno : Computer Press, 2007. 416 s. ISBN 978-80-251-1705-7.

[4] HEROUT, Pavel. *Java – bohatství knihoven : 2. upravené a rozšířené vydání.* České Budějovice : KOPP, 2007. ISBN 978-80-7232-307-4

[5] HEROUT, Pavel. *Java a XML.* České Budějovice : KOPP, 2007. ISBN 978-80-7232-307-4.

[6] HEROUT, Pavel. *Učebnice jazyka Java : Třetí rozšířené vydání.* České Budějovice : KOPP, 2008. ISBN 987-80-7232-323-4.

[7] KAPLAN, Joshua. *MatlabControl* [online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://code.google.com/p/matlabcontrol/>.

[8] KHADKEVICH, Maksim. *JAMAL* [online]. 2010 [cit. 2011-05-10]. Dostupné z WWW: <http://jamal.sourceforge.net/>.

[9] KIZIL, Ali. *MATLAB Server Pages* [online]. Istambul : 2007 [cit. 2011-05-10]. Dostupné z WWW: <http://msp.sourceforge.net/>.

[10] KLIMKE, Andreas. *How to acces Matlab from Java* [online]. SRN: Universität Stuttgart, 2005 [cit. 2011-05-10]. Dostupné z WWW: <http://preprints.ians.uni-stuttgart.de/downloads/2003/2003-005.pdf>. [cit. 2010]

[11] MATHWORKS, INC., *MATLAB*. 1984-2011 [cit. 2011-05-10].

[12] MU̇LLER, Stefan. *JMatLink* [online]. 2005 [cit. 2011-05-10]. Dostupné z WWW: <http://jmatlink.sourceforge.net/>.

[13] ORACLE. *Java SE 6 Documentation* [online]. 2011 [cit. 2011-05-10]. JDK 6 Remote Method Invocation (RMI)-related APIs & Developer Guides. Dostupné z WWW: <http://download.oracle.com/javase/6/docs/technotes/guides/rmi/>.

[14] WALLS, Craig; BREIDENBACH, Ryan . *Spring in Action: Second Edition* . [s.l.]: Manning Publications Co., 2008. ISBN 1-933988-13-4.

[15] *Apache Tomcat* [online]. 1999, 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://tomcat.apache.org/>.

[16] *Cerberus FTP Server*[online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://www.cerberusftp.com/>.

[17] *Spring framework* [online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://www.springframework.org/>.

[18] *Tomcat – SSL Configuration HOW-TO* [online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://tomcat.apache.org/tomcat-5.5-doc/ssl-howto.html>.

[19] *XML Tutorial* [online]. 2011 [cit. 2011-05-10]. Dostupné z WWW: <http://www.w3schools.com/xml/default.asp>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

AOP	Aspect Oriented Programming – Aspektově orientované programování
API	Application Programming Interface – Rozhraní pro programování aplikací
CSS	Cascading Style Sheets – Kaskádové styly
DTD	Document Type Definition – Definice typu dokumentu
HTTPS	Hypertext Transfer Protocol Secure – Zabezpečený protokol HTTP
JAR	Java ARchive – Souborový archív pro programovací jazyk Java
JMI	Java-to-MATLAB Interface – Java rozhraní pro přístup k programu MATLAB
JNI	Java Native Interface – Java rozhraní pro komunikaci s nativními kódy
JRE	Java Runtime Enviroment – Prostředí pro spuštění programů napsaných v jazyku Java
JSDK	Java Software Development Kit – Balíček pro tvorbu Java programů
JSP	Java Server Pages – Webové stránky na bázi Javy
JVM	Java Virtual Machine – Virtuální stroj jazyka Java
MCR	MATLAB Compiler Runtime – Prostředí pro spuštění programů zkompilovaných kompilátorem MATLABu
OSS	Open Source Software – Software s otevřeným zdrojovým kódem
RMI	Remote Method Invocation – Vzdálené volání metod
XML	Extensible Markup Language – Rozšiřitelný značkovací jazyk
XSD	XML Schema Definition – Schéma popisující strukturu XML souboru

SEZNAM PŘÍLOH

A Obsah CD **48**

B Adresářová struktura projektu **49**

C Zdrojové kódy použité pro testování **50**

C.1 JAMAL 50
C.2 MatlabControl 52
C.3 Měření uplynulého času 54
C.4 Rozhraní pro analýzy 55
C.5 Třída pro analýzu 55
C.6 Element analýzy 56
C.7 Vstupní a výstupní elementy 56
C.8 Spojovací třída 56
C.9 Měření úkonů 58
C.10 Testovací analýzy 59
C.11 Testovací třída 60
C.12 Analýza Dvojnasobek 60

A OBSAH CD

Obsah CD:

- Apache Tomcat – instalační soubor s programem Apache Tomcat verze 6.0.32.
- Cerberus – instalační soubor s programem Cerberus FTP Server verze 3.0.0.
- Elektonická verze dokumentu – pdf s touto prací a zdrojové *.tex* soubory.
- Jmatlink – knihovna *JMatLink.dll* pro Windows 32-bit.
- JMC Server GUI – grafické rozhraní pro spuštění *rmiServer* druhů spojení
- JSDK – instalační balíček JSDK verze 1.6.25. Instalační soubor *jdk-6u25-windows-i586.exe*.
- Webový archiv s aplikací – archiv *jmc.war* se všemi zdrojovými kódy a konfiguracemi.

B ADRESÁŘOVÁ STRUKTURA PROJEKTU

Obsah archívu *jmc.war*:

- analyses – adresář pro ukládání analýz, dočasných analýz (editace a tvorba analýzy) a m-filů. Lze změnit v konfiguračních souborech *jmc-flow.xml* a *jmc-servlet.xml* uvnitř třídy s *id=directoryService*. Adresář pro analýzy i dočasné analýzy musí obsahovat XSD šablonu.
- css – místo pro uložení stylů a obrázků pro webové stránky.
- jsp – složka se všemi webovými stránkami.
- users – adresář pro uložení uživatelů. Každému uživateli se vytvoří struktura pro ukládání jeho nahraných souborů (*files*) a pro uložení výsledků z analýz (*result*). Názvy adresářů jdou změnit jako u prvního bodu v konfiguračních souborech. Adresář by ale měl být pod složkou *%tomcat%/webaps/jmc/*, tak jak je tomu v základním nastavení kvůli mapování na webových stránkách.
- Web-INF – Adresáře: *cfg* (konfigurace konektorů), *classes* (zdrojové kódy), *flows* (konfigurace pro webové dialogy WebFlow), *lib* (knihovny). Dále soubory: *jmc-flow.xml* (konfigurace pro WebFlow), *jmc-servlet.xml* (konfigurace pro url mapování, zacházení s chybami, limit velikosti souborů pro nahrávání a lokalizace), *jmc-security.xml* (konfigurace zabezpečení, účty a jejich práva, Acegi filtry), *web.xml* (základní konfigurace webové aplikace, povolení filtrů Acegi zabezpečení, zahrnutí kontextů z konfiguračních souborů pro zabezpečení, WebFlow a mapování...).

C ZDROJOVÉ KÓDY POUŽITÉ PRO TESTOVÁNÍ

C.1 JAMAL

Výpis C.1: Implementace v projektu JAMAL

```
package com.jamal.client;
public class MatlabClientImpl implements cz.vutbr.fekt.utko.xstefe05.matlabconnector.
    IMatlabConnection {
    private MatlabCaller matlabCaller;
    private Object[] output;
    private Stopky s~= new Stopky();
    public MatlabClientImpl() {
    }
    public MatlabClientImpl(String host) throws JamalException {
        this(MatlabCaller.HOST_ADDRESS, MatlabCaller.MATLAB_EXECUTABLE_PATH);
    }
    public MatlabClientImpl(String host, String matlabExecutablePath) throws JamalException {
        this(host, matlabExecutablePath, MatlabCaller.TIME_OUT);
    }
    public MatlabClientImpl(String host, String matlabExecutablePath, int timeOut) throws
        JamalException {
        init(host, matlabExecutablePath, timeOut);
    }
    private void init(String host, String matlabExecutablePath, int timeOut) throws
        JamalException {
        try {
            matlabCaller = (MatlabCaller) Naming.lookup(String.format("rmi://%s:%d/%s", host,
                Registry.REGISTRY_PORT, MatlabCaller.SERVICE_NAME));
        } catch (Exception e) {
            boolean successfllyConnected = false;
            if (host.equalsIgnoreCase("localhost") || host.equals("127.0.0.1")) {
                try {
                    System.out.println(String.format("%sMatlab host %s seems to be dead. Trying
                        to launch MatlabServer on the localhost...", JAMAL_PREFIX, host));
                    String command = String.format("%s -automation -r \"com.jamal.server.
                        MatlabServer;quit;\"", matlabExecutablePath);
                    Runtime.getRuntime().exec(command);
                    //Now wait until Matlab starts
                    for (int i = 0; i < timeOut; i++) {
                        try {
                            Thread.sleep(1000);
                            matlabCaller = (MatlabCaller) Naming.lookup(String.format("rmi://%s
                                :%d/%s", host, Registry.REGISTRY_PORT, MatlabCaller.SERVICE_NAME
                                ));
                            successfllyConnected = true;
                            break;
                        } catch (Exception e1) {
                            System.out.println(String.format("%sTrying to connect to
                                MatlabServer... %d", JAMAL_PREFIX, i));
                        }
                    }
                } catch (IOException e1) {
                    throw new JamalException(String.format("%sMatlab host %s seems to be dead",
                        JAMAL_PREFIX, host), e);
                }
            }
            if (successfllyConnected) {
                System.out.println(String.format("%sSuccessfully connected to MatlabServer!",
                    JAMAL_PREFIX));
            } else {
                throw new JamalException(String.format("%sCould not connect to MatlabServer on
                    host %s", JAMAL_PREFIX, host), e);
            }
        }
    }
    public Object[] executeMatlabFunction(String matlabFunctionName, Object[] inputArgs, int
        numberOfOutputArgs) throws JamalException {
        try {
            output = matlabCaller.executeMatlabFunction(matlabFunctionName, inputArgs,
                numberOfOutputArgs);
```

```
            if (output != null && output.length > 0 && output[0] instanceof Exception) {
                System.out.println(String.format("%s%s", JAMAL_PREFIX, ((Exception) output[0]).
                    getLocalizedMessage()));
                output = new Object[0];
                throw new JamalException("Matlab has thrown an exception ", (Exception) output
                    [0]);
            }

        } catch (RemoteException e) {
            throw new JamalException(String.format("%sError occured while executing Matlab
                function %s", JAMAL_PREFIX, matlabFunctionName), e);
        }
        return output;
    }
    /**
     * Shuts down MatlabServer
     *
     * @throws JamalException exception thrown
     */
    public void shutDownServer() throws JamalException {
        executeMatlabFunction(MatlabFunction.QUIT, null, 0);
    }
    public long createConnectionMC() {
        s.start();
        try {
            this.init(MatlabCaller.HOST_ADDRESS, MatlabCaller.MATLAB_EXECUTABLE_PATH,
                MatlabCaller.TIME_OUT);
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        s.printStop("Jamal createConnectionMC");
        return 1l;
    }
    public void destroyConnectionMC(long identifier) {
        s.start();
        try {
            shutDownServer();
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        s.printStop("Jamal destroyConnection");
    }
    public double[][] receiveVariableMC(String varName) {
        s.start();
        Object[] in = new Object[1];
        Object[] arrayDim = null;
        String arrayDimCmd = "size(" + varName + ")";
        in[0] = arrayDimCmd;
        try {
            arrayDim = executeMatlabFunction("eval", in, 1);
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        Object[] out = null;
        String cmd = varName;
        in[0] = cmd;
        try {
            out = executeMatlabFunction("eval", in, 1);
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        double[][] result = objectToDouble(out, arrayDim);
        s.printStop("Jamal receiveVariableMC");
        return result;
    }
    public void sendCmdMC(String cmd) {
        s.start();
        Object[] in = new Object[1];
        in[0] = cmd;
        try {
            executeMatlabFunction("eval", in, 0);
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        s.printStop("Jamal sendCmdMC");
    }
    public void sendVariableMC(String varName, double[][] data) {
        s.start();
        Object[] inData = new Object[1];
```

```
        inData[0] = varName + "=[" + arrayToString(data) + "]";
        try {
            executeMatlabFunction("eval", inData, 0);
        } catch (JamalException ex) {
            Logger.getLogger(MatlabClientImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        s.printStop("Jamal sendVariableMC");
    }
    public String arrayToString(double[][] x) {
        s.start();
        int capacity = x.length * x[0].length * 10;
        StringBuffer sb = new StringBuffer(capacity);
        String result = "";
        for (int i = 0; i < x.length; i++) {
            for (int j = 0; j < x[0].length; j++) {
                sb.append(Double.toString(x[i][j]) + " ");
            }
            sb.append(";");
        }
        result = sb.substring(0, sb.lastIndexOf(";"));
        s.printStop("Jamal arrayToString");
        return result;
    }
    private double[][] objectToDouble(Object[] o, Object[] rowsAndCols) {
        double[][] result = null;
        int[] rowsCols = new int[2];
        double[] rc = null;
        rc = (double[]) rowsAndCols[0];
        rowsCols[0] = (int) rc[0];
        rowsCols[1] = (int) rc[1];
        double[][] tmpResult = null;
        if (o!= null) {
            for (int i = 0; i < o.length; i++) {
                if (o[i] instanceof double[]) {
                    if (tmpResult == null) {
                        tmpResult = new double[o.length][];
                    }
                    tmpResult[i] = (double[]) o[i];
                }
            }
        }
        result = new double[rowsCols[0]][rowsCols[1]];
        for (int i = 0, x = 0; i < rowsCols[0]; i++) {
            x = i;
            for (int j = 0; j < rowsCols[1]; j++) {
                result[i][j] = tmpResult[0][x];
                x += rowsCols[0];
            }
        }
        return result;
    }
    @Override
    public String toString() {
        return "Jamal";
    }
```

C.2 MatlabControl

Výpis C.2: Implementace v projektu MatlabControl.

```
public class MatlabControlImpl implements cz.vutbr.fekt.utko.xstefe05.matlabconnector.
    IMatlabConnection {
                private Stopky s= new Stopky();
    private RemoteMatlabProxy proxy;
    private final long SLEEPTIME = 50;
    private final int MULTIPLIER = 20;
    public MatlabControlImpl() {
        setDaemon(true);
    }
    public void sendCmd(String cmd) throws MatlabInvocationException {
        proxy.eval(cmd);
    }
    public void sendCmdMC(String cmd) {
```

```
        s.start();
        try {
            Thread.sleep(SLEEPTIME);
            sendCmd(cmd);
        } catch (InterruptedException ex) {
            Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex);
        } catch (MatlabInvocationException ex) {
            try {
                Thread.sleep(SLEEPTIME*MULTIPLIER);
                sendCmdMC(cmd);
            } catch (InterruptedException ex1) {
                Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex1)
                    ;
            }
        }
        s.printStop("MatlabControl sendCmdMC");
    }
    public void sendVariable(String varName, double[][] data) throws MatlabInvocationException {
        Object o~= new Object();
        o~= data;
        proxy.setVariable(varName, o);

    }
    public void sendVariableMC(String varName, double[][] data) {
        s.start();
        try {
            Thread.sleep(SLEEPTIME);
            sendVariable(varName, data);
        } catch (InterruptedException ex) {
            Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex);
        } catch (MatlabInvocationException ex) {
            try {
                Thread.sleep(SLEEPTIME*MULTIPLIER);
                sendVariableMC(varName, data);
            } catch (InterruptedException ex1) {
                Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex1)
                    ;
            }
        }
        s.printStop("MatlabControl sendVariableMC");
    }
    private double[][] receiveVariable(String varName) throws MatlabInvocationException {
        double[][] result = null;
        Object variable = null, dimension = null;
        variable = this.proxy.returningEval(varName, 1);
        dimension = this.proxy.returningEval("size(" + varName + ")", 1);
        result = objectToDouble(variable, dimension);
        return result;
    }
    public double[][] receiveVariableMC(String varName) {
        s.start();
        double[][] result = null;
        try {
            Thread.sleep(SLEEPTIME);
            result = receiveVariable(varName);
        } catch (InterruptedException ex) {
            Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex);
        } catch (MatlabInvocationException ex) {
            try {
                Thread.sleep(SLEEPTIME*MULTIPLIER);
                result = receiveVariableMC(varName);
            } catch (InterruptedException ex1) {
                Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex1)
                    ;
            }
        }
        s.printStop("MatlabControl receiveVariableMC");
        return result;
    }
    public long createConnectionMC() {
        s.start();
        try {
            //Create a factory
            RemoteMatlabProxyFactory factory = new RemoteMatlabProxyFactory();
            //Get a proxy, launching MATLAB in the process
            this.proxy = factory.getProxy();
        } catch (MatlabConnectionException ex) {
            Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
```

```
        s.printStop("MatlabControl createConnection");
        return 11;
    }
    public void destroyConnectionMC(long identifier) {
        s.start();
        try {
            this.proxy.exit();
        } catch (MatlabInvocationException ex) {
            Logger.getLogger(MatlabControlImpl.class.getName()).log(Level.SEVERE, null, ex);
        }
        s.printStop("MatlabControl destroyConnection");
    }
    private double[][] objectToDouble(Object o, Object rowsAndCols) {
        double[][] result = null;
        int[] rowsCols = new int[2];
        double[] rc = null;
        rc = (double[]) rowsAndCols;
        rowsCols[0] = (int) rc[0];
        rowsCols[1] = (int) rc[1];
        double[] tmpResult = null;
        if (o instanceof double[]) {
            tmpResult = (double[]) o;
        }
        result = new double[rowsCols[0]][rowsCols[1]];
        for (int i = 0, x = 0; i < rowsCols[0]; i++) {
            x = i;
            for (int j = 0; j < rowsCols[1]; j++) {
                result[i][j] = tmpResult[x];
                x += rowsCols[0];
            }
        }
        return result;
    }
    @Override
    public String toString() {
        return "MatlabControl";
    }
}
```

C.3 Měření uplynulého času

Výpis C.3: Třída pro měření uplynulého času.

```
package cz.vutbr.fekt.utko.xstefe05.stopky;
public class Stopky {
    private long start;
    private long stop;
    public Stopky() {
        this.start = System.nanoTime();
    }
    public void start() {
        this.start = System.nanoTime();
    }
    public void printStart(String s) {
        this.start = System.nanoTime();
        System.out.println("tic " + s);
    }
    public long stop() {
        this.stop = System.nanoTime();
        return (this.stop - this.start);
    }
    public void printStop() {
        this.stop = System.nanoTime();
        System.out.println((this.stop - this.start) / 1000 + "micros");
    }
    public void printStop(String name) {
        this.stop = System.nanoTime();
        System.out.println(name + " " + (this.stop - this.start) / 1000 + " micros");
    }
}
```

C.4 Rozhraní pro analýzy

Výpis C.4: Rozhraní pro analýzy.

```
package cz.vutbr.fekt.utko.xstefe05.analyses;
public interface IAnalysis {
    public String getName();
    public int getInputNumber();
    public int getOutputNumber();
    public void setInputVariables(Vector<AnalysisElement> in);
    public void setOutputVariables(Vector<AnalysisElement> out);
    public Vector<AnalysisElement> getInputVariables();
    public Vector<AnalysisElement> getOutputVariables();
}
```

C.5 Třída pro analýzu

Výpis C.5: Třída pro analýzu.

```
package cz.vutbr.fekt.utko.xstefe05.analyses;
public class Analysis implements IAnalysis {
    private String name;
    private Vector<AnalysisElement> output;
    private Vector<AnalysisElement> input;
    public Analysis(String name, Vector<AnalysisElement> output, Vector<AnalysisElement> input)
        {
        this.name = name;
        this.output = output;
        this.input = input;
    }
    @Override
    public String toString() {
        String result=getName()+" number of output elements: "+ output.size()+" number of input
            elements: "+ input.size();
        return result;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public int getInputNumber() {
        return this.input.size();
    }
    public int getOutputNumber() {
        return this.output.size();
    }
    public Vector<AnalysisElement> getInputVariables() {
        return this.input;
    }
    public Vector<AnalysisElement> getOutputVariables() {
        return this.output;
    }
    public void setInputVariables(Vector<AnalysisElement> in) {
        this.input = in;
    }
    public void setOutputVariables(Vector<AnalysisElement> out) {
        this.output = out;
    }
}
```

C.6 Element analýzy

Výpis C.6: Element analýzy.

```
package cz.vutbr.fekt.utko.xstefe05.analyses;
public abstract class AnalysisElement {
    protected String name;
    protected Object data;
    public Object getData() {
        return data;
    }
    public void setData(Object data) {
        this.data = data;
    }
    public String getName() {
        return name;
    }
}
```

C.7 Vstupní a výstupní elementy

Výpis C.7: Vstupní a výstupní elementy.

```
package cz.vutbr.fekt.utko.xstefe05.analyses;
public class InElement extends AnalysisElement {
    public InElement(String name, double[][] data) {
        this.name = name;
        this.data = data;
    }
    public InElement(String name) {
        this.name = name;
    }
}
//-------------------------
package cz.vutbr.fekt.utko.xstefe05.analyses;
public class OutElement extends AnalysisElement {
    public OutElement(String name) {
        this.name = name;
    }
}
```

C.8 Spojovací třída

Výpis C.8: Spojovací třída.

```
package cz.vutbr.fekt.utko.xstefe05.matlabconnector;
public class MatlabConnector extends Thread {
    private String connectorName;
    private IMatlabConnection matlabConnection;
    private STATE state = STATE.UNUSED;
    private String cmd;
    private IAnalysis analysis;
    private final String CLOSEALL = "close all";
    private final String CLEANALL = "clear all";
    private double[][] data;
    private Vector<double[][]> outputData;
    private long identifier=0;
    public static enum STATE {RUNNING, IDLE, UNUSED}
    public MatlabConnector(IMatlabConnection matlabConnection) {
        this.matlabConnection = matlabConnection;
        identifier = this.matlabConnection.createConnectionMC();
        this.connectorName= matlabConnection.toString();
    }
    @Override
```

```
    public String toString() {
        return connectorName;
    }
    public void destroyConnection() {
        this.matlabConnection.destroyConnectionMC(identifier);
    }
    public Vector<double[][]> getOutputData() {
        return outputData;
    }
    public void setAnalysis(IAnalysis analysis) {
        this.analysis = analysis;
    }
    public STATE getSTATE() {
        return state;
    }
    private void cleanWorkspace() {
        this.matlabConnection.sendCmdMC(CLEANALL + CLOSEALL);
    }
    @Override
    public void run() {
        this.state = STATE.RUNNING;
        int inputsCount = this.analysis.getInputNumber();
        int outputsCount = this.analysis.getOutputNumber();
        Vector<AnalysisElement> input, output;
        input = this.analysis.getInputVariables();
        output = this.analysis.getOutputVariables();
        String inVarString = "", outVarString = "", inVar, outVar;
        // puts variables to MATLAB
        for (AnalysisElement ae : input) {
            this.data = (double[][])ae.getData();
            inVar = ae.getName();
            inVarString += inVar + ", ";
            this.matlabConnection.sendVariableMC(inVar, data);
        }
        if (inputsCount != 0) {
            inVarString = inVarString.substring(0, inVarString.lastIndexOf(","));
        }
        for (AnalysisElement ae : output) {
            outVarString += ae.getName() + ", ";
        }
        if (outputsCount != 0) {
            outVarString = outVarString.substring(0, outVarString.lastIndexOf(","));
        }
        //puts complete analysis command to MATLAB (like [a]=method(x,y))
        this.cmd = "[" + outVarString + "] = " + this.analysis.getName() + "(" +
                inVarString + ");";
        this.matlabConnection.sendCmdMC(cmd);
        if (outputsCount == 0) {
        } else if (outputsCount > 0) {
            outputData = new Vector<double[][]>(outputsCount);
            for (AnalysisElement ae : output) {
                outVar = ae.getName();
                outputData.add(this.matlabConnection.receiveVariableMC(outVar));
            }
        }
        cleanWorkspace();
        this.state = STATE.IDLE;
        return;
    }
}
```

C.9 Měření úkonů

Výpis C.9: Měření úkonů.

```
package cz.vutbr.fekt.utko.xstefe05.matlabconnector.test;
public class ConnectionTester {
    private Stopky s= new Stopky();
    private Stopky s2= new Stopky();
    private ConnectionAnalysesOut[] cao;
    public ConnectionTester(MatlabConnector[] connections, IAnalysis[] analysis) {
        setTester(connections, analysis);
    }
    public void setTester(MatlabConnector[] connection, IAnalysis[] analysis) {
        this.cao = new ConnectionAnalysesOut[connection.length];
        int i = 0;
        int j = 0;
        for (MatlabConnector c : connection) {
            AnalysesOut[] tmp = new AnalysesOut[analysis.length];
            j = 0;
            for (IAnalysis a : analysis) {
                tmp[j++] = new AnalysesOut(a);
            }
            this.cao[i++] = new ConnectionAnalysesOut(tmp, c);

        }
    }
    public void run() {
        for (ConnectionAnalysesOut c : this.cao) {
            //System.gc();
            s.printStart(c.toString());
            for (AnalysesOut ao : c.getAo()) {
                s2.printStart(ao.toString());
                c.getConnection().setAnalysis(ao.getAnalysis());
                c.getConnection().run();
                ao.setOut(c.getConnection().getOutputData());
                System.out.println(c+" "+ao.toString()+" end");
            }
            c.getConnection().destroyConnection();
            s.printStop(c.toString());
            System.out.println("-----------------------------------");
        }
    }
    public void vypis2DPole(double[][] pole) {
        if (pole != null) {
            for (int i = 0;
                    i < pole.length;
                    i++) {
                for (int j = 0; j < pole[i].length; j++) {
                    System.out.print(pole[i][j] + " ");
                }
                System.out.println();
            }
        }
    }
//////////////////////
    private class AnalysesOut {
        protected IAnalysis analysis;
        protected Vector<double[][]> out;
        public AnalysesOut(IAnalysis a) {
            this.analysis = a;
            this.out = new Vector<double[][]>();
        }
        public IAnalysis getAnalysis() {
            return analysis;
        }
        public Vector<double[][]> getOut() {
            return out;
        }
        public void setOut(Vector<double[][]> o) {
            this.out = o;
        }
        @Override
        public String toString() {
            return analysis.toString();
        }
    }
    private class ConnectionAnalysesOut {
        protected AnalysesOut[] ao;
```

```
        protected MatlabConnector connection;
        public AnalysesOut[] getAo() {
            return ao;
        }
        public MatlabConnector getConnection() {
            return connection;
        }
        public ConnectionAnalysesOut(AnalysesOut[] a, MatlabConnector c) {
            this.ao = a;
            this.connection= c;
        }
        @Override
        public String toString() {
            return connection.toString();
        }
    }
}
```

C.10 Testovací analýzy

Výpis C.10: Testovací analýzy.

```
package cz.vutbr.fekt.utko.xstefe05.matlabconnector.test;
public class TestAnalyses {
    private static Analysis analysis;
    private static double[][] dataCreator(int rows, int cols) {
        double[][] result = null;
        if (rows > 0 && cols > 0) {
            result = new double[rows][cols];
            int tmp = 1;
            for (int i = 0; i < rows; i++) {
                for (int j = 0; j < cols; j++) {
                    result[i][j] = tmp++;
                }
            }
        }
        return result;
    }
    public static Analysis getAnalysisDvojnasobek(int rows, int cols) {
        String name = "dvojnasobek";
        double[][] dataIn = dataCreator(rows, cols);
        Vector<AnalysisElement> in, out;
        in = new Vector<AnalysisElement>();
        out = new Vector<AnalysisElement>();
        in.add(new InElement("vstup", dataIn));
        out.add(new OutElement("vystup"));
        return new Analysis(name, out, in);
    }
    public static Analysis getAnalysisMagic(int n) {
        if (n < 1) {
            return null;
        }
        String name = "magic";
        double[][] dataIn = new double[1][1];
        dataIn[0][0] = n;
        Vector<AnalysisElement> in, out;
        in = new Vector<AnalysisElement>();
        out = new Vector<AnalysisElement>();
        in.add(new InElement("vstup", dataIn));
        out.add(new OutElement("vystup"));
        return new Analysis(name, out, in);
    }
    public static Analysis getAnalysisRot90(int rows, int cols) {
        String name = "rot90";
        double[][] dataIn = dataCreator(rows, cols);
        Vector<AnalysisElement> in, out;
        in = new Vector<AnalysisElement>();
        out = new Vector<AnalysisElement>();
        in.add(new InElement("vstup", dataIn));
        out.add(new OutElement("vystup"));
        return new Analysis(name, out, in);
    }
    public static Analysis getAnalysisFliplr(int rows, int cols) {
```

```
        String name = "fliplr";
        double[][] dataIn = dataCreator(rows, cols);
        Vector<AnalysisElement> in, out;
        in = new Vector<AnalysisElement>();
        out = new Vector<AnalysisElement>();
        in.add(new InElement("vstup", dataIn));
        out.add(new OutElement("vystup"));
        return new Analysis(name, out, in);
    }
}
```

C.11 Testovací třída

Výpis C.11: Testovací třída.

```
package cz.vutbr.fekt.utko.xstefe05.matlabconnector.test;
public class Test {
    public static void main(String[] args) {
        MatlabConnector[] connection = null;
        IAnalysis[] analysis = null;
        connection = new MatlabConnector[3];
        connection[2] = new MatlabConnector(new JMatLink());
        connection[1] = new MatlabConnector(new MatlabClientImpl());
        connection[0] = new MatlabConnector(new MatlabControlImpl());
        int count=20;//count of every analysis run on each connection = 20x each analysis
        int analysisCount=4;
        int dimension=25;//dimension of matrix
        analysis = new IAnalysis[count*analysisCount];
        for (int i = 0, tmp = 0; i < count*analysisCount; i++) {
            tmp = i % analysisCount;
            switch (tmp) {
                case 0:
                    analysis[i]=TestAnalyses.getAnalysisFliplr(dimension, dimension);
                    break;
                case 1:analysis[i]=TestAnalyses.getAnalysisMagic(dimension);
                    break;
                case 2:analysis[i]=TestAnalyses.getAnalysisDvojnasobek(dimension, dimension);
                    break;
                case 3:analysis[i]=TestAnalyses.getAnalysisRot90(dimension, dimension);
                    break;
            }
        }
        ConnectionTester t = new ConnectionTester(connection, analysis);
        t.run();
    }
}
```

C.12 Analýza Dvojnasobek

Výpis C.12: Analýza Dvojnasobek.

```
function [b]=dvojnasobek(a)
b=a.*2;
```