

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

8. dubna 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 prototypů knihoven	9
5.1	Výběr použitých platforem	9
5.2	Sdílení byznys kontextů mezi službami	10
5.2.1	Síťová komunikace	10
5.2.2	Použité technologie	10
5.2.2.1	Protocol Buffers	10
5.2.2.2	gRPC	11
5.3	Knihovna pro platformu Java	12
5.3.1	Použité technologie	12
5.3.1.1	Apache Maven	12
5.3.1.2	AspectJ	12
5.4	Knihovna pro platformu Python	12
5.4.1	Použité technologie	12
5.4.2		12
5.5	Knihovna pro platformu Node.js	12
5.5.1	Použité technologie	12
5.5.2	NPM	12
5.6	Systém pro centrální správu byznys pravidel	12
5.6.1	Použité technologie	12
5.6.2	Detekce cyklů	12
5.7	Shrnutí	12
6	Verifikace a validace	13
6.1	Testování prototypů knihoven	13
6.1.1	Platforma Java	13
6.1.2	Platforma Python	14
6.1.3	Platforma Node.js	14
6.2	Případová studie: e-commerce systém	14
6.2.1	Model systému	14
6.2.2	Use-cases	14
6.2.3	Byznys kontexty	14
6.2.4	Service discovery	14
6.2.5	Order service	14
6.2.6	Product service	14
6.2.7	User service	14
6.2.8	Nasazení systému pro centrální správu byznys kontextů	14
6.3	Shrnutí	14

7	Závěr	15
7.1	Analýza dopadu použití frameworku	15
7.2	Budoucí rozšiřitelnost frameworku	15
7.3	Možností uplatnění navrženého frameworku	15
7.4	Další možnosti uplatnění AOP v SOA	15
7.5	Shrnutí	15
A	TODO Screenshots	19
B	Seznam použitých zkratk	21
C	Obsah přiloženého CD	23

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 [5]. 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 [4]. 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 všestranným 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 [3]. Takové požadavky nazýváme *cross-cutting concerns*. V rámci OOP

je programátor nucen v objektech 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 paradigmatickým poskytováním kompletní framework pro návrh programu. V ideálním případě je tedy k návrhu systému využita kombinace AOP s jiným paradigmatickým. 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

```
[T1]fontenc [utf8]inputenc
```


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 prototypů knihoven

Součástí zadání této práce je implementace prototypů knihoven pro framework navržený v kapitole 4 pro tři rozdílné platformy, z nichž jedna musí být *Java*. V této kapitole si popíšeme jaké platformy jsme vybraly a jakým způsobem byly prototypy knihoven implementovány. Součástí kapitoly je i stručná rešerše technologií, které byly použity pro dosažení vytyčených cílů.

Pro splnění cílů bylo potřeba vyřešit také několik technických otázek. Těmi je přenos byznys kontextů mezi jednotlivými službami, podpora aspektově orientovaného programování v daném programovacím jazyce a využití principu *runtime weavingu* a integrace knihoven do služeb, které je budou využívat.

5.1 Výběr použitých platforem

Mimo jazyk *Java*, který byl určen zadání, byl pro implementaci vybrán jazyk *Python* a ekosystém *Node.js*, který slouží jako běhové prostředí pro jazyk *JavaScript*. Výběr byl proveden na základě aktuálních trendů ve světě softwarového inženýrství. Projekt GitHub¹ z roku 2014, který shrnuje statistiky repozitářů populární služby pro hosting a sdílení kódu GitHub², určil jazyky *JavaScript*, *Java* a *Python* jako tři nejaktivnější. Služba GitHub následně sama zveřejnila statistiky za rok 2017 v rámci projektu Octoverse³ a dospěla ke stejnému závěru, ačkoliv *Python* se umístil na druhé pozici na úkor jazyka *Java*. Podle průzkumu oblíbeného programátorského webového portálu Stack Overflow⁴ se umístily tyto jazyky v první čtveřici nejpopulárnějších jazyků pro obecné použití.

¹<http://github.info/>

²<https://github.com/>

³<https://octoverse.github.com/>

⁴<https://insights.stackoverflow.com/survey/2017#technology>

5.2 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.

5.2.1 Síťová komunikace

Abychom mohli přenášet byznysové kontexty a jejich pravidla po síti, musíme zvolit protokol a jednotný formát, ve kterém spolu budou jednotlivé služby komunikovat. Tento formát tedy musí být nezávislý na platformě a ideálně by měl být co nejefektivnější v rychlosti přenosu.

5.2.2 Použité technologie

5.2.2.1 Protocol Buffers

Pro přenos byznysových kontextů byl zvolen open-source formát *Protocol Buffers*⁵ vyvinutý společností Google⁶. 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 [6]. Zdrojový kód ?? znázorňuje zápis schématu zasílaných zpráv ve formátu Protobuffer.

Listing 5.1: Příklad definice schématu zpráv v jazyce Protobuffer

```
message Person {
    required string name = 1;
    required int32 id = 2;
    optional string email = 3;

    enum PhoneType {
        MOBILE = 0;
        HOME = 1;
        WORK = 2;
    }

    message PhoneNumber {
        required string number = 1;
        optional PhoneType type = 2 [default = HOME];
    }
}
```

⁵ <https://developers.google.com/protocol-buffers/>

⁶ <https://www.google.com/>

```
}  
  
    repeated PhoneNumber phone = 4;  
}
```

5.2.2.2 gRPC

Pro komunikaci byznys kontextů nám nestačí pouze přenosový formát, je potřeba také popsat schéma samotné komunikace. K tomu byl zvolen open-source framework gRPC⁷, který staví na technologii Protocol Buffers a poskytuje vývojáři možnost definovat komunikaci pomocí protokolu RPC.

Listing 5.2: gRPC

```
// The greeting service definition.  
service Greeter {  
    // Sends a greeting  
    rpc SayHello (HelloRequest) returns (HelloReply) {}  
}  
  
// The request message containing the user's name.  
message HelloRequest {  
    string name = 1;  
}  
  
// The response message containing the greetings  
message HelloReply {  
    string message = 1;  
}
```

⁷ <https://grpc.io/>

5.3 Knihovna pro platformu Java

5.3.1 Použité technologie

5.3.1.1 Apache Maven

5.3.1.2 AspectJ

5.4 Knihovna pro platformu Python

5.4.1 Použité technologie

5.4.2

5.5 Knihovna pro platformu Node.js

5.5.1 Použité technologie

5.5.2 NPM

Node.js package manager

5.6 Systém pro centrální správu byznys pravidel

5.6.1 Použité technologie

5.6.2 Detekce cyklů

5.7 Shrnutí

Veškerý kód je hostován v centrálním repozitáři ve službě GitHub⁸ a je zpřístupněn pod open-source licencí MIT⁹. Knihovny pro jednotlivé platformy tedy lze libovolně využívat, modifikovat a šířit. [T1]fontenc [utf8]inputenc

⁸ <https://github.com/klimesf/diploma-thesis>

⁹ <http://www.linfo.org/mitlicense.html>

Kapitola 6

Verifikace a validace

V této kapitole

6.1 Testování prototypů knihoven

Prototypy knihoven, jejichž implementaci jsme popsali v kapitole 5, byly také důkladně otestovány pomocí sady jednotkových a integračních testů.

V rámci konceptu *continuous integration* [2] byl kód po celou dobu vývoje zasílán do centrálního repozitáře a s pomocí nástroje Travis CI¹ byl automaticky spouštěno jeho sestavení a otestování. Systém zároveň okamžitě informoval vývojáře o jejich výsledcích. To umožnilo v krátkém časovém horizontu identifikovat konkrétní změny v kódu, které do programu vnesly chybu. Tím byla snížena pravděpodobnost regrese a dlouhodobě se zvýšila celková kvalita kódu.

6.1.1 Platforma Java

Prototyp knihovny pro platformu byl testován pomocí nástroje JUnit², který poskytuje všechny potřebné funkce.

¹<https://travis-ci.org/>

²<https://junit.org/junit4/>

6.1.2 Platforma Python

6.1.3 Platforma Node.js

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. – BECK, K. *Refactoring: improving the design of existing code*. Boston, Massachusetts, USA : Addison-Wesley Professional, 1999.
- [2] FOWLER, M. – FOEMMEL, M. Continuous integration. *Thought-Works* <http://www.thoughtworks.com/ContinuousIntegration.pdf>. 2006, 122, s. 14.
- [3] KICZALES, G. et al. Aspect-oriented programming. In *European conference on object-oriented programming*, s. 220–242. Springer, 1997.
- [4] RENTSCH, T. Object oriented programming. *ACM Sigplan Notices*. 1982, 17, 9, s. 51–57.
- [5] VAN ROY, P. et al. Programming paradigms for dummies: What every programmer should know. *New computational paradigms for computer music*. 2009, 104.
- [6] 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

-- nutforms-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
-- nutforms-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