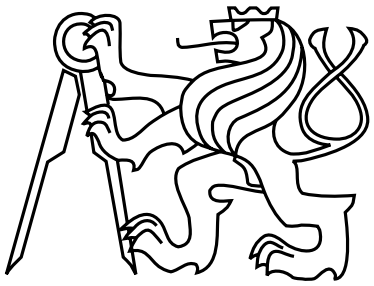
Na tomto místě bude oficiální zadání vaší práce

- Toto zadání je podepsané děkanem a vedoucím katedry,
- musíte si ho vyzvednout na studijním oddělení Katedry počítačů na Karlově náměstí,
- v jedné odevzdané práci bude originál tohoto zadání (originál zůstává po obhajobě na katedře),
- ve druhé bude na stejném místě neověřená kopie tohoto dokumentu (tato se vám vrátí po obhajobě).

České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačů



Diplomová práce

Centrální správa a automatická integrace byznys pravidel v architektuře orientované na služby

Bc. Filip Klimeš

Vedoucí práce: Ing. Karel Čemus

Studijní program: Otevřená informatika, Magisterský

Obor: Softwarové inženýrství

18. března 2018

Poděkování

Zde můžete napsat své poděkování, pokud chcete a máte komu děkovat.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2018 .....

Abstract

Translation of Czech abstract into English.

Abstrakt

Abstrakt práce by měl velmi stručně vystihovat její obsah. Tedy čím se práce zabývá a co je jejím výsledkem/přínosem.
Očekávají se cca 1 – 2 odstavce, maximálně půl stránky.

Obsah

1 Úvod **1**

2 Analýza **3**

2.1 Byznys pravidla . . . 3
2.2 Architektura orientovaná na služby . . . 3
2.3 Problémy . . . 3
2.4 Identifikace požadavků na implementaci frameworku . . . 3
2.5 Shrnutí . . . 3

3 Rešerše **5**

3.1 Architektura orientovaná na služby . . . 5
3.2 Modelem řízená architektura . . . 5
3.3 Aspektově orientované programování . . . 5
3.4 Aspect-driven Design Approach . . . 6
3.5 Stávající řešení reprezentace byznys pravidel . . . 6
3.5.1 Drools DSL . . . 6

4 Návrh **7**

4.1 Formalizace architektury orientované na služby . . . 7
4.1.1 Join-points . . . 7
4.1.2 Advices . . . 7
4.1.3 Pointcuts . . . 7
4.1.4 Weaving . . . 7
4.2 Architektura frameworku . . . 7
4.3 Metamodel . . . 7
4.4 Expression . . . 7
4.5 Registr byznys kontextů . . . 7
4.6 Byznys kontext weaver . . . 7
4.7 Centrální správa byznys kontextů . . . 7

5 Implementace **9**
5.1 Sdílení byznys kontextů mezi službami . . . 9
5.1.1 Schéma síťové komunikace . . . 9
5.1.2 Použité technologie . . . 9
5.1.2.1 Protocol Buffers . . . 9
5.1.2.2 gRPC . . . 9
5.2 Knihovna pro platformu Java . . . 10
5.2.1 Použité technologie . . . 10
5.2.1.1 Apache Maven . . . 10
5.2.1.2 AspectJ . . . 10
5.3 Knihovna pro platformu Python . . . 10
5.3.1 Použité technologie . . . 10
5.3.2 . . . 10
5.4 Knihovna pro platformu Node.js . . . 10
5.4.1 Použité technologie . . . 10
5.4.2 NPM . . . 10
5.5 Systém pro centrální správu byznys pravidel . . . 10
5.5.1 Použité technologie . . . 10
5.5.2 Detekce cyklů . . . 10
5.6 Shrnutí . . . 10

6 Verifikace a validace **11**
6.1 Testování naprogramovaných knihoven . . . 11
6.1.1 Platforma Java . . . 11
6.1.2 Platforma Python . . . 11
6.1.3 Platforma Node.js . . . 11
6.1.4 Continuous Integration . . . 11
6.2 Případová studie: e-commerce systém . . . 12
6.2.1 Model systému . . . 12
6.2.2 Use-cases . . . 12
6.2.3 Byznys kontexty . . . 12
6.2.4 Service discovery . . . 12
6.2.5 Order service . . . 12
6.2.6 Product service . . . 12
6.2.7 User service . . . 12
6.2.8 Nasazení systému pro centrální správu byznys kontextů . . . 12
6.3 Shrnutí . . . 12

7 Závěr **13**
7.1 Analýza dopadu použití frameworku 13
7.2 Budoucí rozšiřitelnost frameworku 13
7.3 Možností uplatnění navrženého frameworku 13
7.4 Další možnosti uplatnění AOP v SOA 13
7.5 Shrnutí 13

A TODO Screenshots **17**

B Seznam použitých zkratek **19**

C Obsah přiloženého CD **21**

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Kapitola 2

Analýza

2.1 Byznys pravidla

2.2 Architektura orientovaná na služby

2.3 Problémy

2.4 Identifikace požadavků na implementaci frameworku

2.5 Shrnutí

[T1]fontenc [utf8]inputenc

Kapitola 3

Rešerše

3.1 Architektura orientovaná na služby

3.2 Modelem řízená architektura

3.3 Aspektově orientované programování

Programování je komplexní disciplína s teoreticky neomezeným počtem možností, jakým programátor může řešit zadaný problém. Ačkoliv každá úloha má své specifické požadavky, za relativně krátkou historii programování se stihlo ustálit několik ideologií, tzv. programovacích paradigmat, které programátorovi poskytují sadu abstrakcí a základních principů [4]. Díky znalosti paradigmatu může programátor nejen zlepšit svou produktivitu, ale zároveň může snáze pochopit myšlenky jiného programátora a tím zlepšit kvalitu týmové spolupráce.

Jedním z nejpopulárnějších paradigmat používaných k vývoji moderních enterprise systémů je nepochybně objektově orientované programování (OOP). To vnímá daný problém jako množinu objektu, které spolu intereagují. Program člení na malé funkční celky odpovídající struktuře reálného světa [3]. Je vhodné zmínit, že objekty se rozumí jak konkrétní koncepty, například auto nebo člověk, tak i abstraktní koncepty, namátkou bankovní transakce nebo objednávka v obchodě. Objekty se pak promítají do kódu programu i do reprezentace struktur v paměti počítače. Tento přístup je velmi snadný pro pochopení, vede k lepšímu návrhu a organizaci programu a snižuje tak náklady na jeho vývoj a údržbu.

Ačkoliv je OOP velmi silným a všestraným nástrojem, existují problémy, které nelze jeho pomocí efektivně řešit. Jedním takovým problémem jsou obecné požadavky na systém, které musejí být konzistentně dodržovány na více místech, které spolu zdánlivě nesouvisí. Příkladem může být logování systémových akcí, optimalizace správy paměti nebo uniformní zpracování výjimek [2]. Takové požadavky nazýváme *cross-cutting concerns*. V rámci OOP

je programátor nucen v ojektech manuálně opakovat kód, který zodpovídá za jejich realizaci. Duplikace kódu vede k větší náchylnosti na lidskou chybu a k vyšším nárokům na vývoj a údržbu daného softwarového systému [1].

Aspektově orientované programování (AOP) přináší řešení na výše zmiňované problémy. Extrahuje obecné požadavky, tzv. *aspekty* do jednoho místa a pomocí procesu zvaného *weaving* je poté automaticky distribuuje do systému. Weaving může proběhnout staticky při kompilaci programu nebo dynamicky při jeho běhu. V obou případech ale programátorovi ulehčuje práci, protože k definici i změně aspektu dochází centrálně a tím je eliminována potřeba manuální duplikace kódu. Je nutno poznamenat, že AOP není paradigmatem poskytujícím kompletní framework pro návrh programu. V ideálním případě je tedy k návrhu systému využita kombinace AOP s jiným paradigmatem. Pro účely této práce se zaměříme na kombinaci AOP a OOP.

Aspekt

Join-point

Pointcut

Advice

Weaving

3.4 Aspect-driven Design Approach

Aspect-driven Design Approach (ADDA)

Vzhledem k požadavkům na implementaci našeho frameworku stanoveným v předchozí kapitole 2 se AOP a na něm stavějící ADDA jeví jako vhodný přístup, který nám pomůže dosáhnout cíle.

3.5 Stávající řešení reprezentace byznys pravidel

3.5.1 Drools DSL

Kapitola 4

Návrh

4.1 Formalizace architektury orientované na služby

4.1.1 Join-points

4.1.2 Advices

4.1.3 Pointcuts

4.1.4 Weaving

4.2 Architektura frameworku

4.3 Metamodel

4.4 Expression

4.5 Registr byznys kontextů

4.6 Byznys kontext weaver

4.7 Centrální správa byznys kontextů

[T1]fontenc [utf8]inputenc

Kapitola 5

Implementace

V této kapitole si popíšeme jakým způsobem byl implementován navržený framework pro platformy Java, Python a Node.js.

5.1 Sdílení byznys kontextů mezi službami

Pro sdílení byznys kontextů mezi jednotlivými službami je potřeba je přenášet po síti ve formátu, který bude nezávislý na platformách jednotlivých služeb.

5.1.1 Schéma síťové komunikace

5.1.2 Použité technologie

5.1.2.1 Protocol Buffers

Pro přenos pravidel byl zvolen open-source formát *Protocol Buffers*[1] vyvinutý společností Google[2]. Umožňuje explicitně definovat a vynucovat schéma dat, která jsou přenášena po síti, bez vazby na konkrétní programovací jazyk. Zároveň je díky binární reprezentaci v přenosu velmi efektivní, oproti formátům jako je JSON nebo XML [5].

5.1.2.2 gRPC

Pro komunikaci byznys kontextů nám nestačí pouze přenosový formát, je potřeba také popsat schéma samotné komunikace. K tomu byl zvolen open-source framework gRPC[3], který staví na technologii Protocol Buffers a poskytuje vývojáři možnost definovat komunikaci pomocí protokolu RPC.

[1] https://developers.google.com/protocol-buffers/
[2] https://www.google.com/
[3] https://grpc.io/

5.2 Knihovna pro platformu Java

5.2.1 Použité technologie

5.2.1.1 Apache Maven

5.2.1.2 AspectJ

5.3 Knihovna pro platformu Python

5.3.1 Použité technologie

5.3.2

5.4 Knihovna pro platformu Node.js

5.4.1 Použité technologie

5.4.2 NPM

Node.js package manager

5.5 Systém pro centrální správu byznys pravidel

5.5.1 Použité technologie

5.5.2 Detekce cyklů

5.6 Shrnutí

Veškerý kód je hostován v centrálním repozitáři ve službě GitHub[4] a je zpřístupněn pod open-source licencí MIT[5]. Knihovny pro jednotlivé platformy tedy lze libovolně využívat, modifikovat a šířit. [T1]fontenc [utf8]inputenc

[4] https://github.com/klimesf/diploma-thesis
[5] http://www.linfo.org/mitlicense.html

Kapitola 6

Verifikace a validace

6.1 Testování naprogramovaných knihoven

6.1.1 Platforma Java

6.1.2 Platforma Python

6.1.3 Platforma Node.js

6.1.4 Continuous Integration

Po celou dobu vývoje jednotlivých knihoven byl využíván nástroj Travis CI[1], který po každém přidáním kódu do repozitáře spouštěl automatizované testy a informoval vývojáře o jejich výsledcích. To umožnilo okamžitě identifikovat změny, které do kódu vnesly chybu.

[1] https://travis-ci.org/

6.2 Případová studie: e-commerce systém

6.2.1 Model systému

6.2.2 Use-cases

6.2.3 Byznys kontexty

6.2.4 Service discovery

6.2.5 Order service

6.2.6 Product service

6.2.7 User service

6.2.8 Nasazení systému pro centrální správu byznys kontextů

6.3 Shrnutí

Kapitola 7

Závěr

7.1 Analýza dopadu použití frameworku

7.2 Budoucí rozšiřitelnost frameworku

7.3 Možností uplatnění navrženého frameworku

7.4 Další možnosti uplatnění AOP v SOA

7.5 Shrnutí

Literatura

[1] FOWLER, M. *Refactoring: improving the design of existing code*. Pearson Education India, 2009.

[2] KICZALES, G. et al. Aspect-oriented programming. In *European conference on object-oriented programming*, s. 220–242. Springer, 1997.

[3] RENTSCH, T. Object oriented programming. *ACM Sigplan Notices*. 1982, 17, 9, s. 51–57.

[4] VAN ROY, P. et al. Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music*. 2009, 104.

[5] VARDA, K. Protocol buffers: Google's data interchange format. *Google Open Source Blog, Available at least as early as Jul*. 2008, 72.

Příloha A

TODO Screenshots

[T1]fontenc [utf8]inputenc

Příloha B

Seznam použitých zkratek

OOP Objektově orientované programování

Příloha C

Obsah přiloženého CD

```
|-- nutfroms-example/       Ukázkový systém využívající knihovnu
|  |-- dist/                Zkompilované zdrojové soubory pro distribuci
|  |-- docs/                Dokumentace
|  |-- src/                 Zdrojový kód aplikace
|
|-- nutforms-ios-client/    Klientská část knihovny pro platformu iOS
|  |-- client/              Zdrojové soubory knihovny
|  |-- clientTests/         Zdrojové soubory testů knihovny
|  |-- dist/                Zkompilované zdrojové soubory pro distribuci
|  |-- docs/                Dokumentace
|
|-- nutfroms-server/        Serverová část knihovny
|  |-- dist/                Zkompilované zdrojové soubory pro distribuci
|  |-- docs/                Dokumentace
|  |-- layout/              Layout servlet
|  |-- localization/        Localization servlet
|  |-- meta/                Metadata servlet
|  |-- widget/              Widget servlet
|
|-- nutforms-web-client/    Klientská část knihovny pro webové aplikace
|  |-- dist/                Zkompilované zdrojové soubory pro distribuci
|  |-- docs/                Dokumentace
|  |-- src/                 Zdrojové soubory knihovny
|  |-- test/                Zdrojové soubory testů knihovny
|
|-- text/                   Text bakalářské práce
```