ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ v PRAZE
Fakulta elektrotechnická
Katedra počítačů

Redukce sémantické mezery při integraci externích datových zdrojů

Diplomová práce

Studijní program: Otevřená informatika
Studijní obor: Softwarové inženýrství

Vedoucí práce: doc. Ing. David Šišlák, Ph.D.

Bc. Jiří Miroslav Kačena
Praha 2023

ČVUT
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kačena** Jméno: **Jiří Miroslav** Osobní číslo: **465874**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Redukce sémantické mezery při integraci externích datových zdrojů

Název diplomové práce anglicky:

Semantic gap reduction during integration of external data sources

Pokyny pro vypracování:

Cílem diplomové práce je analýza, návrh, implementace a testování systému, který usnadní integraci datových zdrojů s rozdílnou sémantikou vstupů/výstupů. Řešení bude zapojeno do již existujícího rezervačního systému od společnosti Aaron Goup, který je určený pro letecké společnosti, cestovní kanceláře a cestovní agentury. Navrhované řešení, bude řídit počty transakcí, na získání dat od poskytovatele, které není poskytovatel schopen zaslat v jedné odpovědi. Řešení musí být navrhnuto tak, aby bylo jednoduše konfigurovatelné a aby jej dokázal nastavit uživatel i bez znalosti jakéhokoliv programovacího jazyka. Bude kladen důraz na to, aby bylo možné konfiguraci kdykoliv změnit, bez nutnosti zásahu do kódu. Po vytvoření bude funkčnost systému vyzkoušena uživateli. Tyto testy budou zdokumentovány a vyhodnoceny.

Seznam doporučené literatury:

[1] Hoffmannová, D: Distribuce a prodej letenek v současné letecké dopravě. Vysoká škola obchodní v Praze, 2021.
[2] Schildt, H.: Mistrovství Java. Computer Press, 2014.
[3] Salatino, M., Maio, M., Aliverti, E.: Mastering JBoss Drools 6. Packt Publishing, 2016.
[4] Vojíř, S.: Výběr specifikace business rules pro praktické nasazení v podnikové informatice. Systémová integrace, 3/2015.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. David Šišlák, Ph.D. centrum umělé inteligence FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.09.2022** Termín odevzdání diplomové práce: **10.01.2023**

Platnost zadání diplomové práce: **19.02.2024**

doc. Ing. David Šišlák, Ph.D.
podpis vedoucí(ho) práce

podpis vedoucí(ho) ústavu/katedry

prof. Mgr. Petr Páta, Ph.D.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.

Datum převzetí zadání Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

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 8.1.2023 Podpis . . . .

Poděkování

Chtěl bych vyjádřit poděkování zejména svému vedoucímu diplomové práce doc. Ing. Davidu Šišlákovi, Ph.D., jakožto odbornému konzultantovi, za jeho ochotu a vřelý přístup.

Dále bych rád poděkoval celé společnosti AARON GROUP a svým kolegům. Především Ing. Ondřeji Volrábovi, Ph.D. za jeho ochotu a pomoc s pochopením problematiky.

V neposlední řadě bych rád poděkoval své manželce a rodině za jejich podporu a pomoc během studia.

// Abstrakt

Diplomová práce se zabývá vývojem řešení, které usnadní integraci datových zdrojů s rozdílnou sémantikou vstupů/výstupů. Řešení bylo zapojeno do již existujícího rezervačního systému společnosti AARON GROUP, který komunikuje s mnoha poskytovateli dat. S jeho pomocí je možné řídit počty transakcí pro získání dat od poskytovatele, jež není schopen zaslat v jedné odpovědi. Vytvořená knihovna, která řeší daný problém, je založena na business pravidlech, ve kterých je uložena rozhodovací logika. V knihovně je použit Drools rule engine, který slouží ke spouštění a vyhodnocování business pravidel. Pro vytváření business pravidel byla využita aplikace Business Central, která data ukládá do lokálního Git repozitáře. Pro jejich získání byla vytvořena aplikace Bridge, která daná business pravidla stáhne z Git repozitáře a uloží je do databáze, odkud je může načíst i hlavní systém.

Klíčová slova: Business pravidla, Rule engine, Drools, Rule based systém

Abstract

The diploma thesis deals with the development of a solution that would simplify the integration of data sources with different input/output semantics. This solution was integrated with an already existing reservation system of the AARON GROUP company, which communicates with many data providers. With its help, it is possible to manage the number of transactions to obtain data from the providers, when it is not possible for them to send these in one response. The created library, which solves the given problem, is based on the business rules storing decision logic. The library uses the Drools rule engine to run and evaluate business rules. The Business Central application, that stores data into the local Git repository, was used to create the business rules. To obtain the rules. the Brigde application was created to download the stated business rules from the Git repository and stores them in the database. The main system can retrieve them from there as well.

Keywords: Business rules, Rule engine, Drools, Rule based system

Obsah

1. ÚVOD **13**

1.1. Představení systému na distribuci leteckých nabídek 13

1.2. Představení společnosti AARON GROUP a uvedení do problematiky 15

2. POPIS PROBLÉMU A MOŽNOSTI ŘEŠENÍ **18**

2.1. Přímá implementace 18

2.2. Rozhodovací komponenta 19

2.2.1. Naivní implementace 19

2.2.2. Využití interpretovaných jazyků 20

2.2.3. Rule-base systém 20

2.3. Shrnutí 21

3. BUSINESS PRAVIDLA **22**

3.1. Využití business pravidel 22

3.2. Business rule engine 23

3.3. Business Rule Management System 23

3.4. JSR 94 23

3.5. Užívané specifikace business pravidel 24

3.5.1. OpenL Tablets 24

3.5.2. Easy Rules 24

3.5.3. Jess 25

3.5.4. Open Rules 25

3.5.5. Drools 26

3.6. Shrnutí 26

4. NÁVRH IMPLEMENTACE **28**

5. SYSTÉM DROOLS **29**

5.1. Historie 29

5.2. Popis projektu KIE 30

5.2.1. Drools 30

5.2.2. jBPM 30

5.2.3. OptaPlanner 31

5.2.4. Business Central 31

5.3. Rule engines a Production Rule Systems (PRS) 31

5.4. Životní cyklus provádění pravidel 32

5.5. Algoritmus Rete ... 33
5.5.1. Síť Rete ... 34
5.6. Algoritmus Phreak ... 38
5.6.1. Vyhodnocování pravidel ... 38
5.7. Drools Runtime ... 39
5.8. Struktura pravidel Drools ... 41
5.8.1. Rule ... 42
5.8.2. Levá strana pravidla ... 42
5.8.3. Pravá strana pravidla ... 43

6. GIT ... 44

7. BUSINESS CENTRAL ... 45
7.1. Seznam nástrojů ... 46
7.2. Propojení Business Cental a KieServer ... 47
7.3. Správa uživatelů ... 47
7.4. Průchod aplikací ... 48
7.4.1. Editor business pravidel ... 48

8. IMPLEMENTACE ... 52

9. APLIKACE BRIDGE ... 55

10. VYTVÁŘENÍ PRAVIDEL ... 60
10.1. Problematika zvolení správné Třídy ... 60
10.2. Yaml ... 61
10.3. Výčtový typ ... 61
10.4. Primitivní datové typy a základní třídy Java ... 61
10.5. Návrh objektu DroolsWraper ... 62
10.6. Vytváření pravidla v Business Central ... 65

11. KNIHOVNA BUSINESS REQUEST SPLITTER ... 66
11.1. Bezpečnost ... 67
11.2. Využití metody fireUntilHalt při zacyklení ... 67
11.3. Využití mechanismu Future při zacyklení ... 68
11.4. Zaznamenávání událostí při běhu engine ... 69

12. ZAPOJENÍ KNIHOVNY BUSINESS REQUEST SPLITTER DO KOMPONENTY NDCPRICER ... 71

13. TESTOVÁNÍ 77
13.1. Jednotkové testy 77
13.2. Integrační testy 78
13.3. Uživatelské testování 78
13.4. Shrnutí 81

14. ZÁVĚR 82

15. LITERATURA 84

16. SEZNAM OBRÁZKŮ 88

17. PŘÍLOHY 90

1. Úvod

Diplomová práce se zabývá návrhem, vývojem a zapojením systému, který je určený pro společnost AARON GROUP[1], která se již od roku 1998 zabývá primárně vývojem rezervačních systémů pro letecké společnosti, cestovní kanceláře a cestovní agentury. Systémy společnosti AARON GROUP komunikují s mnoha poskytovateli dat, a tato komunikace je složitá nejen na provedení, ale také na konfiguraci. Diplomová práce si klade za cíl tuto komunikaci usnadnit díky řešení, jež bude řídit počty transakcí na získání dat od poskytovatele, které není schopen zaslat v jedné odpovědi.

1.1. Představení systému na distribuci leteckých nabídek

Do 70. let 20. století letecké společnosti nabízely své letenky individuálně v kamenných prodejnách, nebo na vlastních pobočkách na letištích. Jednalo se o přímý kontakt letecké společnosti s koncovým zákazníkem. Data o letech byla ukládána do jednoduchých databází, a letenky byly prodávány na základě letového řádu, kapacit sedadel a jednotných ceníků. S přibývajícím počtem cestujících přestával být tento systém udržitelný a letecké společnosti musely přijít s novým řešením. Díky technologickému pokroku se v 70. letech začaly vytvářet centrální rezervační systémy, zkráceně CRS. Díky těmto systémům se na letecké společnosti mohly napojit cestovní kanceláře, a tím rozšířit místa, kde si zákazníci mohli zakoupit jejich letenky. Tyto systémy se v průběhu let dále vyvíjely a zdokonalovaly.

Letecké společnosti v 80. letech díky nárůstu zájmu o leteckou dopravu přestávaly zvládat udržovat a spravovat své rezervační systémy. Tento problém jim pomohly vyřešit firmy, které začaly rezervační systémy sdružovat. Začaly tak vznikat globální distribuční systémy, zkráceně GDS[2], které dodnes patří mezi nejvýznamnější systémy v leteckém průmyslu. Tyto systémy nesdružují pouze letecké společnosti, ale i ostatní poskytovatele služeb cestovního ruchu, jako například hoteliéry, pronajímatele aut a jiné. Globální distribuční systém nenabízí pouze nabídky, ale zajišťuje také servisní služby pro své zákazníky, které se týkají jak zakoupených letů a navazujících leteckých produktů, tak i servisních služeb v cílových destinacích.

I přesto, že globální distribuční systém patří mezi jeden z nejrozšířenějších způsobů distribuce nabídek cestovního ruchu, má i své nevýhody. Mezi tyto nevýhody patří především princip financování, který je založen na platbě od poskytovatele služeb, který musí jednotlivým firmám GDS platit nejen za uvedení nabídek, ale také za jednotlivé transakce

spojené s jejich prodejem, popřípadě úpravou. Náklady za transakce v systémech GDS stojí velké letecké společnosti mnoho peněz, proto se také snaží své služby nabízet i jiným způsobem.

V dnešní době se dopravci svým zákazníkům snaží zobrazovat ještě více informací o letech, vytvářejí proto rozhraní API (application programming interface), přes které se cestovní kanceláře a agentury mohou přímo napojit na letecké společnosti. Díky tomuto spojení mohou letecké společnosti na přímo komunikovat s koncovým zákazníkem, a tím od něj mohou získat více informací, než kdyby komunikovaly prostřednictvím GDS. Tuto práci jim v roce 2015 ulehčila asociace IATA (The International Air Transport Association) [3], která přišla se standardem New Distribution Capability (dále jen NDC)[4], což je řada standardů pro distribuci leteckých dat založených na XML. Asociace IATA tyto standardy stále vyvíjí, a několikrát do roka vydává nové verze. Díky NDC standardům se zlepšuje schopnost komunikace mezi leteckými společnostmi a cestovními kancelářemi. Tato komunikace je také otevřena k implementaci a použití jakékoli třetí straně. NDC standardy umožňují posílat více dat než pomocí GDS, leteckým společnostem to dává možnost nabízet různé druhy nabídek, ať už jde o speciální letenky nebo o různé druhy doplňkových služeb. Mohou také získat více informací o svých zákaznících, díky čemuž mohou upravovat svůj obsah na základě toho, co koncový zákazník opravdu požaduje.

Druhý konec odvětví letecké dopravy zaujímají nízkonákladoví dopravci, zkráceně LCC (Low-cost carriers)[5], kteří používají odlišný model než klasické letecké společnosti, které cestujícímu poskytují spoustu dalších služeb. Nízkonákladoví dopravci se snaží cenu letenky co nejvíce snížit, proto místo zpoplatněného systému GDS dávají přednost vytvoření vlastního systému pro prodej letenek. Díky nízké ceně tak dnes nízkonákladoví dopravci tvoří zhruba třetinu všech leteckých společností po celém světě. Při vzniku prvních LCC se však nepředpokládalo, že bude možné nabízet jejich nabídky prostřednictvím cestovních kanceláří nebo jiných online vyhledávačů, protože letecké společnosti nenabízejí API, ke kterému by se cestovní agentury mohly připojit. Několika firmám se však úspěšně povedlo zobrazovat a nabízet letecký obsah i nízkonákladových dopravců. Mezi tyto firmy patří Momondo.com, Kiwi.com nebo Travelfusion.com.

1.2. Představení společnosti AARON GROUP a uvedení do problematiky

Společnost AARON GROUP vyvíjí několik systémů. Mezi hlavní patří rezervační systém Symphony, který slouží k prodeji služeb souvisejících s cestovním ruchem. Jedná se především o prodej letenek od tradičních i nízkonákladových dopravců, ubytování, cestovního pojištění nebo rezervaci parkovacích míst. Systém je dále členěn podle toho, kdo daný produkt využívá, jelikož nabízí řešení pro různé druhy zákazníků.

Symphony.Travel je určen pro cestovní agentury, které díky němu mohou nabízet NDC, GDS i LCC obsah spolu s doplňky. Mezi zákazníky patří například společnost Asiana, provozující stránky Letuška.cz, cestovní kanceláře Invie, Čedok nebo Air Pro a mnoho dalších.

Řešení Symphony.Corporate je určené pro společnosti odpovídající za správu služebních cest pro určitou osobu, společnost či organizaci, které se označují TMCs. Korporátní zákazníci si po přihlášení do systému mohou jednoduše zarezervovat privátní i pracovní cestu s letenkou, ubytováním a dalšími službami. Tyto rezervace mohou být dále kontrolovány a upravovány na základě pravidel společnosti. Za určitých podmínek mohou korporátní klienti vidět také korporátní tarify, což jsou zvýhodněné nabídky od poskytovatelů služeb, případně si zde lze nastavit korporátní bonus programy, nebo korporátní slevy.

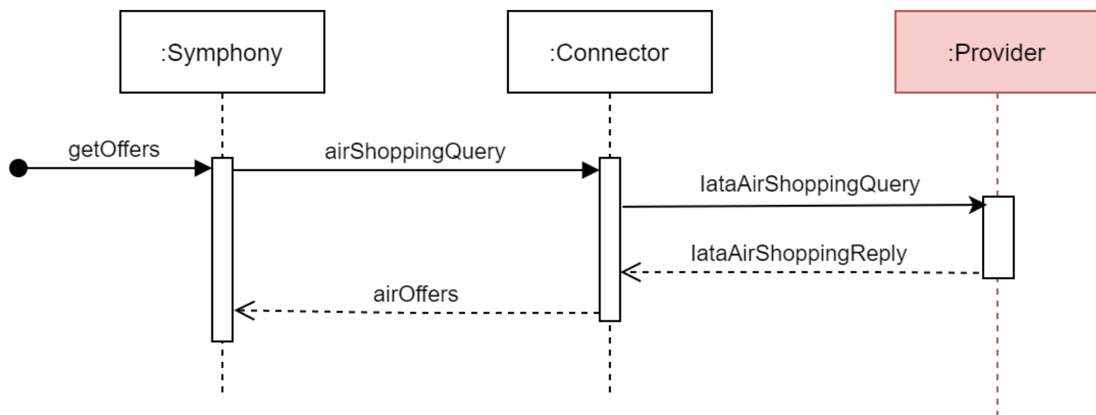
Symphony.Hub slouží vyškoleným agentům cestovní kanceláře, kteří díky němu mohou koncovým zákazníkům zarezervovat letenku a další navazující služby.

Poslední nabízeným řešením je Symphony.Aero, které je určeno pro letecké společnosti. Systém umožnuje vytváření leteckých nabídek, které následně může zobrazit ve webové aplikaci anebo je může poskytnout systému třetí strany. AARON GROUP může vytvořit pro leteckou společnost webovou stránku, skrze níž může nabízet své letenky.

Systém Symphony je tedy schopen zprostředkovat komunikaci mezi zákazníkem a poskytovateli služeb cestovního ruchu, prostřednictvím GDS, NDC i LCC. Celý systém se skládá z mnoha komponent. Jednotlivé komponenty spolu komunikují a mají svůj konkrétní účel. Komponenty, které nás v této práci primárně zajímají se starají o úpravu dat a o komunikaci s jednotlivými leteckými společnostmi, nebo jinými poskytovateli dat, kteří pro komunikaci využívají standardy NDC. A to především z toho důvodu, že se společnost AARON GROUP v současné době zaměřila především na vývoj NDC systému. Momentálně zajišťuje spojení s více než třiceti leteckými společnostmi, které se systémem Symphony komunikují pomocí NDC na přímo, nebo přes jiného IT zprostředkovatele. Tuto komunikaci

nazýváme komunikace s poskytovatelem dat, kterou zajištují komponenty, kterým se ve firmě AARON GROUP říká konektory a jejich úkolem je komunikovat s daným poskytovatel dat a převádět data z jednoho formátu na druhý.

Pro každého poskytovatele dat, se kterým systém Symphony komunikuje, byl vytvořen vždy jeden konektor, a to z několika důvodů. Prvním z nich je, že většina poskytovatelů dat komunikuje pomocí jiné verze IATA NDC standardů, posílají také odlišná data a každý poskytovatel dat navíc většinou používá odlišnou techniku komunikace. Nicméně i přes odlišné verze IATA NDC standardů, fungují všechny konektory velmi podobně. Konektor zjednodušeně funguje následovně. Od hlavního systému dostane požadavek, z něj vytvoří objekt odpovídající dané verzi IATA NDC standardu, který se poté převede do formátu XML[1] a odešle poskytovateli dat. Následnou odpověď od poskytovatele dat převede na objekt, se kterým umí systém Symphony pracovat, a odešle jej vyšší vrstvě.



Obrázek 1.1: Sekvenční diagram komunikace s poskytovatelem dat

Při komunikaci mezi systémem Symphony a poskytovatelem dat se první dotaz nazývá „AirShopping“. V tomto dotazu poskytovatele dat žádáme o nabídky týkající se naší hledané cesty. Do tohoto dotazu můžeme nastavit různá data, kterými upřesníme vyhledávání. Jedná se například o preference na typ kabiny (Ekonomická, Premium, Business, First), preference na přímé lety, preference na čas odletu nebo příletu, nebo pokud se jedná o korporátního zákazníka, můžeme přidat id a kód jeho tarifu. Ovšem ne všechny tyto parametry můžeme nastavit současně, respektive každý poskytovatel dat podporuje jen určité kombinace těchto dat. Například pokud bychom chtěli zobrazit ekonomické a business nabídky od AirFrance, museli bychom tento dotaz rozdělit na dva a výsledky bychom pak sloučili.

[1] XML - eXtensible Markup Language je jazyk mimo jiné určený pro výměnu dat mezi aplikacemi, popřípadě systémy

V momentální implementaci je logika, která rozhoduje o tom, zda se dotaz musí rozdělit na více dotazů, umístěna v každém konektoru zvlášť a jednotlivé dotazy jsou volány postupně. Tato diplomová práce si klade za cíl zmenšit sémantickou mezeru mezi přirozeným požadavkem uživatele a technickými možnostmi leteckých společností (obecně poskytovatelů nabídek) pomocí komponenty, která bude rozhodovat o rozpadech jednotlivých dotazů.

2. Popis problému a možnosti řešení

Cílem diplomové práce je navrhnout řešení, jež bude řídit počty transakcí, na získání dat od poskytovatele, které není poskytovatel schopen zaslat v jedné odpovědi. Řešení musí být navrhnuto tak, aby bylo jednoduše konfigurovatelné a aby jej dokázal nastavit uživatel i bez znalosti jakéhokoliv programovacího jazyka. Důležité je, aby bylo možné konfiguraci kdykoliv změnit, bez nutnosti zásahu do kódu. Protože se požadavky na správnou konfiguraci mohou často měnit, měly by se změny dát provést co nejrychleji. Kdyby byla rozhodovací logika umístěna přímo do kódu, musel by se pro každou změnu měnit zdrojový kód, a následně by se musela vydat nová verze celé aplikace. Dále pak musí být řešení bezpečné a musí zabránit uživateli vytvořit takovou konfiguraci, která by zapříčinila pád celého systému. Také by mělo být navrhnuto tak, aby jej bylo možné jednoduše upravit a využít i v jiném systému.

Většina systémů, které byly vyvinuty ve společnosti AARON GROUP, jsou napsány v jazyce Java[6]. Jedná se o objektově orientovaný programovací jazyk, který patří mezi nejpoužívanější programovací jazyky na světě. I systém Symhony, do kterého bude implementovaná logika zapojena, je napsán v jazyce Java. Je nutné, aby se nově vyvinutý systém dal spustit na platformě, která se ve společnosti AARON GROUP již používá. Dále je potřeba předpokládat, že nově vyvinutý systém bude potřeba v budoucnu upravovat, proto by měl být napsán v jazyce, kterému rozumí co nejvíce vývojářů. Z těchto důvodu bylo uvažováno a hledáno primárně řešení, které je založené na jazyce Java.

Níže jsou představeny jednotlivé přístupy, kterými se dá problém řízení rozpadu jednotlivých dotazů řešit.

2.1. Přímá implementace

Jedním z nejrychlejších a nejjednodušších způsobů, jak k tomuto problému přistoupit, je implementovat logiku, která rozhoduje o rozpadu dotazu, do všech konektorů zvlášť. V každém konektoru by pak byla dost podobná logika, která by podle nakonfigurovaných vlastností pro daný konektor rozhodovala o rozpadech dotazů. Aby se tato logika dala nakonfigurovat, musel by každý konektor komunikovat s nějakým perzistentním úložištěm, z kterého by načítal konfiguraci. Pro vytvoření konkrétní konfigurace pro jednotlivé konektory by se musela vytvořit administrátorská aplikace, která by umožňovala uživateli vytvářet a upravovat jednotlivé konfigurace, které by se následně ukládaly do databáze. Dále, aby bylo možné rozpadnuté dotazy paralelně posílat poskytovateli dat, museli bychom do konektoru přidat logiku, která by se nám starala o práci s vlákny.

Tento přístup by nebyl příliš optimální z několika důvodů. Zaprvé bychom měli v několika komponentách velmi podobný kód, což odporuje zásadám psaní „čistého kódu“. Podle těchto zásad bychom se měli vyvarovat duplikaci kódu. Další nevýhodou je, že přidáváme další logiku do konektoru, který se snažíme mít co nejjednodušší. Dále také do konektoru přidáváme logiku, která tam nepatří, jelikož smysl konektoru je pouze převádět data A na data B, proto by zde neměla být žádná komunikace s databází, nebo logika starající se o práci s vlákny.

2.2. Rozhodovací komponenta

Další možností je logiku, starající se o rozhodování, zda se má daný dotaz rozpadnout, volat před tím, než dotaz pošleme do konektoru, ve kterém je upraven a poslán poskytovateli dat. Díky tomu nemusíme duplikovat kód. Pro tyto účely by se vytvořila nová komponenta, která by na vstupu dostala objekt, jenž by obsahoval informace o tom, s jakými parametry a jakým poskytovatelům dat chceme dotaz poslat. Komponenta by následně vyhodnotila, pro kterého poskytovatele dat a jakým způsobem by se měl požadavek rozpadnout. Následně by se dotaz rozpadnul podle vráceného výsledku z komponenty a paralelně by se provedlo volání konektorů. Z výsledných nabídek se následně odstraní duplikace, spojí do jednoho výsledku a zobrazí se uživateli.

Výše uvedená komponenta se dá vytvořit mnoha způsoby, které jsou níže detailněji popsány, ale všechny fungují na podobném principu. Aby komponenta odpovídala zadání, musí se rozhodovat podle nastavené konfigurace. Ta bude ve všech případech uložena v databázi a díky tomu se s ní dá lehce pracovat. Pro správu konfigurací by byla vytvořena speciální aplikace, ve které by se jednotlivé konfigurace daly vytvářet a upravovat. Všechna řešení také pracují se stejnými objekty jak na vstupu, tak i na výstupu z komponenty. Hlavní rozdíl mezi jednotlivými řešeními je v přístupu k rozhodovací logice a ve způsobu práce s konfigurací.

2.2.1. Naivní implementace

Prvním z těchto řešení je naimplementovat rozhodovací logiku bez použití již existujících knihoven či jiných řešení. V tomto konkrétním případě by to znamenalo, že by konfigurace pro každého poskytovatele dat byla uložena v konkrétní tabulce v databázi, kde by v jednotlivých sloupcích byly nadefinovány možnosti rozpadu dotazu, které daný poskytovatel dat podporuje. Implementace by mohla vypadat následovně. Po přijetí požadavku komponenta v databázi, popřípadě v cache, vyhledá konfigurace pro konkrétní

poskytovatele dat. Následně podle dané konfigurace postupně rozhodne, pro jaké poskytovatele dat a jakým způsobem se má daný požadavek rozpadnout.

Toto řešení je velmi snadno implementovatelné, má ovšem řadu nevýhod – např. je špatně škálovatelné. Pokud budeme chtít danou komponentu využít i v jiné části systému, budeme ji muset přepracovat. Ze začátku se počítá s tím, že systém bude operovat pouze s malým množstvím dat. To se však změní s připojením nových konektorů, popřípadě po zapojení komponenty i do jiné části systému. Pokud komponenta přestane zvládat množství dat, bude se muset celá přeprogramovat, nebo opětovně vytvořit.

2.2.2. Využití interpretovaných jazyků

Další možností je využití interpretovaných jazyků[7] pro oddělení rozhodovací logiky od zbytku kódu. Rozhodovací logika by společně s konfigurací pro konkrétního poskytovatele dat byla umístěna v souboru, jenž by byl uložen v databázi. Díky tomu nemusíme při změně business požadavků upravovat zdrojový kód a vydávat novou verzi celé aplikace, ale můžeme upravit jen jeden soubor, který je uložený v databázi. Poté, co by komponenta přijala požadavek, by z databáze načetla soubor s rozhodovací logikou pro konkrétního poskytovatele dat, a následně by jej spustila. Rozhodovací logika by vyhodnotila, jak se má daný požadavek rozpadnout, a po doběhnutí všech souborů, by komponenta poslala výsledek vyšší vrstvě. Pro implementaci konfiguračních souborů, můžeme využít mnoho skriptovacích jazyků jako například JEXL, KAWA, Jython a další.

Díky tomu, že jsou konfigurační soubory psány v programovacím jazyce, můžeme využívat syntaktickou kontrolu kódu, pro testování jednotlivých souborů. Tyto soubory však může psát jen uživatel, který danému programovacímu jazyku rozumí, což značně zmenšuje počet lidí, kteří by daný skript byli schopni napsat, popřípadě upravit. Bylo by tedy potřeba vytvořit program, jenž by podle určitého vstupu vygeneroval daný skript s rozhodovací logikou.

2.2.3. Rule-base systém

Komponentu, jež bude rozhodovat o rozpadu dotazů, je možné naimplementovat jako rule-base systém neboli systém založený na pravidlech, který umožňuje oddělit business logiku od implementace. Rule-base systém[8] se typicky skládá ze sady faktů nebo zdroje dat, sady business pravidel (business rules) a řídící jednotky, která sadu pravidel s daty spouští a řídí. V našem případě by konfigurace pro každého poskytovatele dat byla umístěna v několika pravidlech, která by následně byla uložena v databázi. Při spuštění komponenty se

spustí i řídící jednotka, do které se načtou všechna pravidla. Při příchodu požadavku do komponenty se následně požadavek předá řídící jednotce, která na něj aplikuje svá pravidla, a tím získáme požadovaný výsledek. Existuje několik přístupů, jak business pravidla vytvářet a jak s nimi pracovat, každý má své výhody i nevýhody. Na rozdíl od skriptovacích jazyků není v business pravidlech určených pro rule-base systém umístěna žádná složitá logika, takže je možné vytvořit administrátorskou aplikaci, která bude moci podle zadání, business pravidla pro určitý rule-base systém vytvářet. Existují i business pravidla, pro které již existuje aplikace pro vytváření, popřípadě správu jednotlivých pravidel.

2.3. Shrnutí

Pro vyřešení problému řízení počtu transakcí na získání dat od poskytovatele, které není schopen zaslat v jedné odpovědi, byly představeny čtyři způsoby. Přístup přímé implementace do každého konektoru zvlášť je snadný na implementaci, ale do budoucna není udržitelný. Předpokládá se, že konektorů bude přibývat, takže je kladen důraz na to, aby v jednotlivých konektorech bylo co nejméně logiky, kterou je nutné udržovat. Z toho důvodu je zapotřebí vytvořit rozhodovací komponentu, která se bude volat předtím, než se dotaz odešle do jednotlivých konektorů. Komponentu lze naimplementovat několika způsoby. Je možné v ní vytvořit vlastní rozhodovací logiku bez využití již existujícího řešení. Ta bude konfiguraci pro konkrétního poskytovatele dat načítat z databáze, podle které bude rozhodovat, jakým způsobem se má dotaz rozpadnout. Pro malé množství dat se jedná o elegantní řešení, ale s přibývajícími daty se kvůli různým úpravám komponenta zkomplikuje, a po čase z ní vznikne aplikace, která svým chováním a velikostí bude podobná již existujícímu systému, jenž používá business pravidla. Dále lze využít i existující systémy pro správu pravidel, které již vyřešily některé problémy spojené s implementací podobné komponenty. Nebo lze využít interpretované jazyky. Jejich největší nevýhodou však je, že skripty musí vytvářet pouze uživatel, který daný skriptovací jazyk zná. Proto je pro implementaci komponenty nejvhodnější využít business pravidla, a vytvořit rule-base systém. Při zvolení vhodného systému pro správu pravidel je možné splnit všechny zadané podmínky.

3. Business pravidla

V dnešní rychle se měnící době, se mohou firemní požadavky na systém měnit ze dne na den, a je potřeba na ně rychle reagovat. U velkých a složitých systému je ovšem problém je po změně požadavků rychle přepracovat. Tento problém lze z části vyřešit oddělením rozhodovací logiky od implementace softwarového řešení. Díky tomuto řešení nemusí vývojáři často procházet fázemi návrhu a implementace aplikací, jelikož mají možnost provádět změny v obchodní logice rychle a jednoduše. Pro oddělení rozhodovací logiky od zbytku aplikace je možné využít business pravidla (business rules)[9]. Nejedná se o konkrétní technologii, spíše jde o koncept, který je podporován řadou specifikací a softwarových řešení. Je tedy důležité zvolit správnou techniku a nástroj pro vyvíjenou aplikaci.

Business pravidlo definuje nebo omezuje některý aspekt podnikání a vždy je buď pravdivé, nebo nepravdivé.[10] Pravidla popisují operace, definice a omezení, která platí pro organizaci, a mohou se vztahovat na lidi, procesy, podnikové chování nebo na výpočetní systémy ve společnosti. Příkladem business pravidla je věta „Občan mladší osmnácti let nemůže vlastnit řidičské oprávnění na osobní automobil“.

Jednotlivá pravidla jsou tvořena z faktů, které jsou tvořeny z termínů. Termínem je myšlena konkrétní entita reálného života, jako je věc nebo činnost. Fakta se následně skládají z termínů a jejich vzájemných vztahů. V matematické logice se dají pravidla vyjádřit pomocí predikátového kalkulu ve tvaru „antecedent → konsekvent“, díky tomu se dá pravidlo vyjádřit jako logická formule. Predikáty jsou z pohledu business pravidel označovány jako fakta. Jednotlivá pravidla mohou obsahovat kvantifikátory (existenční i univerzální) a logické spojky (negace, konjunkce, disjunkce).

Ve většině standardů jsou pravidla zaznamenávána v IF-THEN (pokud-tak) tvaru, kde se antecedent daného pravidla označuje jako podmínka a konsekvent je jeho tělem. V těle pravidla se nemusí nacházet pouze logická formule, může se zde nacházet i programový kód, který má být vykonán. Business pravidla lze také rozdělit na deklarativní, která definují další fakta, na represivní, která zajišťují konzistenci odvozovací báze, pravidla akční a reakční, která vyvolávají konkrétní aktivitu, a na pravidla odvozovaná.

3.1. Využití business pravidel

Pro vyvinutí rule-base systému, jenž využívá business pravidla, je nejprve nutno zvolit vhodný zápis pravidel, a také softwarovou podporu pro správu pravidel a jejich spouštění. Je možné implementovat vlastní systém na spouštění pravidel, nebo je možné využít již

existující řešení, kterému se říká rule engine. Není to však vhodné řešení, jelikož existuje spousta jiných kvalitních řešení. Systémů pro správu pravidel, jenž se označují jako engine, je mnoho, a vybrat tedy správné řešení vhodné pro náš systém je obtížné.

Na výběr jsou jak obecné standardy, tak často užívané proprietární specifikace jednotlivých odvozovacích engine.

3.2. Business rule engine

Business rule engine[2] je komponenta, která zajišťuje spouštění pravidel. Na vstupu dostane sadu pravidel a sadu objektů, na jejichž základě probíhá odvozování, při kterém se podle typu systému využívá dopředné či zpětné řetězení. Častěji se využívá dopředné řetězení, které bývá založeno na algoritmu Rete[11] nebo na jeho modifikacích. Jedná se o algoritmus pro určení pravidel, které se mají spustit na základě jeho vstupních dat. Některé systémy dokonce podporují i hybridní odvozování, jedná se například o systémy Drools[12] a Jena reference[13]. Rule engine může být naimplementován jako komponenta, která se zapojí přímo do vyvíjené aplikace, nebo může být naimplementován jako samostatný systém, který je dostupný pomocí webových služeb nebo pomocí API.

3.3. Business Rule Management System

Pro správný chod systému je potřeba mít možnost udržovat a spravovat všechna používaná pravidla. Další důležitou částí systému je tak komponenta Business Rule Management System (BRMS), která se používá pro správu business pravidel. Pomocí BRMS může uživatel jednotlivá pravidla vytvářet, upravovat a může určovat, která pravidla se mají spouštět.

3.4. JSR 94

Některé implementace Rule engine poskytují implementaci API podle definice JSR 94 (Java Specification Requests)[14], díky jemuž lze získat nezávislost na libovolném rule engine, který si vybereme. Díky JSR 94 můžeme v budoucnu vyměnit rule engine za jiný, který poskytuje standard JSR 94, aniž bychom museli výrazně měnit pravidla či způsob interakce s rule engine. To ovšem neznamená, že nová pravidla budou vypadat stejně, nejspíš je budeme muset přepsat, ale nebudeme muset přepisovat části naší aplikace, abychom mohli používat nový rule engine.

---

[2] neexistuje vhodný překlad, proto bude nadále používán výraz rule engine nebo drools engine

3.5. Užívané specifikace business pravidel

V současné době existuje mnoho systému, které se dají využít pro práci s business pravidly. V této kapitole je představeno několik přístupů pro práci s business pravidly, které mají své výhody i nevýhody. Všechny je ovšem možno využít pro vyvinutí systému v jazyce Java.

3.5.1. OpenL Tablets

Systém OpenL Tablets[15] slouží pro správu business pravidel, a obsahuje také business rules engine založený na rozhodovacích tabulkách. Za předpokladu, že uživatelé umí pracovat s programem Excel[16], umožňují OpenL Tables překlenou propast mezi business manažerem a vývojářem. OpenL Tables je vhodný pro firmy, jež svoje obchodní pravidla mají sepsané v aplikacích Excel nebo Word[17], a jejich zaměstnanci s nimi jsou zvyklí pracovat. Pro implementaci OpenL Tables se musí pravidla upravit, aby je bylo možné vyhodnocovat, ale po nasazení a upravení, je možné tato pravidla spravovat i bez podpory technického pracovníka.

Systém se skládá z Business rules engine (jednotka pro vyhodnocování pravidel), WebStudio (webová aplikace pro správu pravidel), Rule Services (framework a service pro vyhodnocování pravidel, jako RESTful[3] nebo Kafka[4][18]) a Rules Repository (implementační koncept úložiště pravidel využívající Git[19], databáze nebo blob storage). Umožnuje spravovat pravidla jak v aplikaci Excel, tak i v aplikaci WebStudio ve webovém prohlížeči. Obchodní logika je primárně v rozhodovacích nebo vyhledávacích tabulkách, kvůli zjednodušenému způsobu aplikace všech obchodních algoritmů.

3.5.2. Easy Rules

Easy Rules[20] je jednoduchý Java rules engine, který je založený na POJO[5] framework pro definování pravidel. Je schopen z primitivních pravidel pomocí složeného vzoru vytvořit složitá pravidla. Na rozdíl od tradičních přístupů easy rules nepoužívá k oddělení pravidel od aplikace žádné doménově specifické jazyky jako například XML[6]. Místo nich používá

---

[3] RESTful API je architektonický styl pro aplikační programové rozhraní (API), které používá HTTP požadavky pro získávání a odesílání dat

[4] Apache Kafka je open source systém pro distribuované streamování událostí

[5] Plain Old Java Object – jedná se o Java objekt, který není vázán žádným zvláštním omezením, objekt POJO nemá žádné konvence pojmenování pro proměnné a metody

[6] XML - eXtensible Markup Language je jazyk určený pro výměnu dat mezi aplikacemi a pro publikování dokumentů

třídy a metody založené na anotacích pro vkládání business logiky do aplikace. Řešení je vhodné pro aplikaci, kterou budou spravovat pouze vývojáři, jelikož jsou pravidla implementována přímo v kódu aplikace.

3.5.3. Jess

Jess[21] je kombinací skriptovacího jazyka a rule engine, který umožňuje zápis pravidel jak ve formátu Jess rule language (rozšířené syntaxi založené na Lisp[7], tak také v podobě JessML (formát založený na XML). Jess umožnuje zapsat různé druhy pravidel, jako jsou IF-THEN pravidla, fakta, funkce, a také komplexní skripty. Ovšem pro uživatele, kteří neznají funkcionální jazyk Lisp, může jazyk Jess ze začátku připadat zvláštně. Výhodou je, že je velmi expresivní a dokáže vyjádřit složité logické vztahy s velmi malým množstvím kódu. Jess Rule Engine nabízí také API ve standardu JSR 94. Jess není open source řešení, je zdarma pro vzdělávací účely, ovšem pro použití Jess pro komerční systémy je nutná licence. Dále pak není k dispozici žádný externí systém pro správu samotných pravidel.

3.5.4. Open Rules

Systém Open Rules[22] se podobně jako OpenL Tablets zaměřil na zákazníky, kteří se nechtějí učit nový jazyk pro tvorbu business pravidel, ale umí pracovat s aplikací Excel. Open Rules umožnuje spravování business pravidel pomocí aplikací MS Excel, Google Sheets nebo OpenRules Graphical Explorer. Pomocí těchto aplikací jsou v tabulkové struktuře ukládány definice termínů, rozhodovacích struktur a také celé workflow. Open Rules poskytují dva modely rule engine. Jeden je sekvenční a druhý je inferenční. Přístup k jednotlivým modelům je možný pomocí jednoduchého Java OpenRules API nebo standardního rozhraní JSR-94. Výchozí modul sekvenčního rule engine provede všechna rozhodnutí a pravidla počínaje hlavním rozhodnutím, které může vyvolat další dílčí rozhodnutí a rozhodovací tabulky. Posloupnost provádění různých rozhodnutí a dílčích rozhodnutí je tedy specifikována uživatelem. OpenRules engine je možné spustit i pomocí aplikace Dockter[8].

---

[7] Lisp - jedná se o jeden z nejstarších funkcionálních programovacích jazyků
[8] Docker jedná se o open source projekt, jehož cílem je poskytnout jednotné rozhraní pro izolaci aplikací do kontejnerů.

3.5.5. Drools

Open source systém od společnosti Red Hat s názvem Drools[12], je systém pro správu business pravidel (BRMS), jehož business rules engine umožnuje dopředné i zpětné řetězení, využívající vylepšenou implementaci algoritmu Rete. Podporuje také standard Java Rules Engine API specifikace JSR94. Pravidla, s kterými rule engine pracuje, jsou ukládána v textové podobě do souboru s příponou .drl, kde těchto pravidel může být uloženo i více. Pravidla jsou v IF-THEN formě a každé z nich obsahuje identifikační název, podmínku a akci, která se má případně vykonat. Pravidla jsou svázána s jazykem Java. Jednotlivé entity, použité v pravidlech, musí být uvedeny na začátku souboru, a také musí být deklarovány v aplikaci, jež obsahuje rule engine. Jednotlivá pravidla mohou volat externí funkce, nebo mohou obsahovat i složitější kód. Drools engine je možné spustit ve dvou režimech: statefull(stavový) a stateless(beztavový). Při režimu statefull jsou v paměti uloženy všechny nahrané objekty, nad kterými se vyhodnocovala pravidla, a díky tomu mohou být na sebe navázány. V paměti jsou do té doby, dokud není zavolán příkaz *.dispose()*. Režim stateless drží v paměti objekt jen do té doby, dokud nad ním nejsou vyhodnocena všechna pravidla, poté je z paměti odstraněn. Jednotlivé objekty tak na sebe nemohou být navázány. Společnost Red Hat dále vytvořila aplikaci Business Central, která usnadňuje vývoj business pravidel pro koncové uživatele prostřednictvím uživatelského rozhraní, které má rozmanitou sadu funkcí a umožnuje nasazení těchto pravidel na server ve formě artefaktů. Data vytvořené v aplikace Business Cental se ukládají do lokálního Git repozitáře, na který je možné se připojit a data z něj získat.

3.6. Shrnutí

Aby vyvíjený systém správně fungoval a mohl být dobře spravovatelný a konfigurovatelný, je důležité zvolit správný standard, popřípadě systém pro práci s business pravidly. Nejdříve je nutné si ujasnit, kdo a jakým způsobem bude pravidla vytvářet a upravovat. Dále je nezbytné, aby byl nově implementovaný systém kompatibilní s jinými informačními technologiemi, které jsou v daném podniku využívány. Je také vhodné, aby nový systém pro správu a vyhodnocování business pravidel měl dobrou dokumentaci, aby vývojáři, kteří v budoucnu budou systém spravovat, měli dostatek informací o struktuře a použití systému.

Pro systém řízení rozpadu dotazů byl z výše uvedených zvolen systém Drools. Jelikož většina backendových systémů od společnosti AARON GROUP je napsána v jazyce Java, je vhodné využít řešení, které je implementováno ve stejném jazyce, jelikož zapojení a údržba bude mnohem snadnější. Výhodou systému Drools je jeho podrobná dokumentace a velká

základna uživatelů, kteří sdílí své poznatky při vytváření aplikací, jež využívají právě Drools rule engine. Další velkou výhodou je aplikace Business Central, která poskytuje uživatelské rozhraní pro tvorbu pravidel. Problematickou částí bude vytvoření aplikace, která pravidla z aplikace získá a uloží je do databáze, ale i přes to, je Drools rule engine vhodné využít.

4. Návrh implementace

Pro řešení problému rozhodování rozpadu dotazů posílaných poskytovateli dat, bude vytvořena komponenta, jež bude fungovat jako rule-base systém. Bude přijímat od hlavního systému požadavky, obsahující informace o daném dotazu, a na základě business pravidel rozhodne, jakým způsobem se má dotaz rozpadnout, následně pak vrátí výsledek hlavnímu systému. Bude vytvořena tak, aby ji bylo možné použít i v jiném systému na řešení podobného problému.

Hlavním prvkem celé komponenty budou business pravidla. Pro každého poskytovatele dat bude vytvořena speciální sada pravidel, jež bude definovat, jakým způsobem a za jakých okolností se mají dotazy rozpadat. Pravidla budou v IF-THEN formě. V IF části se bude porovnávat vstupní objekt neboli fakt s konfigurací a v THEN části se následně budou nastavovat parametry rozpadu. Pravidla budou uložena v databázi, odkud se budou komponenty při spouštění načítat.

Komponenta bude přijímat objekt obsahující informace o požadavku, který se má poskytovateli dat odeslat. Tento vstupní objekt bude vycházet z již existujícího objektu typu NdcAirShoppingQuery, podle kterého konektor vytvoří dotaz, který odesílá poskytovateli dat. Business pravidla podle vstupního objektu rozhodnou, jestli a jakým způsobem se má dotaz rozpadnout, a výsledek předá vyšší vrstvě. Ta podle něj upraví, popřípadě rozpadne, objekt typu NdcAirShoppingQuery na více objektů, které následně paralelně odešle ke zpracování daným konektorům. V nich následně dojde k převedení objektů typu NdcAirShoppingQuery na objekty odpovídajícím standardům Iata, které následně odešlou poskytovatelům dat.

Komponenta bude obsahovat Drools rule engine, ve kterém budou pravidla spouštěna a vyhodnocována. Jelikož s pravidly bude vždy interagovat pouze jeden objekt, respektive fakt, tak bude použita bez stavová implementace rule engine. Při vytvoření instance dané komponenty jí budou předány pravidla, které budou uloženy v databázi. Jakmile dostane daná pravidla, použije při sestavování rule engine. Objekt, jenž přijde na vstupu komponenty, se vloží do engine a následně na něj budou aplikována všechna pravidla. Po vyhodnocení všech pravidel bude objekt z paměti engine odstraněn.

Pro vytváření pravidel bude využita aplikace Business Central, která poskytuje uživatelsky příjemné rozhraní pro vytváření business pravidel. Aplikace pravidla ukládá do lokálního Git repozitáře. Pro jejich získání bude nutné vytvořit aplikaci, která bude detekovat vytvoření, popřípadě změnu pravidla. Následně dané pravidlo zpracuje a uloží jej do databáze.

5. *Systém Drools*

5.1. Historie

Během osmdesátých a devadesátých let, byly vynaloženy velké finanční prostředky na vytvoření umělé inteligence. Jednalo se například o americkou společnost Thinking Machines Corporation[23], která se od roku 1983 zabývala vývojem superpočítačů a umělou inteligencí, nebo o Japonské ministerstvo mezinárodního obchodu a průmyslu, které vyvíjelo počítač páté generace (Fifth Generation Computer Systems).

V roce 1994 vyhlásila společnost Thinking Machines Corporation bankrot –do té doby však dokázali vyrobit jedny z nejvýkonnějších superpočítačů té doby. Jednalo se o řadu masivně paralelních superpočítačů řady Connection Machine (CM), které ze začátku měly 64K (65 536) bitových sériových procesorů (16 procesorů na čip) a později v menších konfiguracích 16K a 4K. Tyto počítače byly naprogramovány pomocí různých programovacích jazyků jako třeba *Lisp, *C nebo CM Fortran. Zmíněné jazyky používaly proprietární kompilátory k překladu kódu do paralelní instrukční sady. Prvních několik počítačů Connection Machine bylo navrženo podle architektury SIMD[9] zatímco pozdější byly navrženy podle architektury MIMD[10].

Japonský vývoj počítačového systému páté generace[11] začal v roce 1982 s cílem vytvořit počítače využívající masivně paralelní výpočty a logické programování. Výsledkem měl být počítač s výkonem podobným superpočítači, který by poskytnul platformu pro budoucí vývoj umělé inteligence. Programovacím jazykem pro FGCS byl zvolen Prolog. FGCS předběhl svou dobu, a v roce 1992 skončil komerčním neúspěchem, velmi však přispěl k na poli logického programování.

Výše zmíněné projekty sice neuspěly, v rozvoji mnoha různých disciplín však představují důležitý krok – např. strojové učení, počítačové vidění, nebo reprezentace znalostí. Jejich odnože se pak dostaly do komerčních systémů, jako jsou expertní systémy BRMS[12].[24] Systémy správy obchodních pravidel (BRMS) jsou komplexní platformy

---

[9] Single Instruction, Multiple Data – jedná se o typ počítačové architektury, ve které existuje jedna řídicí jednotka, která odesílá stejnou instrukci různým procesorům, které pracují na různých datech
[10] Multiple Instruction, Multiple Data – jedná se o typ počítačové architektury, ve které má každý procesor svou vlastní řídicí jednotku, díky tomu může každý procesor provádět různé instrukce nad různými daty
[11] zkráceně FGCS
[12] BRMS - Business Rules Management System

pro správu rozhodování, které organizacím umožňují vytvářet, spravovat a implementovat škálovatelná obchodní pravidla v rámci celého podniku.

Na těchto základech se zrodil open source systém pro správu obchodních pravidel Drools, který v roce 2001 založil Bob McWhirter v a registroval jej u SourceForge. První verze systému Drools nebyla nikdy vydána, jelikož řešení bylo založené na metodě vyhledávání hrubou silou (Brute-force search), která byla příliš pomalá. Vydána byla až druhá verze, která byla založena na algoritmu Rete, v roce 2005 byla začleněna do JBoss Enterprise Application Platform. V roce 2006 společnost JBoss koupila společnost Red Hat. Od té doby se systém stále vyvíjel, a nyní se pracuje již na osmé verzi.

5.2. Popis projektu KIE

KIE[13] je zaštiťující projekt, který si klade za cíl sjednotit všechny související technologie od společnosti Red Hat, pod které spadá i Drools. Funguje také jako jádro sdílené mezi všemi projekty. Obsahuje následující projekty nabízející kompletní portfolio řešení pro automatizaci a řízení:

5.2.1. Drools

Drools je systém řízení obchodních pravidel s dopředným a zpětným zřetězením založených na odvození pravidel, který umožňuje rychlé a spolehlivé vyhodnocování obchodních pravidel a komplexní zpracování událostí. Modul pravidel je také základním stavebním kamenem pro vytvoření expertního systému, což je v umělé inteligenci počítačový systém, který napodobuje rozhodovací schopnost lidského experta.

5.2.2. jBPM

Systém jBPM[25] je určený pro modelování, spouštění a monitorování obchodních procesů a případů v průběhu jejich životního cyklu. Modelování obchodních cílů je dosaženo popisem kroků, které je k jejich dosažení třeba provést a pořadí těchto cílů je znázorněno pomocí vývojového diagramu. Tento proces výrazně zlepšuje viditelnost a agilitu dané obchodní logiky. jBPM se zaměřuje na spustitelné obchodní procesy, což jsou obchodní procesy, které obsahují dostatek podrobností, takže je lze skutečně spustit na BPM engine. Spustitelné obchodní procesy překlenují propast mezi podnikovými uživateli a vývojáři,

[13] KIE - Knowledge Is Everything

protože jsou na vyšší úrovni, a využívají koncepty specifické pro doménu, kterým obchodní uživatelé rozumí, ale lze je také přímo spustit.

5.2.3. *OptaPlanner*

OptaPlanner[26] je nástroj pro optimalizaci a řešení diskrétních optimalizačních problémů. Systém kombinuje optimalizační heuristiku a metaheuristiku s velmi efektivním výpočtem skóre. Je určený pro programátory k efektivnímu řešení optimalizačních problémů.

5.2.4. *Business Central*

Business Central, známá také jako Workbench, je plnohodnotná webová aplikace pro sestavování business pravidel a procesů. Je zde možné vytvářet business pravidla, procesy, a další aktiva, která lze následně spravovat a testovat. Business Central mohou používat vývojáři, obchodní analytici a specialisté na obchodní automatizaci k vytváření projektů, modelů, procesů, případů a pravidel.

5.3. *Rule engines a Production Rule Systems (PRS)*

Pojem „Rule engine“ může označovat jakýkoli systém, který používá business pravidla v libovolné formě, jež lze aplikovat na data k získání výsledků. Díky tomu může pojem zahrnovat širokou škálu systémů.[27]

Původní rule engine od Drools se dá označit jako specifický typ zvaný Production Rule System (PRS), kde PRS označuje počítačový program, který se obvykle používá k poskytování nějaké formy umělé inteligence, jež sestává především ze sady pravidel chování, ale zahrnuje také mechanismus nezbytný k dodržování těchto pravidel, když systém reaguje na stavy světa.

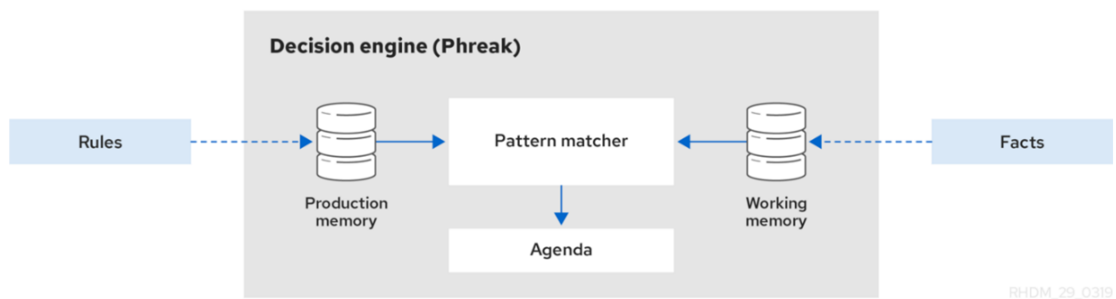
Pravidla jsou sestavena ze dvou částí, z podmínky, která se za běhu vyhodnocuje na základě aktuálních dat a faktů, a z akce, která je vykonána za předpokladu, že je podmínka splněna. Pravidlo má následující formu

```
when
  // Podmínka
then
  // Akce
```

V pravidlech se nepoužívá imperativní if – else forma, je zde použita deklarativní when – then forma. Rule engine vyhodnocuje pravidla nejefektivnějším způsobem, díky tomu

pravidla neurčují podrobný sled kroků, místo toho deklarují podmínky, za jakých se mají akce provádět.

V produkční paměti jsou uložena pravidla a fakta, která se vyhodnocují pomocí rule engine, jsou uchovávána v pracovní paměti. Fakta mohou být během vykonávání upravována. Proces mapování nových nebo existujících faktů, které vstupují do pravidla, se nazývá „shoda vzorů" (pattern matching) a je prováděna inferenčním enginem. Akce se provádějí v reakci na změny v datech, tento přístup se nazývá „datově řízené" (Data-driven). Samotné akce mohou změnit data, která se zase mohou shodovat s jinými pravidly a způsobit jejich spuštění, toto se označuje jako dopředné řetězení.



Obrázek 5.1: Přehled základních komponent Decision engine neboli Rule Engine

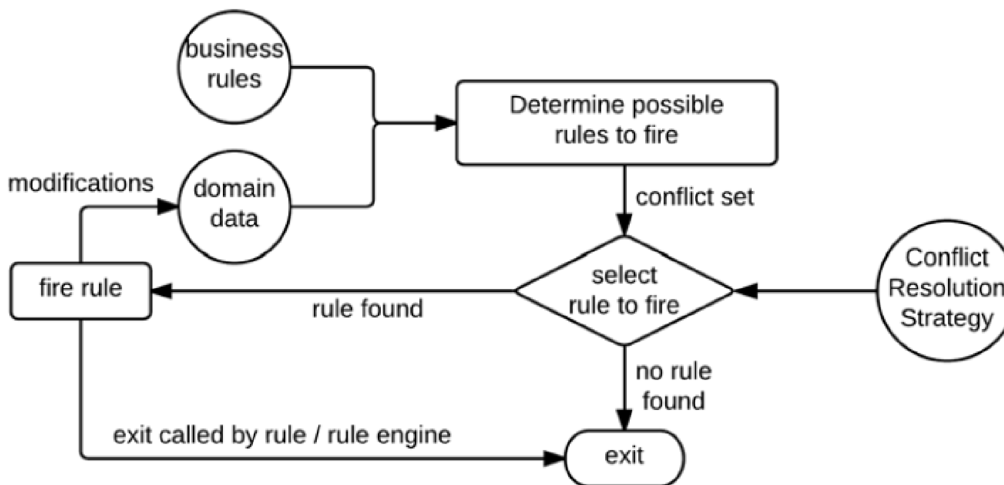
Další přístup se nazývá „zpětné řetězení", který je možné charakterizovat jako cílově řízené (gol driven). Rule engine je spuštěn se záměrem, kterému se snaží vyhovět. Pokud to nedokáže, snaží se dosáhnout dílčího cíle, který by pomohl splnit neznámou část aktuálního cíle. Tato akce může pokračovat rekurzivně a končí, až když se dosáhne počátečního cíle, nebo neexistují žádné další dílčí cíle. Například princip činnosti programovací jazyka Prolog[28] je založen na zpětném řetězení.

5.4. Životní cyklus provádění pravidel

Data a informace v objektově orientovaném prostředí jsou reprezentovány instancemi objektů. Když jsou tyto instance vloženy do pracovní paměti rule engine, stanou se v terminologii Drools fakty, a od nynějška mohou být hodnoceny podle deklarovaných podmínek pravidel.[29] Rule engine optimalizuje vyhodnocování podmínek, a zajišťuje co nejrychlejší určení pravidel pro vyhodnocení. Pokud není rule engine nastaven jiným způsobem, neprovádí automaticky akce v business pravidlu okamžitě po vyhodnocení podmínek. Jakmile engine zjistí, že podmínky v pravidle jsou pro určitou skupinu faktů

splněné, pravidlo a spouštěcí fakta se přidají do seznamu, kterému se nazývá Agenda. Pravidla v agendě nemusí být spuštěna, respektive vykonána, okamžitě. Pro jejich spuštění, je zapotřebí zavolat speciální příkaz. V tu chvíli jsou provedeny všechny akce, které pravidla obsahují.

Jakmile je zavolána metoda pro spuštění pravidel a agenda obsahuje více jak jednu akci, engine musí určit, která akce se má spustit jako první. Toto rozhodnutí je založeno na strategii řešení konfliktů. Výsledek strategie řešení konfliktů může být založen na mnoha faktorech, jako jsou atributy pravidla, aktuálnost pravidla, která deklaruje, kolikrát bylo pravidlo spuštěno, složitost pravidla nebo pořadí načítání pravidel. Jakmile je určeno pořadí akcí, provede se první z nich, což může mít za následek změnu některých faktů. Následně je potřeba přehodnotit pravidla vůči nově upraveným faktům v pracovní paměti a v důsledku toho je potřena nahrát nová pravidla do agendy, popřípadě odstranit stávající. Jakmile jsou provedeny všechny akce, které byly naplánovány, zkontroluje se obsah agendy. Pokud agenda stále obsahuje nějaké akce, postup pokračuje určením nových akcí ke spuštění. Tento cyklus pokračuje, dokud není agenda prázdná. Celý proces je znázorněn na následujícím obrázku *Obrázek 5.2*.



Obrázek 5.2: Přehled cyklu při provádění pravidel

5.5. Algoritmus Rete

Drools engine je rychlý a škálovatelný díky tomu, že je založen na algoritmu Rete[11] , který vynalezl Dr. Forgym a poprvé publikoval v roce 1974. Název Rete je převzat z latiny, kde slova rete znamená síť. Algoritmus byl vyvinut k efektivní aplikaci mnoha pravidel nebo vzorů na mnoho objektů nebo faktů ve znalostní bázi.

Naivní implementace by mohla postupně zkontrolovat každé pravidlo proti známým faktům ve znalostní bázi, v případě potřeby toto pravidlo vyhodnotit a poté přejít k dalšímu pravidlu. Tento naivní přístup začíná být příliš pomalý i pro středně velké znalostní báze pravidel a faktů. Algoritmus Rete tento problém řeší efektivnější metodou.

Systém založený na algoritmu Rete buduje síť uzlů. Každý uzel, vyjma kořene, odpovídá vzoru vyskytujícímu se na levé straně pravidla, respektive odpovídající podmínce pravidla. Tato síť se dá také označit jako „prefixový strom“. Cesta od kořenového uzlu k listu definuje kompletní podmínkovou část pravidla. Jak jsou postupně nová fakta uplatňována nebo upravována, šíří se dané změny po síti, což způsobuje následnou anotaci uzlů, když se fakt shoduje s daným vzorem. Když fakt nebo kombinace faktů způsobí, že jsou splněny všechny vzory pro dané pravidlo, je dosaženo listového uzlu a díky tomu se odpovídající pravidlo spustí.

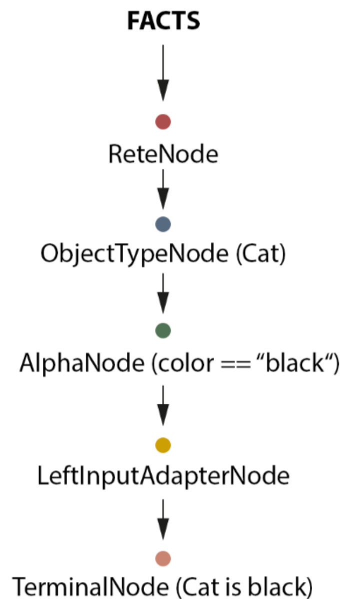
5.5.1. Síť Rete

Algoritmus transformuje podmínky pravidel do sítě Rete. Tato rozlišovací síť je reprezentována kořenovým, acyklickým a orientovaným grafem, a je modifikována vždy, když je ze znalostní báze změněno, přidáno nebo odstraněno pravidlo. Jeho účelem je efektivně filtrovat data procházející sítí.

Síť se skládá z různých typů uzlů. Všechny budou vysvětleny níže. Pro lepší pochopení předpokládejme, že znalostní bázi tvoří pouze následující pravidlo:

```
rule „Black cat“
  when
    Cat( color == “black” )
  then
    print( “Cat is black” )
end
```

Výše uvedené pravidlo má svůj důsledek pro objekt *Cat*, kde má atribut *color* roven *black*. Složení sítě Rete znázorňuje obrázek níže.



Obrázek 5.3: Síť rete pro pravidlo „Black cat"

Uzel Rete, reprezentuje každý kořen v síti Rete, který lze vnímat jako vstup do sítě, kterým musí projít všechny objekty uložené v paměti. Po něm následuje uzel Object Type, jehož účelem je zajistit, aby engine nedělal více práce, než je nezbytně nutné. Budeme-li mít například dva objekty různých typů, tak pokud by se engine pokusil vyhodnotit každý jednotlivý uzel proti každému objektu, promarnil by spoustu cyklů. Ale díky tomu, že engine ví, jakého jsou objekty typu, může každý přidělit konkrétnímu uzlu. V příkladu rete sítě výše můžeme vidět jeden uzel Object Type, pro objekty typu Cat.

Po uzlu Object Type může následovat jeden či více Alpha uzlů. Ten má jeden vstup a definuje vnitroprvkovou podmínku, což znamená, že vyhodnocují omezení jednotlivých faktů. Omezení mohou mít různé formy, jako jsou literály, proměnné, vložená vyhodnocení nebo návratové hodnoty. Více omezení na stejném typu je vyjádřeno více Alpha uzly, z nichž každý představuje jediné omezení. Pořadí omezení v pravidle je důležité, jelikož může ovlivnit dobu provádění, proto je důležité, aby více restriktivní omezení předcházelo méně restriktivnímu. Například pokud budeme mít v pravidlu následující dvě omezení: „Auto má čtyři kola" a „Auto je značky Ferrary", omezení, že auto je značky Ferrary, by mělo být první, jelikož skoro většina aut má čtyři kola, ale málo z nich je značky Ferrary.

Dalším typem uzlu je Left Input Adapter, který se používá k vytvoření n-tice z jednoho faktu, aby bylo možné jej spojit s jiným typ uzlu.

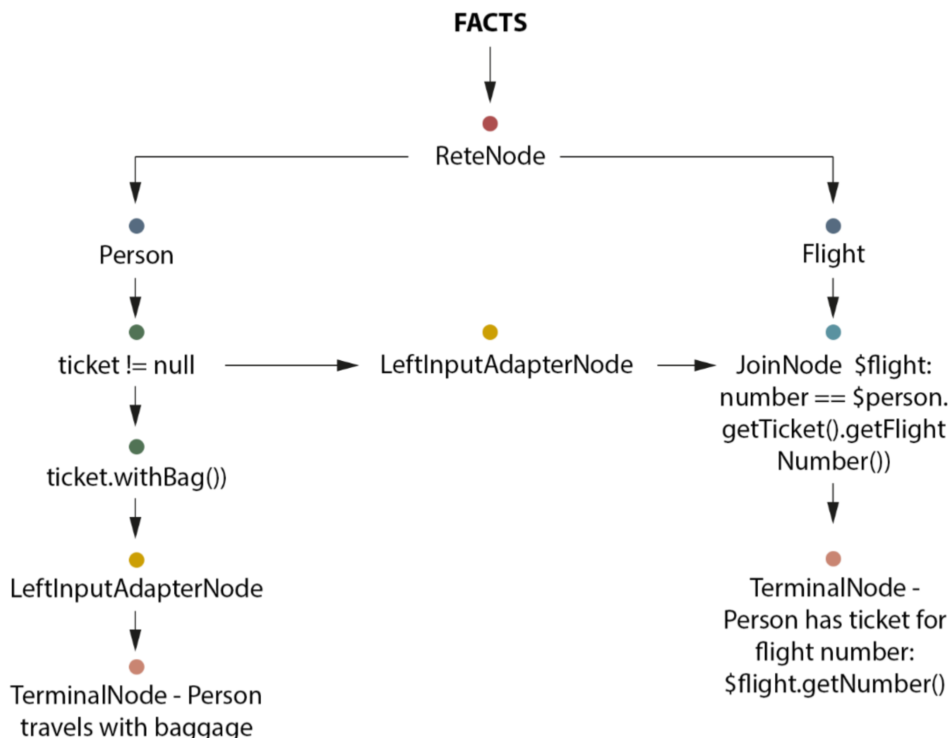
Uzly typu Terminal tvoří listy sítě rete, které reprezentují prováděcí část pravidla, a měly by být přiřazeny agendě pro budoucí provedení. Jedná se o cílový uzel, do kterého

může vyhodnocování dospět, pokud jsou předtím splněny všechny podmínky. Každému pravidlu musí odpovídat alespoň jeden Terminal uzel, ovšem pravidlo s podmíněným disjunktivním spojovacím prvkem “nebo“ má za následek generování podpravidel pro každou možnou logickou větev. Z toho plyne, že jedno pravidlo tedy může mít více koncových uzlů.

Nyní vysvětlíme příklad na obrázku uvedeném výše. Předpokládejme, že do sítě rete vstoupí výchozím uzlem Rete fakt. Následně bude pokračovat do uzlu Object Type, kde dojde k porovnání typu. Pokud bude fakt typu Cat, bude pokračovat do dalšího uzlu, a to do Alpha uzlu. V něm dojde k vyhodnocení omezení daného faktu. Pokud by parametr color v daném faktu nebyl nastaven na „black“, skutečnost by se dále nepropagovala, a metoda vkládání by byla ukončena. V opačném případě by fakt přešel na uzel Left Input Adapted, kde by se přeměnil na n-tici a poté by dosáhl Terminal uzlu, což by vedlo k uložení pravidla „Cat is black“ do agendy. Pokud by byla skutečnost vložena do jiného vstupního bodu nebo byla jiného typu, postup by skončil v příslušném uzlu. Pro lepší pochopení předpokládejme následující pravidla, která budou tvořit znalostní bázi.

První pravidlo očekává fakt typu Person, který bude obsahovat proměnou ticket, ve které bude definované zavazadlo. Pokud daná podmínka bude splněna, vypíše text ve tvaru "Person travels with baggage". Druhé pravidlo očekává dva fakty. První typu Person, který bude obsahovat proměnou ticket. Druhý fakt by měl být typu Flight s proměnou number. Pokud se proměnná number shoduje s proměnou flightNumber v proměnné ticket, tak se vypíše text ve tvaru "Person has ticket for flight number: " a nakonec i proměnná flightNumber.

```
rule „flight with baggage“
  when
    $person : Person( ticket != null && ticket.withBag())
  then
    print("Person travels with baggage");
end

rule „person has a ticket for flight“
  when
    $person : Person( ticket != null)
    Flight( $flight: number == $person.getTicket().getFlightNumber())
  then
    print("Person has ticket for flight number: " + $flight.getNumber());
end
```



Obrázek 5.4: Síť Rete pro pravidla „flight with baggage“ a „person has a ticket for flight“

Uzel Beta je dalším typem uzlů, který má dva vstupy, jeden pro n-tice a druhý pro fakta. Uzel umožňuje srovnání faktů nebo jejich atributů mezi sebou. To je možné díky tomu, že uzel má paměť, která je rozdělená na dvě části. V jedné uchovává informace o došlých faktech a ve druhé o došlých n-ticích. Později lze fakta vyhodnotit oproti faktům přicházejícím na jiný vstup a výsledek propagovat dále.

Beta uzlů existuje několik a mezi nejpoužívanější patří JoinNode, NotNode, ExistsNode, AccumulateNode a CollectNode. Na výše uvedeném obrázku je požit JoinNode, který je zodpovědný za spojení a vyhodnocení n-tice a faktu.

Ve výše uvedené síti Rete *Obrázek 5.4* jde vidět, že je zde pouze jeden sdílený Alpha uzel pro podmínku „ticket != null“. Stalo se tak díky vhodnému pořadí podmínek uvnitř pravidel. Kdyby v pravidle „flight with baggage“ byly podmínky prohozeny, již by nemohl být Alpha uzel sdílený, a musely by zde být dva. Tato skutečnost by zapříčinila rozvětvení sítě rete a vedla by ke zpomalení algoritmu. Aby byl algoritmus skutečně účinný, pravidla by měla sdílet co nejvíce uzlů, aby bylo možné minimalizovat hodnocení stejných podmínek. Obecně také platí, že podmínky a omezení v pravidlech by měly začínat těmi přísnějšími. Z těchto důvodů je důležité, aby se fáze navrhování pravidel neuspěchala.

5.6. Algoritmus Phreak

Drools engine používá pro vyhodnocení pravidel algoritmus Phreak[30], který se vyvinul z algoritmu Rete. Před algoritmem Phreak drools engine využíval algoritmus ReteOO pro objektově orientované systémy, který také vycházel z algoritmu Rete. Předností algoritmu Phreak je jeho škálovatelnost, která je vyšší než u algoritmů Rete a ReteOO. Další výhodou je jeho vyšší rychlost ve velkých systémech.

Dalším rozdílem mezi algoritmy Phreak a Rate je jejich přístup k vyhodnocování pravidel. Algoritmus Phreak je založen na odloženém vyhodnocování, zatímco algoritmus Rete je založen na okamžitém vyhodnocování pravidel. Algoritmus Phreak je cílově orientovaný (goal oriented) zatímco algoritmus Rete je datově orientovaný (data oriented). To se projevuje především ve vykonávání výsledné akce daného pravidla. Jak již bylo vysvětleno dříve, každé pravidlo obsahuje akce, které se mají provést, když je splněna podmíněná část pravidla. Akce se skládají z jedné nebo více metod, které se provádějí na základě podmínek pravidla a dostupných datových objektů. Hlavním účelem akcí pravidel je vložit, odstranit nebo upravit data v pracovní paměti Drools engine. Algoritmus Rete provádí vykonání akce pravidla co nejdříve, což může způsobit spuštění dalšího pravidla, proto se algoritmus Rete kontroluje a porovnává všechny pravidla. Dané porovnávání může, především u větších systémů, vést ke zbytečnému nárůstu složitosti. Algoritmus Phreak se tento problém snaží minimalizovat tím, že je částečné vyhodnocování pravidel záměrně zpožděno, aby bylo možné efektivněji zpracovávat velké množství dat.

5.6.1. Vyhodnocování pravidel

Párování/porovnávání vzorů je proces, ve kterém dochází k porovnání faktů, které vstupují do rule engine, s podmínkami pravidel, které jsou uloženy v produkční paměti. Nemůže dojít k žádnému vyhodnocení pravidel, pokud jsou pravidla odpojena.

Když uživatel nebo automatizovaný systém přidá nebo aktualizuje informace související s pravidly, informace se vloží do pracovní paměti Drools engine ve formě jednoho nebo více faktů. Drools engine porovnává tato fakta s podmínkami v pravidlech, která jsou uložena v produkční paměti, aby určil vhodná provedení pravidel. Tento proces přiřazování faktů k pravidlům je často označován jako párování vzorů. Když jsou splněny podmínky pravidla, Drools engine aktivuje a zaregistruje pravidla v agendě, kde pak Drools engine třídí prioritní nebo konfliktní pravidla v rámci přípravy na provedení.

Když se spustí Drools engine, žádné pravidlo není propojené s daty, respektive s fakty, které mohou pravidlo spustit. V první fázi algoritmus Phreak nespouští akce, které se mají

vykonat za předpokladu, že je podmínka pravidla splněna, ale místo toho jsou zařazeny do prioritní fronty. Algoritmus Phreak používá heuristiku, k nalezení pravidla, které má největší pravděpodobnost spuštění, respektive jehož podmínka bude nejpravděpodobněji splněna. Pokud se v pracovní paměti nachází všechna fakta, která dané pravidlo požaduje na svém vstupu, považuje se pravidlo za propojené s příslušnými daty a může dojít k porovnání vzorů. Následně algoritmus Phreak vytvoří cíl, který reprezentuje dané pravidlo, a umístí cíl do prioritní fronty, která je řazena podle významu pravidel. Vyhodnocují se pouze pravidla, pro které byl cíl vytvořen, a další potenciální vyhodnocení pravidel jsou zpožděna.

5.7. Drools Runtime

Drools umožňuje vytvářet instance Drools engine různými způsoby, takže je možné vybrat, který lépe odpovídá řešenému problému. Každá instance modulu pravidel je zapouzdřeným kontextem, kde budou definovaná pravidla vyhodnocena oproti datům, která této konkrétní instanci budou poskytnuta. Před několika lety byly Rule Engines považovány za velké a monolitické aplikace, běžící na serveru, kterým je možné odesílat data ke zpracování. To pro Drools již nemusí platit, díky tomu, že umožňuje lokálně vytvářet odlehčené instance vyvíjené aplikace. Díky Drools je možné jednu velkou instanci rozdělit na více menších instancí, které mohou zpracovávat různá pravidla a data.[31]

Aby bylo možné začít se sestavováním pravidel a budováním základu aplikace používající Drools engine, je nezbytné porozumět sestavování zdrojů. To je spojeno s následujícími pojmy, které se vyvinuly z předchozích verzí Drools. Jedná se o KieServices, KieContainer, KieModule, KieBase, KieSession. Pomocí těchto pěti pojmů, je možné nastavit jakoukoliv instanci Rule Engine, a určit, jaká pravidla budou pro každou z nich dostupná.

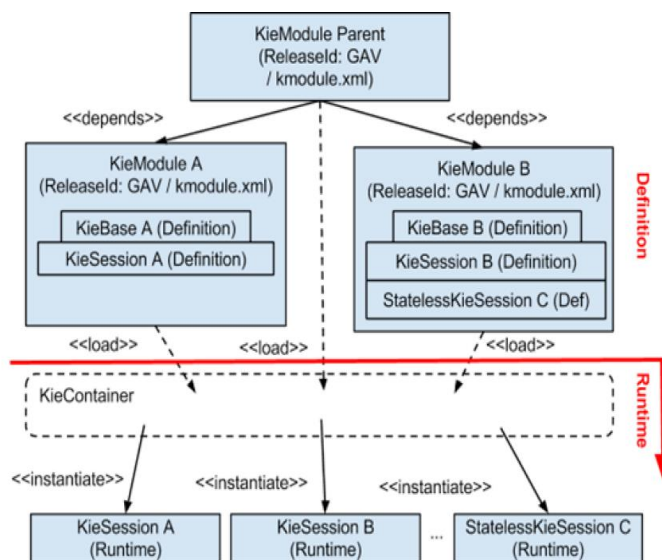
Nejdůležitější je třída KieServices, jelikož poskytuje přístup ke všem z výše uvedeným pojmům tím, že poskytuje registr služeb, kde je možné najít pomocné funkce pro různé účely. Instance KieServices, se získává pomocí statické metody *KieServices.Factory.get()*. Pomocí KieServices je možný přístup k řadě továrních metod (Factory method), službám (Servicies) a obslužných metod (utility method) používaných spolu s instancemi Rule Engine. Nové instance KieContainer, která se vytváří pomocí KieServices, definuje rozsah pravidel, jež budou použita k vytvoření nových instancí Rule Engine.

KieModule je kontejnerem všech prostředků nezbytných k definování sady KieBase. Projekt obsahující KieModule musí obsahovat i soubor *kmodule.xml*, ve kterém je konfigurace, pro seskupování a využívání konkrétních business aktiv, mezi které patří:

business pravidla, business procesy, rozhodovací tabulky atd. Díky tomu každý objekt KieModule obsahuje business aktiva související s určitou oblastí nebo doménou. Objekty KieModule mohou obsahovat další KieModule objekty, díky tomu je možné vytvořit objekt vyšší úrovně obsahující několik business aktiv z různých domén.

KieContainer může hostit KieModule a jeho závislosti, to znamená, že hierarchickou strukturu KieModule lze načíst do instance KieContainer.

Vztahy mezi těmito pojmy jsou znázorněny na následujícím diagramu:



Obrázek 5.5: Diagram reprezentující vztahy mezi pojmy potřebnými při implementaci aplikace s Drools engine

Z diagramu *Obrázek 5.5* je patrné, že KieContainer umožnuje vytvořit instanci KieModul za účelem vytvoření jedné nebo více instancí Rule Engine. Instanci KieModul lze nakonfigurovat různými způsoby. Všechny instance mohou obsahovat stejná pravidla anebo je mezi sebou mohou kombinovat. Dále je v diagramu naznačeno, že je možné se rozhodovat mezi jednotlivými KieModule instancemi. Za předpokladu, že momentálně jsou potřeba pouze pravidla definována v KieModule A, je možné načíst právě tuto instanci, anebo je také možné načíst instanci KieModule Parent, které závisí jak na instanci KieModule A, tak na instanci KieModule B, díky čemuž bude v těchto modulech načtená každá konfigurace.

V KieBase je uložištěm definic znalostí aplikace, které jsou definovány v souboru *kmodule.xml*. Samotná KieBase data neobsahuje, ale místo toho se z KieBase vytvářejí relace, do kterých lze vkládat data a ze kterých lze spouštět instance procesů. KieSession se vytváří z KieBase a představuje instanci rule engine obsahující pravidla v KieBase. Existují dva typy KieSession – stavová (Statefull) a bezstavová (Stateless). Stavová instance

KieSession umožňuje udržovat stav mezi několika interakcemi s Rule Engine. Naproti tomu bezstavová instance StatelessKieSession umožňuje interagovat pouze jednou, přičemž se pro další interakci neukládá žádný stav. Implementační rozdíl mezi stavovou a bezstavovou KieSession je především v tom, že do stavové instance KieSession je možné postupně vkládat několik faktů pomocí metody *insert()* a následně je možné zavolat funkci *fireAllRules()*, čímž dojde k provedení všech pravidel na daných faktech. Zatím co u bezstavové instance StatelessKieSession je k dispozici metoda *execute()*, díky níž je možné do rule engine vložit fakt, respektive kolekci faktů, a následně ihned dojde k provedení všech pravidel.

5.8. Struktura pravidel Drools

Business pravidla DRL[14], jsou definována v textových souborech s příponou *.drl* nebo *.rdrl*. Soubor DRL může obsahovat jedno nebo více pravidel, která definují minimálně podmínky pravidla (when) a akce (then). Soubory DRL se skládají z následujících součástí:

```
package
import
function
query
declare
global
rule "název pravidla"
    // Atributy
  when
    // Podmínky
  then
    // Akce
end

rule "další pravidla "
...
```

Struktura i syntaxe souboru DRL s pravidly je poměrně jednoduchá. Soubor by měl začínat názvem balíku a seznamem importovaných tříd, popřípadě metod. Pravidla dále musí mít unikátní název a měla by být umístěna na konci souboru. To jsou nejstriktnější pravidla, která jsou na soubor kladena. Soubor může obsahovat jedno nebo více pravidel, dotazů a funkcí. Může také definovat deklarace prostředků, jako jsou importy, globální hodnoty a atributy, které jsou přiřazeny a používány pravidly a dotazy. Všechny datové objekty související s pravidlem DRL souborem musí být umístěné ve stejném balíku, který je

[14] DRL - Drools Rule Language

deklarován na začátku souboru. Pokud ovšem pravidla v DRL souboru využívají i jiné datové objekty, které se nenacházejí v daném balíku, lze je naimportovat. Kompletní seznam i s vysvětlením všech klíčových slov, které je možné při tvorbě pravidla využít je možné nalézt v dokumentaci Drools v páté kapitole.[32]

5.8.1. Rule

Pravidlo se skládá z názvu, který musí být unikátní a musí být v uvozovkách, z volitelných atributů, z podmínkové části, ve které je definováno, za jakých okolností se pravidlo provede, a nakonec je definována akce, která je vykonána, pokud je splněna podmínková část. V pravidle může být několik atributů, které upravují chování pravidla. Atributů, které je možné vložit do pravidla, je celá řada. Například lze použít atribut salience, jehož celočíselný atribut definuje prioritu pravidla. Pomocí enabled lze zakázat nebo povolit pravidlo. Atribut date-effective s časem a datumem jako parametr upřesňuje, od kdy je, respektive bude, pravidlo aktivní.

5.8.2. Levá strana pravidla

LHS[15] se používá jako označení podmínkové části pravidla, která může být složena z více podmínek. Podmínky jsou reprezentovány vzory (pattern), které se mohou shodovat s fakty, jež byly vloženy do pracovní paměti. Jednotlivé vzory jsou spojovány podmíněnými prvky, a mezi nejběžnější z nich patří prvek „and“, který jediný nemusí být explicitně uveden. To znamená, že pokud pravidlo obsahuje více vzorů, které musí být všechny splněny, mohou být odděleny pouze čárkou.

Vzory mohou obsahovat omezení související s kontrolovaným typem. Atributy se získávají pomocí *get()*. Fakta a jejich atributy mohou být svázány s proměnnou a použity později v pravidle. Po lepší přehlednost se před proměnné píše symbol dolaru ('$'). Pokud jeden vzor obsahuje více omezení, je možné je oddělit čárkou (','), která nahrazuje logickou spojku a zároveň ('&&'). Pokud jsou ovšem ve vzoru použity i jiné logické operátory, jako například nebo ('||'), musí se místo čárky (',') použít logický operátor a zároveň ('&&').

```
$p : Person( age < 18, gender == “M”)
```

příklad vzoru

[15] LHS - Left Hand Side

5.8.3. Pravá strana pravidla

Akční část pravidla se někdy označuje jako následek nebo RHS[16] a obsahuje seznam akcí, které mají být provedeny. Akce mohou být napsány jazyce Java a kód by měl být atomický. Pravá část pravidla by neměla být moc dlouhá, aby byla přehledná a čitelná. Hlavním účelem akční části pravidla je vkládat, modifikovat a stahovat data a fakta z pracovní paměti. Pro tyto účely je možné využít několika funkcí jako například *insert*(), která umístí nový objekt do pracovní paměti nebo funkci *retract()*, která objekt odebere z pracovní paměti. Dále je tu funkce *update()*, která sdělí engine, že se objekt změnil, a může být nutné přehodnotit pravidla.

Díky široké škále komponent pro vyjádření požadované logiky, kterou poskytuje Drools Rule Language, je možné vytvářet efektivní a srozumitelná pravidla. Osvědčené postupy určují, že každé pravidlo by mělo popisovat jeden scénář a v akční části je třeba se vyvarovat vnořené podmíněné logiky a cyklům.

[16] RHS - Right Hand Side

6. GIT

Jedná se o jeden z nejrozšířenějších distribuovaných systémů pro správu verzí. S použitím Git[33] může na jednom projektu efektivně pracovat více vývojářů. To je možné proto, že neexistuje pouze jedna verze projektu, ale několik. Všechny verze projektu jsou uloženy ve vzdáleném repozitáři, kam mají přístup všichni vývojáři. Vývojář si může stáhnout kopii projektu, na které může provést změny. Při ukládání změn zpět do repozitáře musí nejdříve synchronizovat lokání kopii projektu s aktuální verzí projektu z repozitáře, a pak může bezpečně nahrát všechny změny. Velkou výhodou Git je možnost zobrazení veškeré historie změn provedených na daném projektu, a dokonce je možné se k libovolné verzi vracet. Díky větvení verzí je možné vytvářet a ukládat novou funkcionalitu, aniž by změny měly vliv na hlavní verzi projektu.

Git každý soubor uloží pouze jednou, a to při jeho vytvoření. Změny, které byly provedeny na souboru se zapisují do snapshotu[17]. Při ukládání práce na server, Git vytvoří commit (potvrzení), který obsahuje snapshot všech souborů v určitém okamžiku. Sobory jsou ukládány v binárním tvaru, takže je možné ukládat i jiné druhy souborů než pouze textové. Commit v sobě mimo jiné obsahuje informace o tom, kdo a kdy jej vytvořil, a také komentář, který k němu přidal. Jednotlivé commits jsou propojeny, a tvoří tak acyklický orientovaný graf, který znázorňuje historii změn. Z hlavní větve projektu je možné vytvářet i vedlejší větve, a tím dochází k větvení grafu. Jednotlivé větve jsou tvořeny commity vztahujícím se k dané větvi.

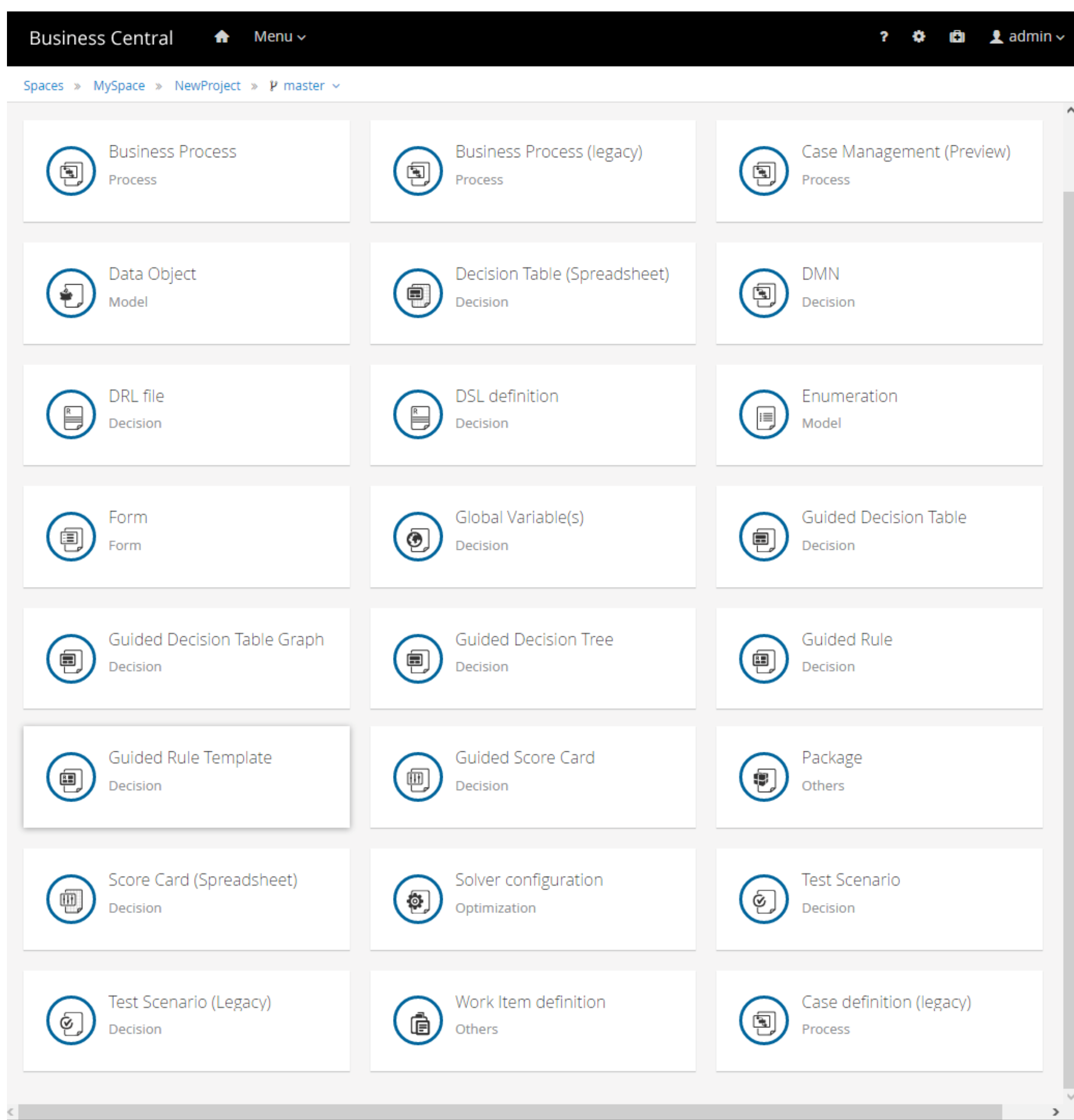
Větvení je velmi vhodné například pro vývoj nové funkcionality. Vývojář si může vytvořit větev z hlavní větvě projektu, ve které může vyvíjet novou funkcionalitu. Jakmile dokončí práci na svojí větvi, může ji spojit s hlavní, a tím docílí vložení svých změn do hlavní verze projektu. Git také umožňuje vytváření žádosti o přijetí změny (pull requests), které umožňují kontrolu kódu jiným vývojářem před sloučením vedlejší větve s hlavní větví. Diskuze v žádostech o přijetí změn jsou neocenitelné pro zajištění kvality kódu a zvýšení znalostí v týmu. [34]

Vývojáři systému drools a především aplikace Business Gentral si Git zvolili pro interní ukládání dat. Git operace dokázali zakrýt před běžným uživatelem, který může s aplikací jednoduše pracovat bez znalosti Git.

[17] snapshot - snímek, který obsahuje stav daného souboru v daném čase

7. Business Central

Business Central, známý také jako KIE Workbench, je plnohodnotná webová aplikace pro sestavování business pravidel a procesů. Mohou se zde vytvářet business pravidla a procesy, které lze následně spravovat a testovat. A to díky řadě nástrojů typu drag-and-drop pro tvorbu pravidel, objektů, procesů a podobně. Pokud uživatel není spokojen s veškerou funkcionalitou Business Centra, může použít editor pro psaní a úpravu pravidel, objektu a podobně. Business Centrum průběžně vyhodnocuje, zda se v projektu nevyskytuje chybný soubor. Pokud na takový soubor narazí, upozorní na tento problém uživatele.



Obrázek 7.1: Přehled nástrojů pro tvorbu business aktiv v aplikaci Business Central

7.1. Seznam nástrojů

Business Process – související s řízením podnikových procesů (BPM). Umožňuje vytvářet procesní diagramy. [35]

Data Object, Enumeration, Package, Global Variable(s) – s jejich pomocí lze vytvářet objekty související se strukturou projektu a datovými modely. Umožní vytváření balíků, globálních proměnných, výčtových typů a datových objektů (POJO). Výčtové typy nebo datové objekty editor dokáže vygenerovat podle zadaných zadních parametrů.

Decision Table, DMN, DRL file, DSL definition, Guided Decision (Table, Tree), Guided Rule, Guided Score Card – jedná se o editory pro vytváření a úpravu aktiv spojených s business pravidly. Funkce umožnují uživatelům vytváření business pravidel bez nutnosti přímého psaní pravidel. Rozhodovací tabulky lze vytvořit s předdefinovanými možnostmi, pomocí šablon pravidel, takže se uživatel může zaměřit pouze na konfiguraci business pravidla. Zastoupeny jsou zde i editory pro vytváření grafických návrhů business pravidel.

Form – tato funkce se používá pro snadné vytváření formulářů, které se mohou použít ke sběru dat od uživatelů, díky kterým mohou být následně upraveny business pravidla. Formuláře byly tradičně vyvíjeny pomocí HTML[18], CSS[19] nebo Javascriptu. Business Central umožnuje automatické generování formulářů na základě parametrů bez nutnosti psaní zdrojového kódu. V základě se jedná o podobnou aplikaci jako Google forms.

Case Management – funkcionalita umožnuje uživateli graficky modelovat jednotlivé procesy pomocí prvku z Case Management Model and Notation (CMMN).

Solver configuration – funkcionalita se používá pro plánování zdrojů. Vytvářením Solver konfigurace lze například řešit problémy s plánováním zdrojů pomocí automaticky škálovatelného cloudového prostředí.

Test Scenario – testovací scénáře umožňují ověřit funkčnost business pravidel a faktů před jejich nasazením do produkčního prostředí. S testovacím scénářem využíváte data z vašeho projektu k nastavení daných podmínek a očekávaných výsledků na základě jednoho nebo více definovaných obchodních pravidel. Při spuštění scénáře se porovnají očekávané výsledky a skutečné výsledky instance pravidla. Pokud se očekávané výsledky shodují se skutečnými výsledky, je test úspěšný. Pokud očekávané výsledky neodpovídají skutečným výsledkům, test selže.

[18] HTML - Hypertext Markup Language je značkovací jazyk používaný pro tvorbu webových stránek, které jsou propojeny hypertextovými odkazy

[19] CSS - Cascading Style Sheets je jazyk stylů používaný pro popis prezentace dokumentu napsaného ve značkovacím jazyce, jako je HTML nebo XML

7.2. *Propojení Business Cental a KieServer*

Kie Server je engine, který je zodpovědný za vyhodnocování obchodních aplikací (kjars). Kie Server se v podnikové verzi také nazývá Intelligent Process Server nebo Decision Server. S ostatními službami může komunikovat za pomoci REST[20] nebo JMS[21].

Business Central je primárně určena pro komunikaci s Kie Server engine, respektive očekává se, že aplikace Business Central bude propojena s Kie Server engine. Po vytvoření projektu a pravidel v něm, je možné jej sestavit a nasadit do úložiště Maven artefaktů. Pokud je Business Central nastaven správně, vytvoří se „kjar" soubor, který je následně nasazen na Kie Server. Zde budou zpracovány a připraveny na vyhodnocení. Kie Server je připojen také k hlavní aplikaci, která mu posílá fakta k vyhodnocení. Business Central umožňuje i sledování jednotlivých Kie Serveru, a dokáže i monitorovat procesy, ke kterým na jednotlivých serverech dochází.

Data v aplikaci Business Central se ukládají do Git repozitáře. Každý soubor, který je vytvořen nebo změněn, je převeden na commit a uložen. S více uživateli, kteří pracují na jednom projektu, dochází ke spoustě úpravám na jednotlivých souborech, tedy i ke spoustě commits. Kvůli tomu se může Git historie stát nepřehlednou. Business Central umožnuje vytvářet a pracovat s větvemi (branches), díky tomu se dá projekt snadno vyvíjet i mezi více uživateli. V nastavení každého projektu je definován odkaz na Git repozitář daného projektu, na který je možné se připojit. Přímý přístup ke Git repozitáři je dostupný pomocí dvou protokolů GIT a SSH[22]. Protokol GIT je k dispozici pouze pro čtení a SSH jak pro čtení, tak i pro zápis. Přímý přístup ke Git umožňuje uživatelům klonovat projekt z Business Central, provádět změny lokálně pomocí preferovaného nástroje a posílat změny zpět do Business Central.

7.3. *Správa uživatelů*

Business Central ověřuje uživatele pomocí autentizace a autorizace. Autentizací je míněno, že aplikace dokáže spolehlivě a bezpečně určit, jaký uživatel se přihlásil. Díky autorizaci uživatele aplikace zjistí, jaká práva na provádění akce a zobrazování dat daný uživatel má. Business Central používá následující role: admin, analytik, vývojář, manažer a uživatel. Každá z rolí má povolené určité možnosti a práva.

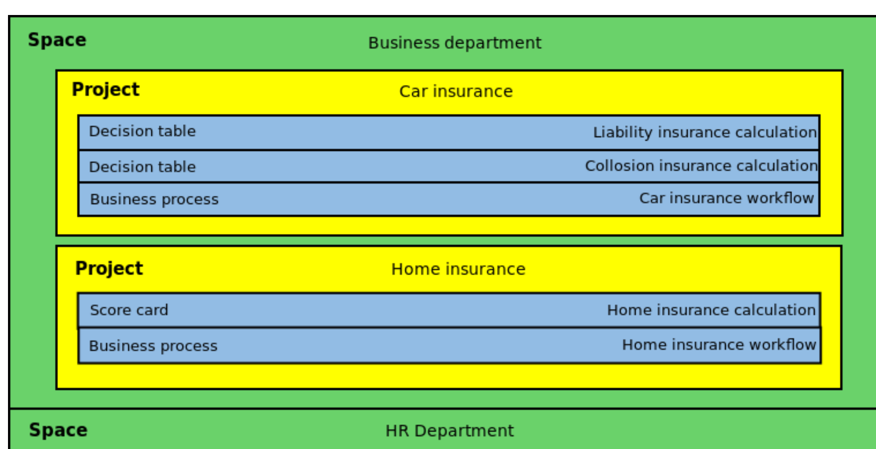
[20] REST - Representational State Transfer je softwarový architektonický styl, který definuje omezení pro vytváření webových služeb

[21] JMS - Java Messaging Services umožnuje jednotlivým komponentám v systému spolu komunikovat

[22] SSH - Secure Shell je zabezpečený komunikační protokol v počítačových sítích, které používají TCP/IP

7.4. Průchod aplikací

Business Central je strukturován na prostory (Space), ve kterých se nachází projekty. Prostory mohou sloužit například pro oddělení produkčního, akceptačního a testovacího prostředí, a mohou obsahovat několik projektů. Projekty jsou následně místem, kde jsou uložena aktiva, a mohou náležet pouze jednomu prostoru. Projekty jsou ve skutečnosti úložiště založené na virtuálním systému souborů, které ve výchozím nastavení používá Git jako backend[23] pro ukládání dat. Business Central umožňuje efektivní využívání funkcí Git, jako je verzování, větvení, a dokonce i externí přístup. Nový projekt lze vytvořit od začátku nebo naklonovat z existujícího repozitáře.

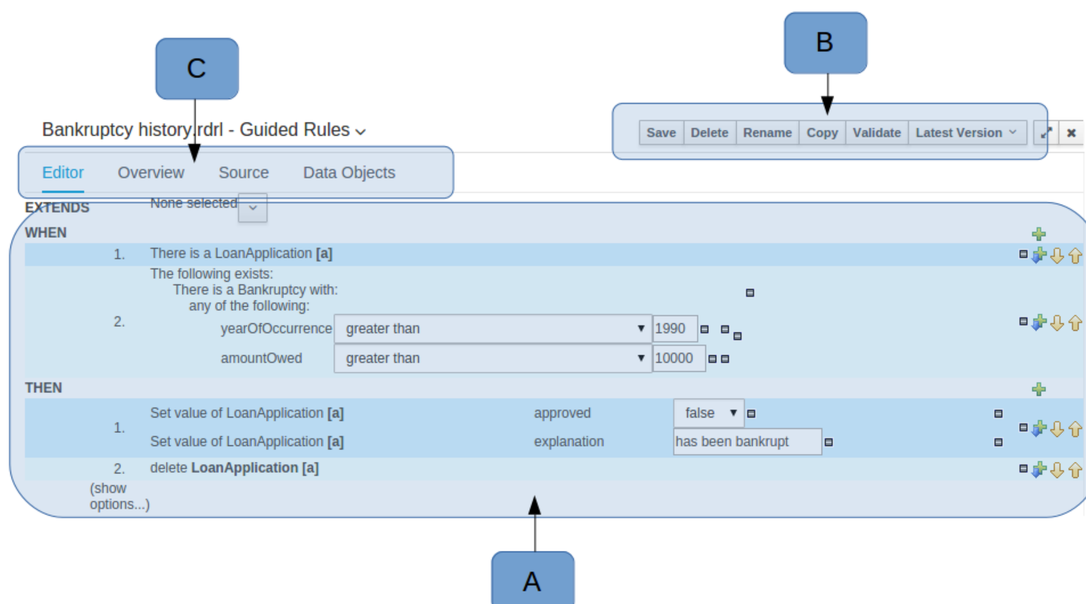


Obrázek 7.2: Struktura projektů v aplikaci Business Central

7.4.1. Editor business pravidel

Po vytvoření projektu uživatel může využít řady editorů, díky kterým může vytvářet a spravovat veškerá aktiva. Aktivum může v jeden okamžik upravovat vždy jen jeden uživatel, aby se předešlo konfliktům při nahrávání úprav. Když uživatel začne upravovat aktivum, automaticky se zamkne. Pokud uživatel začne upravovat již zamčené dílo, zobrazí se vyskakovací oznámení, které uživatele informuje, že aktivum aktuálně nelze upravovat, protože na něm pracuje jiný uživatel. Změny budou zablokovány, dokud editující uživatel neuloží nebo nezavře aktivum nebo se neodhlásí z Business Central. Aplikace obsahuje několik editorů, které již byly vyjmenovány výše. Každý editor je určen pro konkrétní druh aktiva, ale i tak obsahují společné komponenty, například na úpravu aktiv.

[23] backend je označení části webové aplikace, která slouží ke zpracování dat



Obrázek 7.3: Příklad vytváření business pravidla v nástroji Guided Rules

Základní editor se skládá ze tří základních částí, z hlavní části(A), která je měnitelná a zobrazuje aktuální stránku. Na *Obrázek 7.3* je zobrazena stránka pro editaci pravidla, ale každý editor má jinou editační stránku, každý je určen na editování konkrétního druhu aktiv. V sekci B je toolbar[24], který obsahuje akce, jež mají vliv na upravovaná aktiva. Je možné jej uložit, odstranit, přejmenovat, kopírovat, otestovat, a je také možné zobrazit dřívější verze pravidla. Jednotlivé změny, které lze s aktivem udělat, jsou vykonávány na pozadí pomocí Git commit. Předtím, než je změna provedena, je možné přidat krátkou zprávu. V poslední části C je navigate list[25], jehož pomocí je možné se přesunout na další stránky, které popisují dané aktivum anebo obsahující funkcionalitu spojenou s aktivem. V této práci je podrobně představen editor na vytváření a úpravu business pravidel, který je pojmenován Guided Rule. Jednotlivé stránky tohoto editoru budou představeny níže.

Editorská stránka s názvem Editor (v jiných verzí aplikace může být zaván Model) zobrazuje oblasti pravidla, které se dají upravovat. Jedná se o EXTENDS, zde uživatel může vybrat, z jakého, již existujícího, pravidla má nové pravidlo dědit. Dále je zde sekce WHEN, ve které může uživatel nastavit podmínky, za kterých se mají vykonat akce daného pravidla. Podmínky jsou děleny na jednotlivé vzory a v nich si můžeme zvolit objekty, respektive očekávaná fakta, a určit jim očekávané parametry, popřípadě je možné na nich zavolat nějaké

[24] toolbar - ovládací prvek sloužící v uživatelském rozhraní počítače k otevření některých aplikací a doplňků, obsahuje většinou ikony, tlačítka nebo menu
[25] navigate list - ovládací prvek sloužící v uživatelském rozhraní počítače k přepínání mezi jednotlivými stránkami aplikace

funkce. Poslední sekce THEN obsahuje seznam akcí, které se mají vykonat, pokud jsou splněny podmínky v sekci WHEN. Podobně jako v sekci WHEN si je i zde možné vybrat, se kterými objekty se bude pracovat. Je možné v podmínkové sekci vytvořit proměnné, respektive nastavit do proměnné očekávaný fakt, se kterým je možné následně v akční sekci pracovat. I zde je možné na objektu volat různé funkce. V příkladu na *Obrázek 7.3* jde vidět, že v podmínkové sekci je očekáván objekt typu LoanApplication, na který se bude moct odkazovat pomocí proměnné *a*. V akční sekci se pak s objektem *a* pracuje, respektive jsou mu nastaveny proměnné *approve* a *explanation*, a následně je objekt *a* odstraněn z provozní paměti, a tím již nemůže zasáhnout do vykonání jiných pravidel. V této sekci se dají přidávat atributy, které byly vysvětleny v předchozí kapitole.

Obrázek 7.4: Stránka v aplikaci Business centrál s názvem Overview

Stránka s přehledem (Overview) se nachází v každém editoru v aplikaci Business Central a obsahuje informace o daném aktivu, v tomto případě pravidle. Je zde uveden popis daného aktiva, jméno projektu, ve kterém se aktivum nachází, datum poslední změny a datum vytvoření. Dále je zde kompletní historie změn, které byly na aktivu provedeny. Jedná se o seznam obsahující jednotlivé commit vztahující se k danému aktivu. Je možná si jednotlivé změny zobrazit, popřípadě pomocí funkce *Restore* je možné aktivum vrátit do dřívějšího stavu. V sekci *metadata* lze k pravidlu přidat různé značky, které jsou určeny k filtrování. Nachází se zde i přesná cesta k souboru a další údaje. Na stránce přehledu se nachází i konverzační okno, do kterého mohou jednotliví uživatelé přidávat své poznámky či informace.

Na stránce zdroj (source) je možné najít skutečnou podobu pravidla obsaženého v souboru s příponou *.rdlr*. Na této stránce nelze pravidlo nijak měnit. Pro uživatele, kteří chtějí psát pravidla, je určen editor *Drl File.* Rozdíl mezi pravidlem vytvořeným v editoru *Drl File* a G*uided Rules* je patrný právě na příponě. Pravidla vytvořená v editoru *Drl File* mají klasickou příponu *.drl*, kdežto pravidla vytvořená v editoru G*uided Rules* mají příponu *.rdlr*.

Poslední stránka nese název Datové objekty (Data Objects). V ní je možné přidávat a odebírat objekty, které je možné využít v daném pravidle. Je zde možné přidávat základní Java datové objekty z balíčku java.lang a nebo objekty z projektu, ve kterém se pravidlo nachází.

Na vytváření a úpravu Java objektů je určen editor Data Objekts, který obsahuje tři stránky – Editor, přehled a zdroj. Stránka s přehledem je stejná jako u editoru Guiderd Rules. Na stránce Editor lze objekt upravovat pomocí několika nástrojů. Je zde možné přidávat jednotlivé proměnné, u kterých lze nastavovat jméno, datový typ, označení a popřípadě popisek. Stránka Zdroj obsahuje, podobně jako v editoru pravidel, zdrojový kód objektu s tím rozdílem, že zdrojový kód lze měnit. Aplikace podle zadaných parametrů vygeneruje třídu obsahující nadefinované privátní proměnné, prázdný konstruktor a funkce get respektive set pro dané parametry. Vygenerovaný zdrojový kód je možné upravit anebo celý přepsat. Je možné doplnit jednoduché funkce, které je následně možné volat v pravidlech.

8. Implementace

Diplomová práce si klade za cíl navrhnout řešení, které bude řídit počty transakcí pro získání dat od poskytovatele, které není schopen zaslat v jedné odpovědi. Je kladen důraz na snadné a rychlé vytváření konfigurace, aby bylo možné svižně zareagovat na businessové požadavky. Danou konfiguraci musí být schopen vytvořit i uživatel, který nemá žádné znalosti s jakýmkoliv programovacím jazykem. Dále je kladen důraz na robustnost systému a také na vytvoření takového řešení, které bude co nejflexibilnější, aby jej bylo možné využít i v jiném systému na vyřešení podobného, avšak odlišného, problému.

Celkové řešení bude rozděleno do několika systémů. Vše se bude točit kolem business pravidel, ty budou představovat lehce změnitelnou konfiguraci. Drools engine byl zvolen jako rule engine, který bude vyhodnocovat business pravidla. Vybrán byl především z toho důvodu, že systémy od Drools jsou stále vyvíjeny, a mají velmi dobrou dokumentaci. Další výhodou je, že existuje velká základna vývojářů, kteří Drools využívají a diskutují o svých zkušenostech na internetu. Další nespornou výhodou je existence aplikace Business central, ve které je možné vytvářet jednotlivá pravidla, dokonce je může vytvářet i uživatel bez znalosti jakéhokoliv programovacího jazyka.

Jak již bylo zmíněno, aplikace Business Central je navrhnuta tak, aby po sestavení projektu, jenž byl vytvořeném v aplikaci Business Central, byl nasazen na Kie server, na který se mohou připojovat jiné systémy pomocí rozhraní Rest či JSM. Tato architektura ovšem není vhodná pro řešení daného problému hned z několika důvodů. Zaprvé je potřeba se starat o další aplikaci Kie server. Při nasazení nových pravidel bude muset dojít k výpadku. Pro připojení ke Kie serveru bude nutné využívat rozhraní Rest či JSM. Drools engine bude do systému Symphony připojen tak, že objekt servise, která bude obsahovat engine, bude vytvářen za běhu systému. Nicméně jelikož aplikace Business Central využívá k ukládání pravidel Git repositář, je možné se k němu připojit a získat z něj potřebná pravidla. Takto získaná pravidla se mohou uložit do databáze, do které bude mít přístup i systém Symphony. Připojení obstará nově vytvořená aplikace Bridge, která stáhne a zpracuje pravidla, jakmile aplikace Business central uloží data, respektive provede akci commit. Aplikace Bridge může být připojena přímo k databázi a ukládat data rovnou při zpracování anebo je může pomocí rest rozhraní posílat ke zpracování jiné serverové aplikaci.

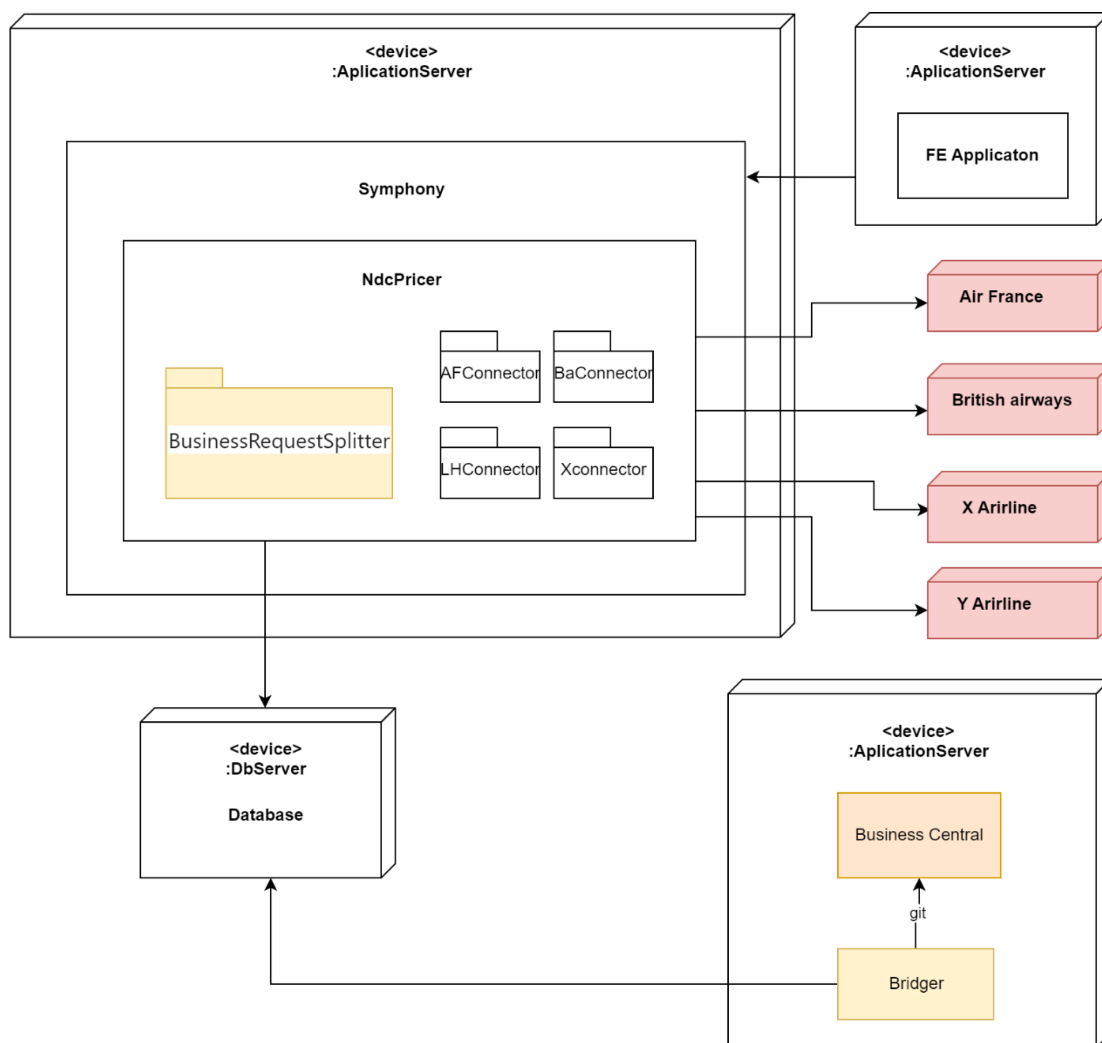
Jakmile budou pravidla v databázi, nic nebrání tomu je začít využívat. Systém Symphony se skládá z mnoha komponent. Komponenty NdcPricer a NdcBooking, se starají o komunikaci s poskytovatelem dat, který poskytuje NDC obsah. Podle standardů IATA se volá dotaz AirShopping pro získání nabídek od poskytovatele dat. Právě získávání nabídek a

vytváření AirShopping dotazů má na starost komponenta NdcPricer. Přesněji řečeno NdcPricer přijímá požadavek z webové aplikace, který obsahuje informace o hledaném letu. NdcPricer z požadavku pozná, jací poskytovatelé dat by se měli zavolat, a následně vytvoří n objektů typu AirShoppingCallInProvider, které obsahují veškerá data o hledaném letu. Následně se jednotlivé objekty pošlou konkrétním konektorům, kde každý konektor zprostředkovává komunikaci s konkrétním poskytovatelem dat. Jak již bylo vysvětleno, existují případy, kdy je pro získání nejlepších nabídek od poskytovatele potřeba rozdělit dotaz na více částí. Funkcionalitu, která bude řídit rozpady dotazů, je proto důležité umístit do NdcPriceru, přesněji do míst, kde je k dispozici list n objektů typu AirShoppingCallInProvider, jenž je možné upravit.

Knihovna obsahující rule engine by měla být dostatečně obecná, aby ji bylo možné využít i v jiných systémech pro řešení jiných problémů. Je proto důležité, aby zde nebyla logika, která je určená pouze pro řešení problému rozpadu AirShopping dotazů. Z tohoto důvodu knihovna nemůže pracovat s objekty typu AirShoppingCallInProvider, místo nich bude používat obecný objekt, který je vhodný i pro jiné využití. Před samotným voláním knihovny z NdcPriceru, bude nutné objekty typu AirShoppingCallInProvider převést na jiné, univerzální objekty. Bude proto vytvořena knihovna s názvem BusinessRequestSplitter, která bude obsahovat Drools engine, jež bude přijímat objekt typu DroolsWrapper a seznam business pravidel. Po vyhodnocení pravidel bude objekt typu DroolsWrapper obsahovat výsledek, podle kterého bude možné v komponentě NdcPricer rozpadnout jednotlivé dotazy.

Na diagramu *Obrázek 8.1* je naznačeno celkové řešení problému. Je zde znázorněn systém Symphony, který dostává požadavky od webové aplikace (vpravo nahoře), které následně zpracuje komponenta NdcPricer. Ta obsahuje několik knihoven zvaných konektory (například AirFranceConnector, BritishAirwaysConnector a další), které se využívají pro komunikaci s poskytovateli dat (červeně zbarvená komponenta). Do komponenty NdcPricer bude přidána knihovna BusinessRequestSplitter(žlutě zbarvená), která bude rozhodovat o tom, jak se mají AirShopping dotazy rozpadnout. Komponenta NdcPricer je propojena s databází, ze které bude načítat business pravidla, jenž bude předávat knihovně BusinessRequestSplitter.

V dolní části diagramu je naznačen server, na kterém poběží aplikace Business Central (oranžově zbarvená), jež se bude využívat pro vytváření business pravidel. Na stejném serveru bude i nově vytvořená aplikace Bridge (žlutě zbarvená), která zajistí zpracování pravidel z aplikace Business Central za pomoci nástroje Git. Aplikace Bridge bude propojená se stejnou databází, s jakou je propojena i komponenta NdcPricer, a díky tomu bude moci do databáze ukládat pravidla po jejich získání z aplikace Business Central.



Obrázek 8.1: Diagram ukazující zapojení knihovny BusinessRequestSplitter systému Symphony a využití aplikace Business Central

Pro ukládání pravidel byla zvolena relační databáze MySql, konkrétně již existující databáze s názvem *db_ndcpricer*, jež je určena pro komponentu NdcPricer. V databázi byla vytvořená tabulka *drl_file* jež obsahuje pouze dva sloupce, a to *file_name* a *file_data*. Jako primární klíč byl zvolen název souboru, jenž je stejný jako název pravidla, které je uloženo v parametru *file_data*. Jelikož rule engine dokáže pracovat pouze s pravidly s unikátním názvem, zvolením názvu pravidla jako primárního klíče zabráníme tomu, aby se do engine dostaly dva soubory se stejným názvem.

9. *Aplikace Bridge*

Aplikace Business Central byla navrhnuta tak, aby se v ní snadno dala vytvářet business pravidla. Jak již bylo zmíněno dříve, aplikace Business Central je strukturována na prostory (Space), ve kterých se nachází projekty. V jednotlivých projektech jsou následně uloženy aktiva, respektive business pravidla. Projekty jsou ukládány do lokálního Git repozitáře. Aplikace je navržena tak, že předpokládá komunikaci se serverem Kie. Po vytvoření projektu a aktiv je možné projekt sestavit a následně nasadit na Kie server. Jak již bylo vysvětleno v předchozí kapitole, v případě této diplomové práce není vhodné využívání Kie serveru, a proto je nutno získat pravidla jiným způsobem. Na lokální Git repozitář se dá připojit pomocí přímého přístupu, který je dostupný přes dva protokoly, a to přes protokol GIT, který slouží pouze pro čtení, a přes protokol SSH, který umožňuje jak čtení, tak i zápis. Přímý přístup ke Git umožňuje uživatelům klonovat projekt z Business Central, provádět změny lokálně pomocí preferovaného nástroje a posílat změny zpět do Business Central.[36] Odkaz na Git repozitář je v každém projektu uveden v nastavení, například v takovémto tvaru "ssh://admin@localhost:8001/MySpace/Airshopping", kde „MySpace“ je jméno prostoru a „Airshopping“ je jméno projektu.

Pro stahování a ukládání pravidel z aplikace Business central do databáze byla vytvořena aplikace Bridge, která je napsána v jazyku Java. Jedná se o aplikaci, která po spuštění zkontroluje, zda byly v repositáři vykonány nějaké změny, a pokud ano, repositář je stáhne do svého lokálního uložiště a následně změněné soubory uloží do databáze, popřípadě je odešle ke zpracování jiné serverové aplikaci. Pro práci s Git aplikace využívá knihovnu JGit[37]. Jedná se o Java implementaci systému Git, která implementuje většinu Git příkazů. JGit lze jednoduše integrovat do Java projektu, jelikož se jedná o Java knihovnu. Pro většinu metod, které JGit poskytuje, je zapotřebí mít lokální kopii repositáře.[38]

Po spuštění aplikace nejprve zkontroluje, zda již existuje definovaná složka s lokálním repositářem. Pokud ji nenalezne, složku vytvoří a následně příkazem *cloneRepository* stáhne Git repositář do dané složky. Následně nalezne všechny soubory s pravidly a uloží je do databáze, popřípadě odešle k zpracování jiné serverové aplikaci. Pokud ovšem kontrola ukáže, že složka s repositářem již existuje, pomocí příkazu *diff* aplikace zjistí, jestli a jaké soubory byly přidány, odstraněny či změněny, a dané soubory následně uloží do databáze, popřípadě je z databáze odstraní.

Příkaz *diff* má dva stromové objekty, které reprezentují hierarchii mezi soubory v Git repositáři, proto příkaz očekává dva parametry typu AbstractTreeIterator, jež určí starý a nový strom, které mezi sebou porovná. Jakmile jsou oba stromy porovnány, metoda call()

vrátí seznam objektů tipy DiffEntries. Jednotlivé objekty v seznamu popisují dané soubory, které byly přidány, odebrány či změněny a lze je také použít k určení změn v rámci určitého souboru. Tříd, které dědí z třídy AbstractTreeIterator, je několik, v aplikaci Bridge je použita třída CanonicalTreeParser. Tu je po vytvoření potřeba pomocí metody *reset* nastavit na konkrétní objekt stromu. Vyžaduje dva parametry, objekt typu ObjectReader, jenž umožňuje procházet jednotlivé soubory v repositáři, a především identifikátor daného stromového objektu.[39]

Výše uvedenou funkcionalitu v aplikaci Bridge obsahuje metoda pullAndDiffRep(). V ní dochází nejprve k uložení identifikátoru aktuálního stromu do proměnné oldHead a následně dojde k provedení příkazu *pull*, jenž aktualizuje daný repositář. Následně je do proměnné newHead uložen identifikátor aktuálního stromu, který je odlišný od předchozího za předpokladu, že došlo ke změnám v repositáři. Oba identifikátory jsou následně použity pro vytvoření objektu typu *CanonicalTreeParser*, který je předán do funkce *diff*. Metoda *pullAndDiffRep()* následně vrací seznam objektů tipy DiffEntries, který vrátila metoda *call()*.

```
Git git = Git.open(new File(path)) ;
Repository repository = git.getRepository();

ObjectId oldHead = repository.resolve("HEAD^{tree}");

git.pull()
      .setTransportConfigCallback(new SshTransportConfigCallback())
      .setCredentialsProvider(new UsernamePasswordCredentialsProvider(userName, userPass))
      .call();
ObjectId newHead = repository.resolve("HEAD^{tree}");

ObjectReader reader = repository.newObjectReader();

CanonicalTreeParser oldTreeIter = new CanonicalTreeParser();
oldTreeIter.reset(reader, oldHead);

CanonicalTreeParser newTreeIter = new CanonicalTreeParser();
newTreeIter.reset(reader, newHead) ;

List<DiffEntry> diffs = git.diff()
      .setNewTree(newTreeIter)
      .setOldTree(oldTreeIter)
      .call();
```

Aby aplikace Bridge nemusela být spouštěna manuálně po každé úpravě v aplikaci Business Central, je možné využít Git háky (Git Hooks). Jedná se o funkcionalitu, která umožňuje spuštění skriptu, když dojde k určitým důležitým akcím. Dané skripty musí být uloženy v podadresáři daného Git adresáře s názvem hooks. Ve většině projektech je cesta

ke složce ve tvaru *.git/hooks*, nicméně v aplikaci Business central je to jiné. Všechny projekty jsou uloženy ve složce .niogit, každý projekt má svůj repozitář. Struktura složky .niogit je naznačena na *Obrázek 9.1.* Cesta ke složce hooks daného projektu je pak .niogit/SpaceName/ProjectName/hooks, kde SpaceName je jméno prostoru, jemuž projekt náleží a ProjectName je jméno daného projektu. Skripty ve složce hooks mohou být napsány v různých jazycích jako jsou Bash[26], Ruby [27]nebo Python[28]. Aby byl soubor spuštěn, musí být správě pojmenován, jelikož po určité akci je spuštěn jen konkrétní skript. Je mnoho druhů skriptů, mezi které patří například pre-commit, jenž je spuštěn před samotnou akcí commit a může sloužit ke kontrole daných změn. K podobným účelům slouží i skript pre-push, který je spuštěn při akci push a umožnuje danou akci i zastavit, pokud vyhodnotí, že je to nutné.[40] Nicméně Business Central podporuje pouze jeden druh skriptu – post-commit. Skript je spuštěn po každé akci commit a aplikace následně čeká na dokončení daného skriptu. Díky tomu nemůže dojít k změně souborů, se kterými skript pracuje. Pokud se vyžaduje, aby každý projekt v aplikaci Business Centrál používal stejný post-commit skript, je potřeba jej nakopírovat do všech hooks složek. Při správném nastavení systémových proměnných je možné kopírovaní skriptu nastavit tak, aby se při vytvoření nového projektu post-commit skript zkopíroval do složky hooks v daném projektu.[41]

Post-commit skrip, který spustí aplikaci Bridge a předá ji cestu k projektu, na kterém jsou prováděny změny, vypadá následovně.

```
#\!/bin/bash
java -jar /opt/jboss/wildfly/bin/bridge.jar $PWD
```

Poté co aplikace Bridge stáhne pravidla z Business Central, převede je na objekty a uloží je do databáze. Dané ukládání je provedeno pomocí Java Database Connectivity (JDBC), což je specifikace aplikačního programovacího rozhraní (API), které definuje jednotné rozhraní pro přístup k relačním databázím. Rozhraní JDBC je napsáno v jazyce Java a skládá se ze sady rozhraní a tříd, které je možné využít při tvorbě aplikace, která má za úkol komunikovat s databází. V aplikaci se vytváří dotazy v jazyce SQL[29], které se následně posílají k vykonání dané databáze.[42] Pokud ovšem není možné se připojit k databázi pomocí rozhraní JDBC, například z bezpečnostních důvodů, je možné pravidla odesílat ke zpracování do jiné serverové aplikace, jež následně může bezpečně komunikovat

[26] Bash je jeden z unixových shellů, který interpretuje příkazový řádek
[27] Ruby je interpretovaný skriptovací programovací jazyk
[28] Python je programovací jazyk, který je vyvíjen jako open source projekt
[29] SQL je standardní jazyk pro ukládání, manipulaci a získávání dat v databázích

s databází. Tuto komunikaci lze provést pomocí Rest rozhraní, například pomocí knihovny java.net.

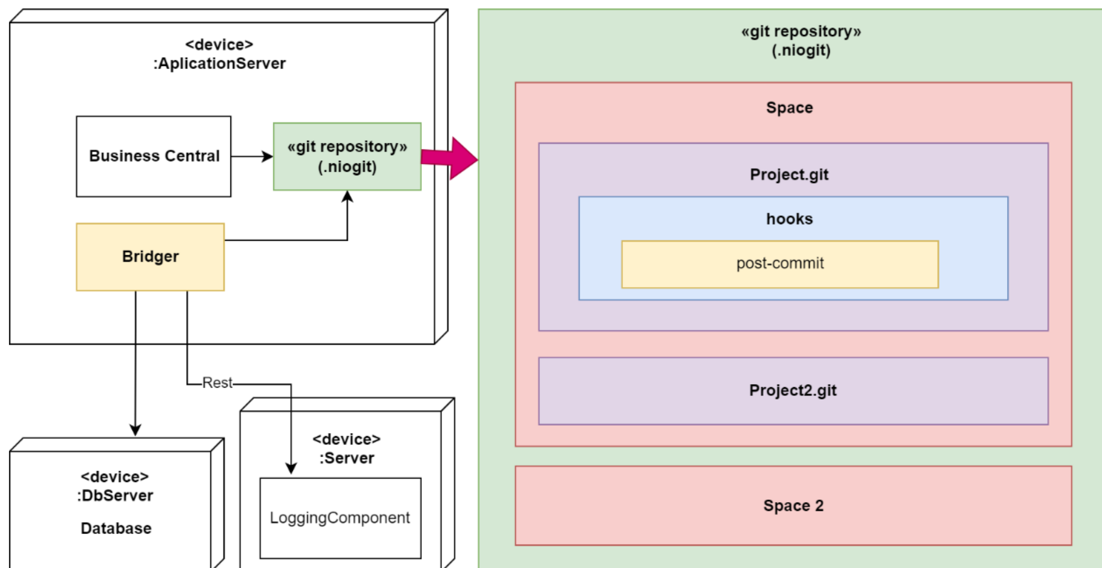
O komunikaci s databází se stará DrlFileService. Jedná se o třídu, jež obsahuje metody *saveDrlFiles*() pro ukládání pravidel, *removeDrlFiles*() pro odstranění pravidel a *getAllDrlFiles*() pro získání pravidel z databáze. Metoda *saveDrlFiles*() postupně uloží všechny objekty obsahující pravidla. Nejprve se zavolá SQL příkaz *select* s názvem pravidla, pro zjištění, zda daný záznam již existuje. Pokud bylo pravidlo již někdy uloženo, je zavolán SQL příkaz *update*, jenž daný záznam aktualizuje. Pokud ovšem příkaz *select* nevrátí žádnou odpověď, znamená to, že je potřeba zavolat SQL příkaz *insert*, který dané pravidlo uloží do databáze.

```
public void insertDrlFile(DrlFile drlFile) {
   Class.forName(JDBC_DRIVER_CLASS_NAME);
   Connection connection = DriverManager.getConnection(dbUrl, dbUser, dbPass);

   Statement statement = connection.createStatement();
   String query = "insert into drl_file (file_name, file_data) values (?, ?)";

   PreparedStatement preparedStmt = connection.prepareStatement(query);
   preparedStmt.setString(1, drlFile.getFileName());
   preparedStmt.setString(2, drlFile.getData());
   preparedStmt.execute();

   statement.close();
   connection.close();
}
```

Ostatní metody fungují podobně jako *saveDrlFile*(). Metoda *removeDrlFiles*() se používá při odstranění pravidla, popřípadě při přejmenování pravidla, jenž je rozděleno do dvou kroků – odstranění pravidla se starým názvem a uložení stejného pravidla s novým názvem. Pro odstranění pravidla se nejprve zkontroluje, zda pravidlo pro opravu v databázi existuje, to se zjistí pomocí příkazu *select*. Pokud pravidlo opravdu existuje, pro odstranění se zavolá příkaz *delete*,. Poslední metoda *getAllDrlFiles*() pomocí příkazu *select* vrací všechna pravidla, která jsou v databázi uložena.

Pro správný běh jakékoliv aplikace je důležité zaznamenávat událostí, které se v aplikaci dějí. Proto byla ve společnosti AARON GROUP vytvořena logovací komponenta, která umožnuje přehledné zobrazování záznamů o událostech, které se v aplikacích staly. S komponentou se dá komunikovat pomocí JMS anebo Rest rozhraní. V aplikaci Bridge je právě využitá druhá možnost. Jelikož aplikace bude komunikovat v zabezpečené síti, pro vytvoření jednoduchého Rest klienta byla použita knihovna java.net, respektive třída HttpURLConnection. Pro zaznamenávání událostí v aplikaci Bridge byla vytvořena třída

LogCom, která poskytuje metody pro zaznamenávání různých typů událostí. Jakmile se aplikace spustí, je vytvořena instance třídy LogCom, která v průběhu běhu aplikace shromažďuje záznamy o událostech. Před ukončením běhu aplikace se uložené záznamy převedou do jedné zprávy, která je následně poslána logovací komponentě pomocí Rest rozhraní.



Obrázek 9.1: Diagram nastiňující problematiku využití aplikace Business Central

Na diagramu *Obrázek 9.1* je naznačena aplikace Business Central, která ukládá data do lokálního Git repositáře. Jeho struktura je naznačena v pravé straně obrázku. Dále je v levé části diagramu naznačena aplikace Bridge, která stahuje data z daného repozitáře a ukládá je do databáze.

10. Vytváření pravidel

Návrh správné struktury pravidel je velmi důležitý, jelikož úpravy a změny v budoucnu by mohly znamenat přepracování mnoha již vytvořených pravidel. Je proto potřeba dbát na zpětnou kompatibilitu pravidel i objektů, které s nimi interagují. Pro návrh pravidel lze využít velké množství funkcionalit, které drools business rule nabízí.

10.1. Problematika zvolení správné Třídy

Jak již bylo ukázáno, Business Central i Drools engine dokáží pracovat skoro s jakýmkoliv typem objektu. Je proto namístě uvažovat o tom, zda by nebylo nejjednodušší pracovat se samotným objektem, ve kterém jsou obsažena data, podle kterých by se měla pravidla rozhodovat. To ovšem není dobrý nápad, a to hned z několika důvodů. Třída, jež je vzorem daného objektu, může být velmi složitá, a může obsahovat mnoho vnořených tříd. Kvůli tomu mohou být důležitá data uložena hluboko v objektu, a aby pravidla mohla s takovým objektem pracovat, musela by v sobě mít mnoho zbytečné logiky, která by umožňovala uživateli se dostat i k těmto vnořeným datům. To by zapříčinilo znepřehlednění pravidel a vedlo by to i ke zhoršení vytváření daných pravidel. Aby bylo možné v aplikaci Business Central vytvářet pravidla, jež by pracovala s daným složitým objektem, bylo by potřeba v aplikaci mít nadefinovanou nejen třídu daného objektu, ale i všechny neprimitivní třídy, které hlavní třída obsahuje.

Z těchto důvodů je potřeba objekt obsahující požadovaná data převést na takový, který bude jednodušší a univerzální. S takovým objektem se již v pravidlech mnohem lépe pracuje. Rule engine může pracovat s různými typy objektů, a tak i jednotlivá pravidla mohou očekávat různé typy objektů. Je však důležité si uvědomit, že všechny třídy, jež jsou vzory objektů, musí být nadefinovány jak v aplikaci Business Central, tak i v aplikaci obsahující rule engine. Jak již bylo vysvětleno, na začátku každého pravidla jsou definovány cesty k třídám, se kterými dané pravidlo pracuje. Jakmile je pravidlo vloženo do rule engine, zkontroluje se, že třídy nadefinované v daném pravidlu existují. Je potřeba si dát pozor na to, aby třída v aplikaci Business Central a třída v aplikaci, která obsahuje rule engine, byla totožná, respektive minimálně se musí totožně jmenovat, musí mít stejně nadefinovaný balík a třída, se kterou pracuje rule engine, musí mít minimálně stejné proměnné a funkce, se kterými se pracuje v daném pravidle. Pokud tedy v budoucnu budou třídy měněny, je potřeba dané zněny provést na dvou místech. To ovšem za předpokladu, že se v aplikaci Business Central s danou třídou pracuje pouze v jednom projektu. Pokud se třída používá ve více projektech, zajištění provedení změn ve všech třídách je mnohem složitější.

10.2. Yaml

Drools pravidla poskytují funkcionalitu *declare*, pomocí níž lze vyřešit problematiku různě definovaných balíků v aplikaci obsahující rule engine a aplikaci Business Central. Díky funkcionalitě *declare*, lze místo importování třídy v pravidle objekt dané třídy rovnou deklarovat. Deklarovaný objekt v pravidle je ve formátu YAML[30], který se používá pro serializaci strukturovaných dat. Třídy, jež jsou vzory objektů, stále musí být definovány i v aplikaci Business Central, ale již nemusí mít definovaný stejný balík jako třída, která je definována v aplikaci obsahující rule engine. Editor pravidel Guided Rule ovšem nedokáže pracovat s funkcionalitou *declare*, proto je potřeba využít pomocnou aplikaci Bridge. Jakmile dojde k uložení pravidla, aplikace Bridge před uložením do databáze nejprve převede třídy nadefinované v aplikaci Business Central do formátu YAML, který je následně vložen do pravidla, jež s objektem dané třídy pracuje. Takto vytvořená pravidla mohou být načtena do rule engine, kde mohou interagovat s objektem stejného typu, který je nadefinován v daném pravidle. Třída, jež je vzorem objektu, který vstupuje do pravidla, nemusí být totožná s třídou v aplikaci Business Central, ale musí obsahovat proměnné, které jsou nadefinované v daném pravidle.

10.3. Výčtový typ

Pro vytváření pravidel je možné využít i datový typ s názvem výčtový typ (enum), který je tvořen konečnou množinou pojmenovaných hodnot. S jeho pomocí je tak možné vytvořit seznam všech možností, které jdou v pravidle použít. Dostáváme se zde ovšem ke stejnému problému jako s třídami – výčtový typ musíme udržovat aktuální ve více aplikacích.

10.4. Primitivní datové typy a základní třídy Java

Aby celkový systém správně fungoval a byl jednoduše spravovatelný, je vhodné vytvořit takovou třídu, kterou nebude potřeba měnit nebo rozšiřovat pokaždé, když se změní hlavní třída obsahující data, podle kterých se mají pravidla rozhodovat. Proto je vhodné využívat primitivní datové typy anebo základní třídy, které Java nabízí. Jedná se o třídy z knihovny Java.lang, ve které se nachází nejdůležitější třídy jazyka Java. Je zde umístěna třída Object, což je kořen hierarchie tříd. Z ní vychází všechny ostatní třídy v Javě. Všechny objekty, včetně kolekcí, implementují metody třídy Object. Další důležitou třídou v knihovně

[30] Ain't Markup Language je formát pro serializaci strukturovaných dat.

Java.lang je třída String, která představuje znakový řetězec. Třída Java String nabízí mnoho metod pro práci s textovými řetězci, jako jsou například funkce *split*() a *substring*(), které umožnují rozdělení na menší části nebo lze použít funkci *replace*() pro nahrazení určité části textu. Dále je možné využít třídy z knihovny Java.util, která obsahuje mnoho užitečných tříd a metod, jež se dají použít pro generování čísel, či k práci s datem a časem. Především ale knihovna obsahuje kolekce a další nástroje pro práci s řetězcem. Díky kolekcím, které jsou implementovány pomocí tříd a rozhraní v knihovně Java.util, je umožněno snadné ukládání a organizace skupiny objektů. Způsob ukládání a implementace kolekcí jsou různé. Jedná se například o kolekci *List*, do které se dají ukládat objekty jednoho typu, umožnuje vkládání duplikací, a dokonce i nulového prvku. List obsahuje metody založené na indexu pro vkládání, aktualizaci, odstraňování a vyhledávání prvků. Implementací Listu může být mnoho. Jedná se například o ArrayList nebo LinkedList. Rozhraní typu Set je neuspořádaná kolekce objektů, do kterých nelze uložit duplicitní hodnoty. Jeho implementací je například třída HashSet. Dalším typem kolekce je rozhraní Map, které umožňuje ukládání hodnot podle klíčů. Jeden klíč nelze do mapy vložit dvakrát, jelikož se používá pro vyhledávání prvků v mapě. Implementací rozhraní Map je například třída HashMap.

10.5. Návrh objektu DroolsWraper

V problematice rozpadu dotazů, které jsou posílány poskytovateli dat, je hlavním objektem, ve kterém jsou uložena data, objekt třídy AirShoppingQuery, jenž se skládá z mnoha dalších vnořených tříd. Ve zkratce obsahuje informace o vyhledávání daného letu. Objekt třídy AirshoppingQuery proto není vhodný pro vytváření a interakci s business pravidly. Je důležité, aby navržená třída byla jednoduchá a univerzální. Pro rozhodnutí o tom, jakým způsobem bude dotaz rozdělen, nebudou potřeba všechna data, která se v objektu třídy AirshoppingQuery nachází. V některých případech nebude záležet na konkrétních datech, ale spíše na jejich přítomnosti v dotazu. Například pro aerolinku British Airways může existovat následující požadavek: „Pokud je v dotazu specifikován konkrétní Frequent flyer program, budou do British Airways zaslány dva dotazy. Jeden s konkrétním Frequent flyer programem a druhý bez.“ Z tohoto požadavku je patrné, že pro rozhodnutí, zda se má dotaz rozpadnout, není potřebné znát konkrétní číslo a typ programu, důležitá je informace o přítomnosti daného programu. Informace o Frequent flyer programu se dá zkrátit na bitovou informaci přítomen, nepřítomen.

S ohledem na výše zmíněné pozorování byl navržen objekt DroolsWraper, který má dva parametry. První se nazývá *inputPars* a je typu Map. Parametr inputPars slouží pro ukládání dat, podle kterých se budou pravidla rozhodovat. Klíč i hodnota budou typu

String, jelikož se jedná o univerzální třídu, do které se dají ukládat data. Díky tomu, že parametr inputPars je typu Map, lze do něj ukládat i konkrétnější informace, například přesné hodnoty konkrétních proměnných. Druhý parametr se nazývá *resultSet* a je typu Set. Parametr resultSet slouží k ukládání výsledků po vyhodnocení pravidel. Díky tomu, že se seznam s výsledky nachází ve stejné třídě jako vstupní data, není potřeba používat více typů tříd.

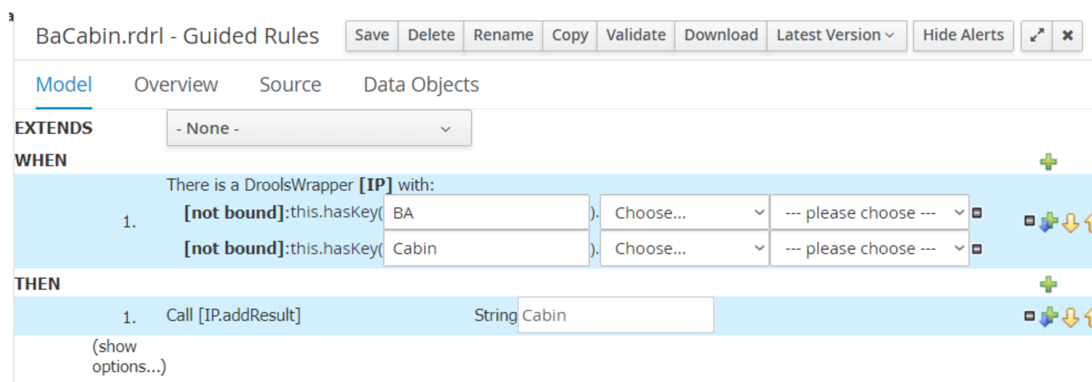
Výše uvedený příklad dotazu AirshoppingQuery s Frequent flyer program, lze uložit do proměnné inputPars, pod klíčovým slovem „FrequentFlyer“ a s hodnotou „present“. Hodnota v tomto případě není podstatná, jelikož pravidlo pro výše uvedený příklad bude pracovat pouze s klíčovým slovem „FrequentFlyer“. Výhody mapy se dají docenit například při ukládání typu cestovní třídy. Pokud v dotazu AirshoppingQuery bude specifikována preference na konkrétní typ cestovní třídy, uloží se tato informace do proměnné inputPars pod klíčem „Cabin“ a hodnotou rovné typu třídy. Pokud pravidlo vyhodnotí, že se má dotaz rozpadnout, například podle typu cestovní třídy, v akční části pravidla uloží klíčové slovo „Cabin“ do proměnné resultSet. V třídě DroolsWraper lze vytvořit několik metod, které lze využívat při tvorbě pravidla. Pomocí metody *hasKey*() je možné zjistit, zda parametr inputPars obsahuje daný klíč. Podobně lze využít funkci *hasNotKey*() k ověření, že parametr inputPars neobsahuje daný klíč. Dále je možné využít funkci *is*(), která přijímá dva parametry – klíč a hodnotu. Funkce se pokusí nalézt výsledek podle zadaného klíče v mapě inputPars, a pokud výsledek nalezne, zkontroluje, zda neobsahuje více hodnot. Pokud ano, pokusí se v nich nalézt hodnotu shodnou se zadanou hodnotu v parametru. Funkce *is()* následně vrací kladnou hodnotu, pokud pod daným klíčem našla zadanou hodnotu, a zápornou, pokud nikoliv. Metoda *isNot*() je následně negací funkce *is*(). Poslední metodou je funkce *addResult*(), kterou je možné využít v akční části pravidla pro uložení výsledku do parametru resultSet.

```
public class DroolsWrapper implements Serializable {

    private Map<String, String> inputPars;
    private Set<String> resultSet;

    public DroolsWrapper() {}

    public Map<String, String> getInputPars(){
        return inputPars;
    }
    public void setInputPars(Map<String, String> inputPars){
        this.inputPars = inputPars;
    }
    public void addParameter(String key, String value){
        if(this.inputPars == null){
            this.inputPars = new HashMap<>();
        }
        inputPars.put(key,value);
    }
    public Set<String> getResultSet() {
        return resultSet;
    }
    public void setResultSet(Set<String> resultSet) {
        this.resultSet = resultSet;
    }
    public void addResult(String result) {
        if(this.resultSet == null){
            this.resultSet = new HashSet<>();
        }
        this.resultSet.add(result);
    }

    public boolean hasKey(String key){
        return this.inputPars != null && this.inputPars.containsKey(key);
    }
    public boolean hasNotKey(String key){
        return this.inputPars == null || !this.inputPars.containsKey(key);
    }
    public String getValue(String key){
        if(this.inputPars != null && this.inputPars.containsKey(key)){
            return inputPars.get(key);
        }
        return "";
    }

    public boolean is(String key, String value){
        if(this.inputPars == null || !this.inputPars.containsKey(key)){
            return false;
        }
        if(this.inputPars.get(key).contains(",")){
           return Arrays.asList(inputPars.get(key).split(",")).contains(value);
        }
        return inputPars.get(key).equals(value);
    }
    public boolean isNot(String key, String value){
        return !is(key, value);
    }
}
```

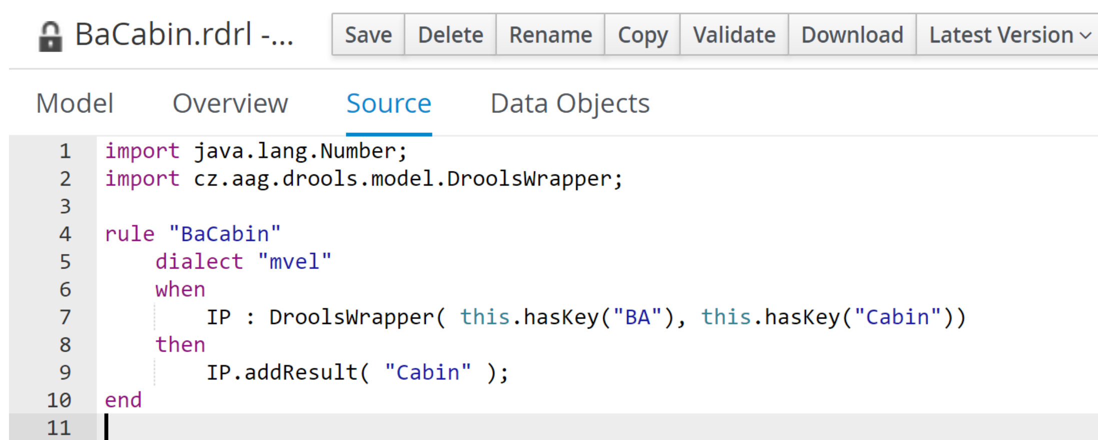
10.6. Vytváření pravidla v Business Central

Díky metodám ve třídě DroolsWraper je možné v aplikaci Business Central jednoduše vytvářet pravidla. Nejprve je potřeba mít v projektu, ve kterém je zamýšleno vytvořit pravidlo, třídu DroolsWrapper. K vytvoření pravidla je možné použít editor Guided Rule. Po pojmenování pravidla je možné si v sekci WHEN zvolit objekt třídy DroolsWrapper, na kterém se budou volat metody. Následně je potřeba objekt pojmenovat. K pojmenovanému prvku je následně možné přidávat výrazy, ve kterých se dají volat všechny metody třídy DroolsWraper. Jakmile je metoda vybrána, je možné zadat její parametry. Po dokončení podmínkové sekce je možné upravit i akční část pravidla. Zde je možné vybrat objekt z podmínkové části a zavolat na něm metodu addResult(). Následně, pokud je aplikace Bridge správně nastavena, je možné pravidlo uložit do databáze.

Na obrázku níže lze vidět vytvořené pravidlo s názvem *BaCabin*, které bylo vytvořeno podle následujícího zadání. Pokud se jedná o dotaz pro dopravce British Airways a v dotazu je specifikován typ kabiny, rozpadne se dotaz na dva – jeden bez definice typu kabiny a druhý s definicí typu kabiny.



Obrázek 10.1: Ukázka obrazovky z aplikace Business Central při vytváření business pravidla



Obrázek 10.2: Ukázka výsledného business pravidla vytvořeného v aplikaci Business Central

11. Knihovna Business Request Splitter

Pro problematiku rozpadu dotazů byla vytvořena univerzální Java knihovna Business Request Splitter, jež implementuje drools rule egine. V knihovně se nachází service RequestSplitterController, která při vytvoření vyžaduje seznam business pravidel a odkaz na logovací komponentu. Po vytvoření RequestSplitterController je možné na objektu zavolat funkci *process*(), která na svém vstupu očekává objekt DroolsWrapper. Poté dojde k vyhodnocení pravidel a funkce *process*() vrátí stejný objekt, který dostala na vstupu, s nově uloženým výsledkem v parametru resultSet.

V kapitole 5.7. byl představen drools engine a byla popsána práce s KieService při vytváření aplikace využívající rule engine. Při využívání drools rule engine se předpokládá, že pravidla budou sestavena s projektem, ve kterém se rule engine nachází. Tato pravidla bývají nadefinována v dokumentu kmodule.xml a uložena ve složce src/main/resources. Tento přístup je ovšem v knihovně Business Request Splitter nevhodný, jelikož při sestavování projektu budou pravidla uložena v databázi, a ne v projektu jako takovém. Pravidla budou předána engine při jeho vytvoření, proto je vhodné využít třídu KieHelper, která se nachází v knihovně org.kie.internal.utils. KieHelper je obslužná třída, která umožňuje vytvářet KieContainer zadáním zdrojů, které do něj chceme zahrnout. Jinými slovy, je možné vytvořit objekt KieHelper a následně pomocí funkce *addContent*() do něj vložit business pravidla ve formě objektu typu String. Následným sestavením objektu KieHelper se získá objekt KieBase, ze kterého se následně získá KieSession. Jedinou nevýhodou třídy KieHelper je fakt, že není součástí veřejného API Drools. Kvůli tomu může v budoucnu trpět zpětně nekompatibilními změnami.[31]

```
private KieBase getKieBase(List<String> rules) {
   KieHelper kieHelper = new KieHelper();
   rules.stream()
         .filter(this::isRuleValid)
         .forEach(rule -> kieHelper.addContent(rule, ResourceType.DRL));
   Results results = kieHelper.verify();
   logResult(results, rules);
   if(results.hasMessages(Message.Level.ERROR)){
      throw new IllegalStateException("Compilation errors during Creating KieBase.");
   }
   return kieHelper.build();
}
```

Poté co je vytvořena instance třídy KieBase, která obsahuje informace o pravidlech, je možné z ní vytvořit KieSession, jež představuje instanci rule engine. Existují dva typy KieSession – stavová (Statefull) a bezstavová (Stateless). V tomto případě bude stačit bezstavová KieSession, jelikož se logika v pravidlech bude rozhodovat pouze na základě kolekce dat z objektu DroolsWrapper. Po získaní instance bezstavové KieSession, přesněji objektu typu StatelessKieSession, je možné na objektu zavolat funkci *execute*(), s jejíž pomocí lze do engine vložit objekt, respektive fakt. Po vložení faktu do engine ihned dojde k vyhodnocení pravidel.

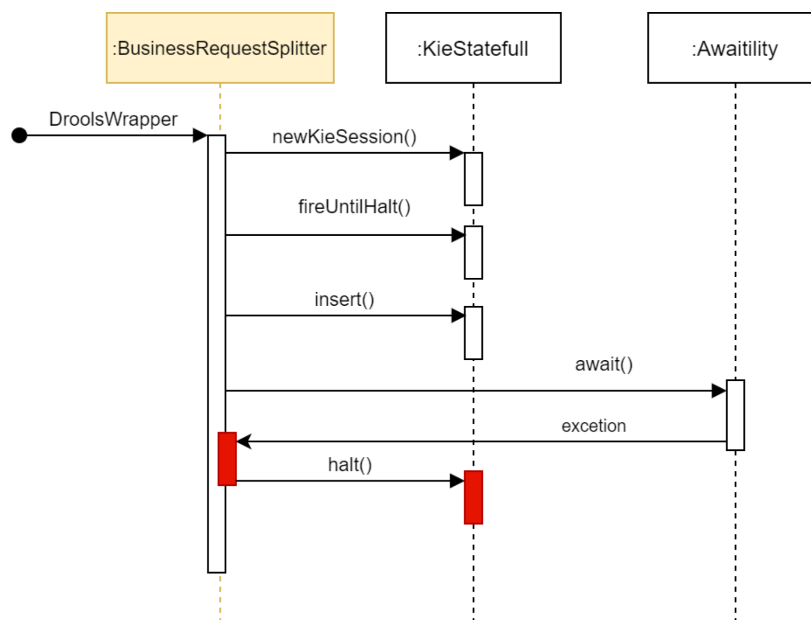
11.1. Bezpečnost

Důležitým aspektem je bezpečnost a prevence. Je potřeba předcházet pádu aplikace, ke kterému může dojít vložením nevhodného pravidla. Proto se každé pravidlo před vložením nejprve otestuje na přítomnost klíčových slov, která musí obsahovat každé pravidlo formátu Drl. Po vložení pravidel do objektu KieHelper je možné zavolat funkci *verify*(), která ověří, že nebyla vložena chybná pravidla. Metoda *verify*() vrací seznam zpráv, které mohou být trojího typu, a to informační, varovné a chybové. Pokud metoda vrátí chybové zprávy, není bezpečné objekt KieHelper sestavit.

Dále je důležité zabránit zacyklení engine. K tomu může dojít v případě, že má třída reprezentující fakt implementovanou metodu, která se může zacyklit. Následně, pokud je tato metoda zavolána, dojde k zacyklení a běh engine skončí přetečením zásobníku (stack overflow). Zabránit danému problému lze několika následujícími způsoby.

11.2. Využití metody fireUntilHalt při zacyklení

První způsob vyžaduje využití stavové implementace třídy KieSession. Ta totiž nabízí metodu *fireUntilHalt*(), která spustí engine v aktivním režimu, ve kterém se engine chová asynchronně. Pravidla jsou průběžně vyhodnocována a spouštěna, dokud není provedeno volání metody *halt(),* která okamžitě přeruší vykonávání. Společně s knihovnou Awaitility, která usnadňuje práci se synchronizací asynchronních operací, je možné naimplementovat následující logiku.[43] Po vložení faktu do engine dojde k vyhodnocení. Pomocí knihovny Awaitility algoritmus počká, dokud nebudou vyhodnocena všechna pravidla, anebo dokud nevyprší předem stanovaná časová doba. Pokud engine nestihne vyhodnotit všechna pravidla do stanovaného limitu, zavolá se metoda *halt()*, která ukončí vykonávání engine.

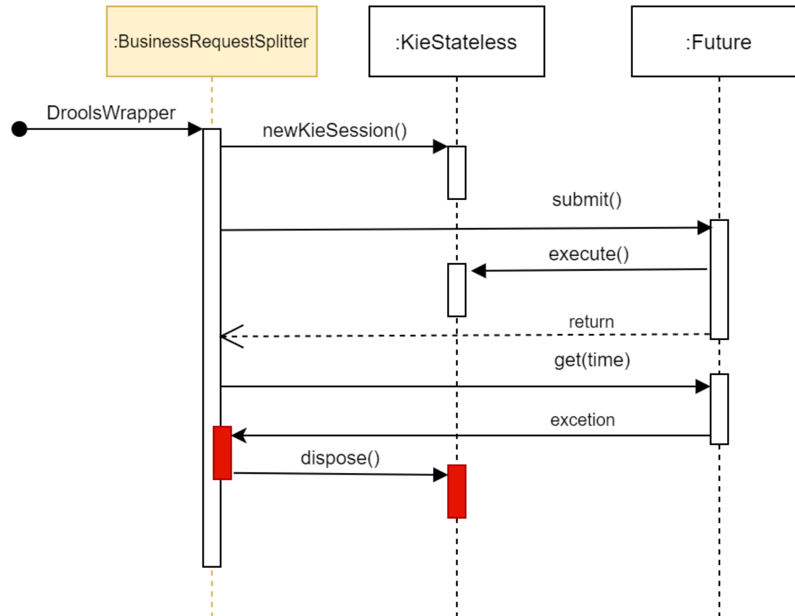


Obrázek 11.1: Sekvenční diagram pro využití mechanismu fireUntilHalt při zastavení engine

11.3. Využití mechanismu Future při zacyklení

Druhý způsob je možné použít jak pro stavovou, tak i pro bezstavovou implementaci KieSession. Je možné využít mechanismus Future[44], který představuje budoucí výsledek asynchronního výpočtu. Tento výsledek se nakonec objeví v budoucnosti po dokončení zpracování.[45] Zjednodušeně řečeno, implementace využívající mechanismus Future funguje podobně jako při použití fireUntilHalt a knihovny Awaitility. Engine je předán fakt a spustí se vyhodnocování, které je přerušeno, pokud není hotové do časového limitu. Implementace logiky za pomoci Future začíná u třídy Callable. Jedná se o rozhraní, jež má jedinou metodu *call()*, a které představuje úlohu, která vrací výsledek. Pro jednoduché případy není potřeba implementovat speciální třídy reprezentující rozhraní Callable, ale je možné využít lambda výraz. Po vytvoření je instance Callable předána třídě ExecutorService, která vrací objekt Future a postará se o spuštění úlohy v novém vlákně. Třídu ExecutorService lze naimplementovat několika způsoby. V tomto případě byla zvolena implementace SingleThreadExecutor, která je schopná zpracovávat jedno vlákno v jeden moment. Po zavolání metody *submit()* je vrácen objekt Future, se kterým je možné dále pracovat. Metoda *get()*, která je volána na objektu Future, vrací skutečný výsledek z výpočtu. Tato metoda blokuje provádění, dokud není úkol dokončen. Pro zjištění, zda je úkol dokončen, je možné zavolat funkci *isDone()*. Metoda *get()* má i přetíženou verzi, která má časový limit a časovou jednotku jako argumenty. Na rozdíl od první zmíněné funkce *get()*, druhá funkce *get()* s argumenty způsobí výjimku TimeoutException, pokud se úloha nedokončí před zadaným časovým limitem. Toho lze využít i pro KieSession. Jakmile by

došlo k zacyklení v engine, metoda *get()* vyhodí výjimku, která je odchycena a následně je zavolána funkce *dispose()* na všech instancích KieSession, která zastaví běh engine.



Obrázek 11.2: Sekvenční diagram pro využití mechanismu Future při zastavení engine

11.4. Zaznamenávání událostí při běhu engine

Pro kontrolu správného běhu engine je důležité zaznamenávat události, které se v engine dějí. K těmto účelům lze využít rozhraní KieRuntimeLogger z knihovny org.kie.api. V knihovně org.drools.core se nachází několik implementací daného rozhraní. Jedná se například o třídu WorkingMemoryFileLogger, která zaznamenává události do vybraného souboru, anebo třídu WorkingMemoryConsoleLogger, jejíž implementace vypisuje události do konzole. Obě výše zmíněné třídy dědí ze třídy WorkingMemoryLogger, jež dokáže zaznamenávat události generované pracovní pamětí. Podtřídy této třídy by měly implementovat metodu *logEventCreated()*, která jako argument očekává objekt typu LogEvent, v němž je uložena zpráva o dané události. Logika v metodě *logEventCreated()* by měla daný objekt zpracovat, například uložit záznam do souboru nebo do databáze. Jelikož systém Symphony obsahuje logovací komponentu, bylo potřeba s ní propojit arron engine a s její pomocí ukládat události, které se odehrávají v engine. Z toho důvodu byla vytvořena třída RequestSplitterKieEngineLogger, která implementuje rozhraní KieRuntimeLogger a zároveň dědí od třídy WorkingMemoryLogger. Při vytváření objektu třídy RequestSplitterKieEngineLogger mu bude předán objekt typu RequestSplitterLog, díky kterému bude možné všechny události ukládat. Pomocí proměnné maxEventsInMemory lze nastavit maximální počet událostí, které jsou drženy v lokální paměti. Jakmile je v paměti

uložen vetší počet událostí něž je hodnota v proměnné maxEventsInMemory, zavolá se metoda *logEvents()*, která všechny uložené události zpracuje, respektive odešle logovací komponenty. Metoda *logEvents()* se také volá při volání metody *close()*, jež odpojí objekt RequestSplitterKieEngineLogger od engine a ukončí tak logování. Objekt třídy RequestSplitterLog při vytvoření očekává instanci třidy LoggingService, jež je zodpovědná za komunikaci s logovací komponentou, a následně i objekt typu UserContext, ve kterém je především uloženo sessionId, díky kterému uživatelé dokážou dohledat dané záznamy o událostech.

Záznamy o událostech, které se staly v engine za bezchybového běhu vypadají následovně: po vložení faktu do engine je událost zaznamenána pomocí logu s názvem OBJECT ASSERTED a výsledku funkce *toString()*, jež se volá na objektu faktu a která umožnuje vypsání všech proměnných, které objekt obsahuje. Jakmile je spuštěno vyhodnocování pravidel, které v stavové implementaci KieSession nastává po zavolání metody *fireAllRules()*, ukládají se informace o následujících třech událostech. Nejprve se zaznamenají události s názvem ACTIVATION CREATED, které obsahují název pravidla, jež bude vykonáno, a také seznam objektů, které do pravidla vstupují. Následně pokračují dvě po sobě jdoucí události s názvy BEFORE ACTIVATION FIRED a AFTER ACTIVATION FIRED. Obě obsahují název pravidla a seznam objektů, které do pravidla vstupují. Rozdíl mezi nimi je v tom, že událost s názvem BEFORE ACTIVATION FIRED obsahuje fakta ve stavu před vyhodnocením pravidla a událost s názvem AFTER ACTIVATION FIRED obsahuje fakta ve stavu po vyhodnocení pravidla.

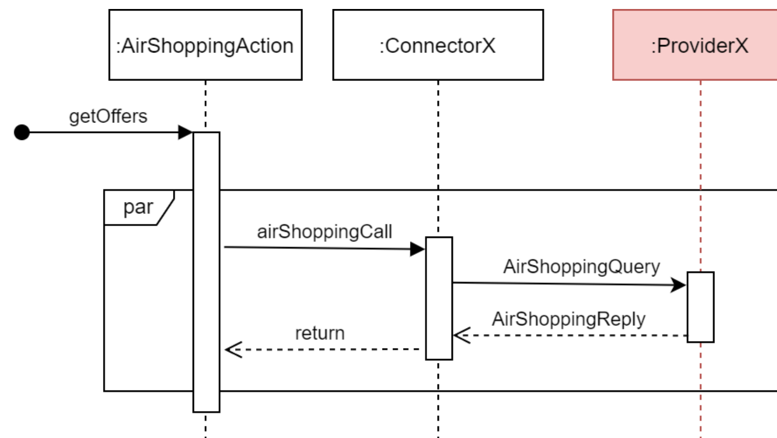
```
OBJECT ASSERTED value:DroolsWrapper{inputPars={PromoCode=present, AF=present,
BBZ=ProgramId}, resultList=null} factId: 1

ACTIVATION CREATED rule:AfBlueBiss activationId:AfBlueBiss [1] declarations:
w=DroolsWrapper{inputPars={PromoCode=present, AF=present, BBZ=ProgramId}, resultList=null}

BEFORE ACTIVATION FIRED rule:AfBlueBiss activationId:AfBlueBiss [1] declarations:
w=DroolsWrapper{inputPars={PromoCode=present, AF=present, BBZ=ProgramId}, resultList=null}

AFTER ACTIVATION FIRED rule:AfBlueBiss activationId:AfBlueBiss [1] declarations:
w=DroolsWrapper{inputPars={PromoCode=present, AF=present, BBZ=ProgramId},
```

12. Zapojení knihovny Business Request Splitter do komponenty NdcPricer

Firma AARON GROUP nabízí rezervační systém Symphony, který umožňuje prodej služeb souvisejících s cestovním ruchem. Jedná se především o prodej letenek od tradičních i nízkonákladových dopravců. Pro vyhledání letů mohou uživatelé využít několik webových aplikací, které jsou určené několika druhům uživatelů, mezi které patří korporátní či běžní zákazníci. I přes odlišnost aplikací je vyhledávací mechanismus ve své podstatě stejný ve všech aplikacích. Uživatel si zvolí odkud a kam chce letět, jestli chce hledat jednosměrnou či zpáteční letenku, a čas odletu či příletu. Dále má uživatel možnosti omezit vyhledávání na preferovaný typ cestovní třídy a na konkrétní leteckou aerolinku. Webová aplikace tato data následně předá systému Symphony, který se postará o vyhledávání NDC, GDS i LCC nabídek. O vyhledávání, nákup a úpravu NDC nabídek se primárně starají dvě komponenty – komponenta NdcPricer, která má na starost vyhledávání nabídek a komponenta NdcBooking, která zprostředkovává nákup vyhledané nabídky. Pro komunikaci s jednotlivými poskytovateli NDC obsahu byly vytvořeny knihovny, které jsou napojeny přímo na konkrétní aerolinii anebo na zprostředkovatele dat, který komunikuje s několika aeroliniemi. Těmto knihovnám se v terminologii AARON GROUP říká konektory, protože jsou učeny k převedení dat od poskytovatele na data, se kterými dokáže systém Symphony pracovat. Tato konverze samozřejmě funguje i druhým směrem. Konektory jsou používány jak v komponentě NdcPricer tak i v komponentě NdcBooking.

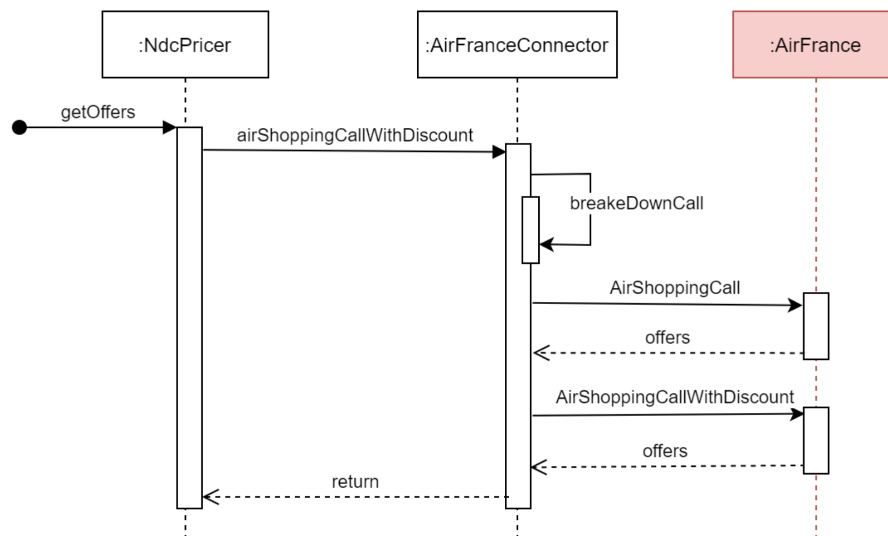
Podle IATA standardů se pro získání nabídek od poskytovatele dat využívá dotaz AirShopping, který v sobě obsahuje informace o hledaných nabídkách. Takový dotaz vznikne již ve webové aplikaci a je předán ke zpracování systému Symphony, který jej následně předá jednotlivým konektorům. V komponentě NdcPricer je na zpracování AirShopping dotazu vytvořena speciální třída AirShoppingAction, která podle konfigurace pozná, do kterých konektorů má dotaz poslat. Logika ve třídě AirShoppingAction z dotazu AirShopping nejprve vytvoří seznam objektů typu AirShoppingCallInProvider, ve kterém je uložen objekt typu NdcAirShoppingQuery, jenž obsahuje většinu informací o hledaném letu. Dále se zde nachází pomocná data jako například user kontext nebo také informace o konkrétním konektoru, kam by dotaz měl být zaslán. Následně jsou všechny objekty typu AirShoppingCallInProvider, které jsou v seznamu uloženy, předány k asynchronnímu zpracování. Jinými slovy, každý dotaz typu NdcAirShoppingQuery z objektu typu AirShoppingCallInProvider je předán danému konektoru, který dotaz zpracuje a odešle poskytovateli dat. Jelikož jsou dotazy do konektorů posílány asynchronně, výsledky také

nepřichází v jeden moment. Jakmile dorazí nabídky od poskytovatele dat, jsou předány vyšší vrstvě, která je pošle webové aplikaci. Uživateli jsou hledané nabídky stále přimícháványdo výsledného seznamu, jenž mu je zobrazen.



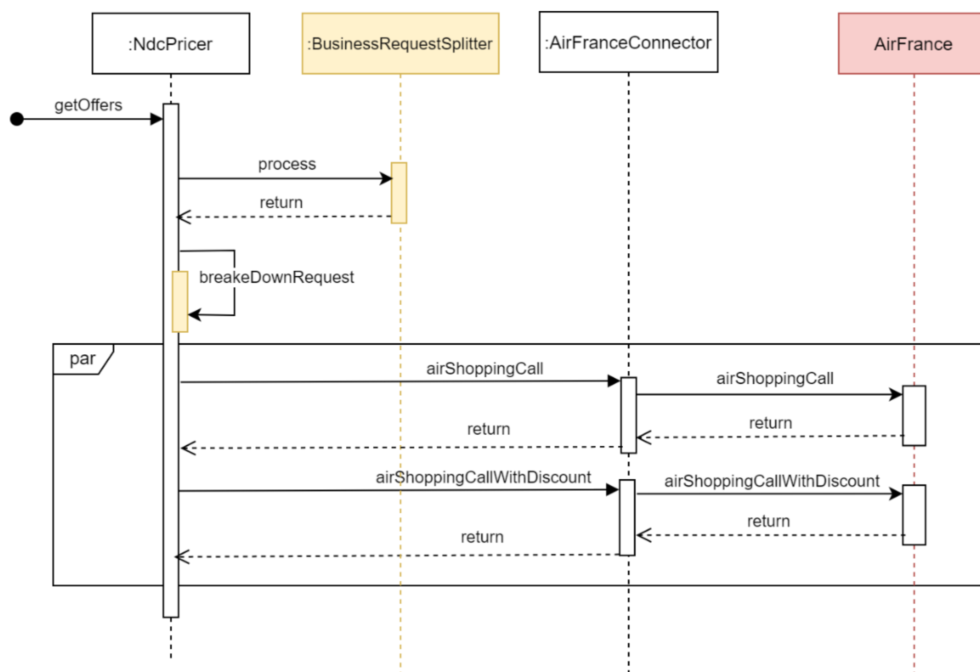
Obrázek 12.1: Sekvenční diagram komunikace s poskytovatelem dat

Na sekvenčním diagramu *Obrázek 12.1* je naznačena komunikace s poskytovatelem dat. Je zde naznačeno, jak třída AirShoppingAction paralelně posílá airShopping dotazy typu NdcAirShoppingQuery do několika konektorů, které je převedou na standardizovaný objekt, a pošlou externímu poskytovateli dat. Odpověď od poskytovatele dat je následně zpracována a odeslána vyšší vrstvě.



Obrázek 12.2: Sekvenční diagram komunikace s AirFrance bez zapojení knihovny BusinessRequestSplitter

Na sekvenčním diagramu *Obrázek 12.2* je naznačena situace před zapojením knihovny BusinessRequestSplitter. Je zde ukázán případ, kdy se do NdcPriceru dostane dotaz určený pro AirFrance, který v sobě obsahuje parametr Discount. Jakmile se tento dotaz dostane do konektoru určenému pro komunikaci s AirFrance, je dotaz rozdělen na dva, které jsou následně postupně poslány do AirFrance.



Obrázek 12.3: Sekvenční diagram komunikace s AirFrance po zapojení knihovny BusinessRequestSplitter

Na sekvenčním diagramu *Obrázek 12.3* je naznačena situace po zapojení knihovny BusinessRequestSplitter. Je zde znovu zachycen stejný případ z diagramu BB – průběh zasílání dotazu obsahuje parametr Discount do AirFrance. Zde je vidět, že funkce, která zajištovala rozpad dotazu v konektoru AF, byla odstraněna. Dotazy jsou nově rozpadány před tím, než jsou paralelně zasílány do konektorů. V nové implementaci proto dojde k rozpadu dotazu pro AirFrance obsahující parametr Discount již v komponentě NdcPrice. Rozpadnuté dotazy jsou následně paralelně odeslány do konektoru AF.

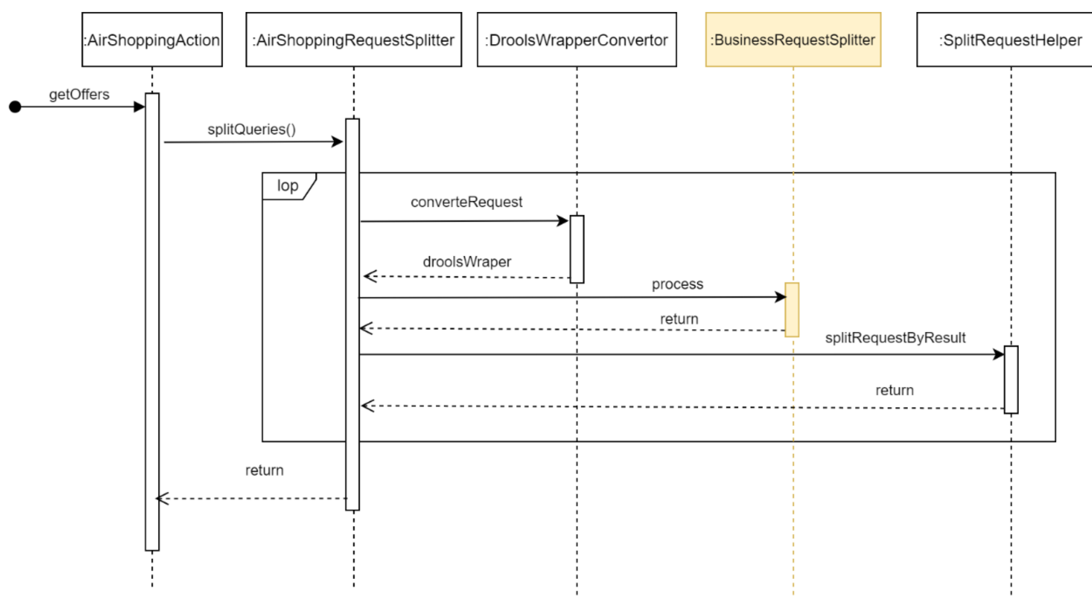
Jelikož ve třídě AirShoppingAction je přítomen seznam objektů typu AirShoppingCallInProvider, je to nejvhodnější místo, kam zapojit knihovnu BusinessRequestSplitter, která nabízí servise RequestSplitterController, s jehož pomocí se dá rozhodnout, zda má být daný dotaz rozpadnut. Proto byla v komponentě NdcPricer vytvořena třída AirShoppingRequestSplitter, která ve svém konstruktoru vytvoří instanci servise RequestSplitterController, přičemž dojde i k sestavení rule engine.

Komponenta NdcPricer je připojena k databázi *db_ndcpricer*, ve které jsou uloženy pravidla v tabulce *drl_file*. Pro získávání a ukládání dat do konkrétní tabulky se v komponentě používá rozhraní DAO[31]. Pro každou tabulku bylo vytvořeno speciální rozhraní, s jehož pomocí lze do dané tabulky ukládat, popřípadě z ní získávat data. I pro tabulku *drl_file* bylo vytvořeno rozhraní DrlFileDao, které poskytuje metodu *getDrlFiles()* pro získání listu objektu typu DrlFile, ve kterém jsou dvě proměnné typu String – název pravidla a pravidlo samotné.

Jelikož vytvoření Rule engine trvá nezanedbatelnou dobu, je instance AirShoppingRequestSplitter vytvořena v konstruktoru třídy AirShoppingAction. Pro vytvoření Rule engine je potřeba seznam pravidel, které jsou uloženy v databázi. Při vytváření instance AirShoppingAction je tedy použito DrlFileDao k získání listu pravidel, jenž je následně předán konstruktoru ve třídě QueryDecayController.

Třída AirShoppingRequestSplitter poskytuje metodu *splitRequests()*, která jako parametr očekává seznam objektů typu AirShoppingCallInProvider. Zjednodušeně řečeno, pokud v seznamu objeví objekt, který je nutné rozdělit na více objektů, rozdělí jej, a tím vytvoří nové objekty, které do seznamu vloží. Na začátku metody *splitRequests()* se zkontroluje, že seznam není prázdný a následně se vytvoří dva seznamy. Seznam s názvem newRequests pro nově vytvořené objekty, které je potřeba do hlavního seznamu přidat, a seznam s názvem requestsToRemove pro objekty, které je ze seznamu potřeba odstranit. Následně je postupně každý objektu typu AirShoppingCallInProvider v seznamu převeden na objekt typu DroolsWrapper, který je předán servise RequestSplitterController, jež pomocí business pravidel rozhodne, jestli má být daný objekt rozpadnut, popřípadě podle jakých parametrů. Jako odpověď ze servise RequestSplitterController přijde kolekce typu Set s názvem *resultSet*, která obsahuje objekty typu String, podle kterých se rozhodne o rozpadu daného dotazu. Za předpokladu, že resultSet není prázdný, je předán společně s objektem typu AirShoppingCallInProvider funkci *splitRequestByResult()*, jež zařídí případný rozpad objektu, respektive naplnění seznamů newRequests a requestsToRemove. Na konci metody jsou z hlavního seznamu odstraněny všechny objekty v seznamu requestsToRemove a přidány všechny objekty ze seznamu newRequests. Na diagramu *Obrázek 12.4* je zachycen výše popsaný průchod funkcí *splitRequests()* ve třídě AirShoppingRequestSplitter.

[31] DAO - Data Access Object



Obrázek 12.4: Sekvenční diagram zachycující použití knihovny BusinessRequestSplitter

Pro jednotlivé možnosti klíčových slov, které je možné použít po definování rozpadu objektu typu AirShoppingCallInProvider, byl vytvořen enum s názvem BreakChoice. Ten je použit jak při konverzi objektu typu AirShoppingCallInProvider na objekt typu DroolsWrapper, tak v logice, která se stará o samotný rozpad objektů. Enum s názvem BreakChoice by se dal použít i pro tvorbu samotných pravidel, ale to by obnášelo několik omezení. V první řadě se předpokládá, že knihovna BusinessRequestSplitter bude použita i v jiné aplikaci pro řešení jiného problému. Kdyby byl v knihovně použit tento enum, došlo by k zavlečení funkcionality řešící konkrétní problém do společné logiky. Dále se předpokládá, že možnosti rozpadu budou přibývat, a tak se bude zvětšovat i enum BreakChoice. To ovšem zase přináší problém udržování jedné aktuální verze enum ve více projektech. Proto se enum BreakChoice používá pouze v komponentě NdcPricer k zpřehlednění kódu.

Pro konverzi dotazu typu AirShoppingCallInProvider na objekt typu DroolsWrapper je určena třída DroolsWrapperConvertor, která poskytuje metodu *convertRequest()*. V ní je implementována logika, která detekuje přítomnost konkrétních dat v dotazu. Jakmile detekuje přítomnost určitých dat v dotazu, uloží tuto skutečnost do proměnné inputPars v nově vytvořeném objektu DroolsWrapper, který je následně vrácen jako odpověď. Proměnná inputPars je typu Map - data se do ní vkládají tak, že daný klíč odkazuje na danou hodnotu. Například je nejprve detekován seznam kódů aerolinek, pro které je dotaz určen. Kódy daných aerolinek jsou následně postupně vloženy do proměnné inputPars tím způsobem, že kód aerolinie tvoří klíč a hodnotu tvoří univerzální slovo „Present“, které je použito v případech, kdy není potřeba využít možnosti uložení nějaké proměnné do hodnoty, na kterou se daným klíčem odkazuje. Dále se kontroluje, zda objekt obsahuje preferenci

na konkrétní typ cestovní kabiny. Uživatel si může vybrat i několik typů cestovní kabiny, které následně v dotazu přijdou v kolekci typu List. Pokud je daný seznam v dotazu přítomen, je převeden na textový řetězec a uložen do proměnné inputPars tím způsobem, že klíč je tvořen slovem „Cabin" a hodnotu reprezentuje daný textový řetězec. Třída DroolsWrapper obsahuje metodu *is()*, která na vstupu požaduje dva parametry – klíč a hodnotu. Při zavolání metody se implementovaná logika pokusí nalézt hodnotu pro daný klíč. Pokud se jí to podaří a nalezne výsledek pro daný klíč, zkontroluje, zda výsledek neobsahuje více hodnot. Pokud ano, pokusí se v nich nalézt hodnotu shodnou se zadanou hodnotou v parametru funkce. Metoda *is()* následně vrací kladnou hodnotu, pokud pod daným klíčem našla zadanou hodnotu, a zápornou pokud nikoliv. Díky tomu lze vytvořit pravidlo, které se může v podmínkové části doptávat na konkrétní typ cestovní kabiny.

Odpověď, která přijde z knihovny BusinessRequestSplitter, je následně zpracována pomocí třídy SplitRequestHelper, v níž dojde k rozpadu daného dotazu, respektive k vytvoření nových objektů typu AirShoppingCallInProvider. Pří vytváření instance třídy SplitRequestHelper konstruktor požaduje dva parametry, instanci již zmíněných seznamů newRequests a requestsToRemove. Po získání odpovědi z knihovny BusinessRequestSplitter je zavolána funkce *splitRequestByResult()*, která vyžaduje na vstupu danou odpověď, respektive kolekci obsahující objekty typu String a aktuální dotaz, pro který byl daný výsledek vytvořen. Funkce v první řadě převede daný výsledek na seznam objektů typu BreakChoice s názvem breakChoices. Jestliže seznam breakChoices není prázdný, je objekt reprezentující aktuální dotaz přidán do seznamu s objekty k odstranění. Následně je z dotazu vytvořen nový objekt typu AirShoppingCallInProvider, který neobsahuje žádnou ze specifikovaných proměnných ze seznamu breakChoices. Z něj se následně pomocí metody *clone()* získávají duplikáty, do kterých jsou postupně přidávány proměnné z dotazu, které jsou specifikovány v seznamu breakChoices. Pro příklad je uvedena situace, kdy do funkce *splitRequestByResult)* přijde dotaz obsahující preference na dva typy cestovní kabiny a ve výsledku z knihovny BusinessRequestSplitter se bude nacházet klíčové slovo *Cabin*. Ve funkci *splitRequestByResult()* bude vytvořen objekt, který nebude obsahovat žádnou preferenci na cestovní typ kabiny, ale jinak bude obsahovat stejná data jako objekt reprezentující daný dotaz. Následně z něj budou vytvořeny dva duplikáty, do kterých přijdou jednotlivé preference na typy cestovních kabin. Ve výsledku tak bude v seznamu obsahujícím objekty k odstranění jeden objekt reprezentující vstupní dotaz a v seznamu s nově vytvořenými objekty budou tři nové dotazy – dotaz bez preference na typ cestovní kabiny, dotaz s první preferovanou cestovní kabinou a dotaz s druhou preferovanou cestovní kabinou.

13. Testování

Testování je důležitá část pří vývoji jakékoliv aplikace. Slouží nejen k odhalení chyb, které se mohou v systému vyskytovat, ale také může přinést nové nápady ke zlepšení. Systém pro rozhodování o rozpadu dotazů byl otestován několika způsoby. Logika implementovaná v knihovnách a v komponentách byla testována pomocí jednotkových testů, díky kterým bylo možné odladit jednotlivé funkcionality v systému. Celková funkčnost systému jako celku byla testována ve vývojovém prostředí, respektive všechny vyvíjené aplikace byly nasazeny na server určený pro vývoj. Zde bylo testováno nejen vytváření pravidel pomocí aplikace Business Central, ale také správné vyhodnocování a rozpad dotazů při vyhledávání leteckých nabídek. Byly využity i integrační testy, které ověřily, že zapojení nové logiky do systému nezpůsobilo chybné fungování daného systému. Funkcionalita byla také otestována pomocí uživatelských testů, respektive byla otestován zaměstnankyní, která pracuje ve firmě AARON GROUP.

13.1. Jednotkové testy

Při psaní jakéhokoliv kódu je velmi snadné nějakou chybu přehlédnout, popřípadě nedomyslet chování vyvíjené funkcionality. Z těchto důvodů je vhodné napsaný kód co nejdříve otestovat – k tomu se dají využít jednotkové testy, které jsou vhodné pro odhalení zásadních chyb, popřípadě pro vývoj samotné funkcionality. Pro psaní jednotkových testů v programovacím jazyku Java se používá framework JUnit [46]. Díky těmto testům byla otestována například konverze objektu třídy AirShoppingCallInProvider na objekt třídy DroolsWrapper v komponentě NdcPricer. Dále pak byla otestována i třída SplitRequestHelper, která slouží pro rozpad dotazu na více objektů. Pro ověření celkové nově implementované logiky v komponentě NdcPricer, byla vytvořena třída AirShoppingRequwstSplitterTest, ve které se nachází několik testů, jež dokážou ověřit správné chování implementované logiky. Jelikož je v konstruktoru AirShoppingRequestSplitter využito DrlFileDao pro získání pravidel z databáze, byla využita technika známá jako „mocking“ k simulaci dané servise. Pomocí knihovny JMockit [47] lze vytvořit falešný objekt DrlFileDao, který simuluje chování třídy DrlFileDao. Díky tomu lze „namokovanému“ objektu nastavit, jaký výsledek má volaná funkce vracet, aniž by byla spuštěna vnitřní logika dané funkce. Při využití JMockit je potřeba na začátku testu specifikovat, jaké metody budou volány na objektu typu DrlFileDao a jaké výsledky mají dané metody vracet.

Podobným způsobem byla otestována i aplikace Bridge a knihovna BusinessRequestSplitter, ve které je otestován i samotný rule engine. Bylo potřeba otestovat i mechanismus Future v knihovně BusinessRequestSplitter, který je použit pro zastavení příliš dlouhého vyhodnocování pravidel. Proto byla vytvořena testovací třída LongRunTest obsahující test, ve kterém je do rule engine vloženo jedno pravidlo, jenž na vstupu očekává fakt typu DroolsWraper. Důležitá logika je v akční části pravidla, jelikož je zde volána metoda *sleep()*, která umožňuje pozastavit vykonávání daného vlákna na určitou dobu. Před uspáním se do proměnné *resultSet* v daném faktu uloží slovo „StartSleep“ a po opětovném spuštění se uloží slovo „Done“. Při vyhodnocování pravidel pak dojde ve třídě RequestSplitterController k výjimce a vykonávání rule engine je zastaveno. Na konci testu se zkontroluje, že v proměnné *resultSet* v daném faktu se vyskytuje pouze slovo „StartSleep“ a tím je ověřeno, že došlo k zastavení rule engine. V knihovně ArronEngine se vyskytují i další testy, které prověřují správné chování systému a ověřují, zda jsou pravidla správně vyhodnocena.

13.2. Integrační testy

K testování celkové funkčnosti vyvíjených systémů bylo ve společnosti AARON GROUP vytvořeno mnoho integračních testů, které testují průchod celým systémem. S jejich použitím je možné otestovat, zda nově implementovaná logika nezpůsobí chybu v jiné části systému. Jednotlivé testy simulují kroky uživatele, který si kupuje danou letenku. Pro otestování správného zapojení knihovny AirShoppingRequestSplitter do komponenty NdcPricer byly použity integrační testy, které testují kup NDC nabídek. Sada testů pro NDC nabídky obsahuje testy, které byly napsány tak, aby testovaly jednotlivé konvertory. S jejich pomocí bylo otestováno, že zapojení knihovny AirShoppingRequestSplitter do komponenty NdcPricer nezpůsobilo žádnou chybu v jiné části systému.

13.3. Uživatelské testování

Aby bylo ověřeno, že byla funkcionalita dobře navrhnuta a naimplementována, bylo ji potřeba celkově otestovat. Proto byla upravená komponenta NdcPricer zapojena do systému Symphony, který byl následně nasazen a spuštěn na vývojovém prostředí. Aplikace Business Central byla spuštěna pomocí aplikace Docker [48], která umožnuje jednoduché spuštění aplikace v kontejneru. V něm je spuštěna i aplikace Bridge, která je připojená k vývojové databázi, k níž je připojen i systém Symphony. Testování probíhalo v průběhu vývoje celkového řešení a testovalo se především, zda je aplikace Business Central vhodná pro vytváření business pravidel, dále pak jestli aplikace Bridge dokáže vytvořená,

popřípadě upravená, pravidla ukládat do databáze. V neposlední řadě byl testován celkový průchod systémem Symphony, respektive bylo kontrolováno podle zalogovaných událostí, zda při vyhledávání leteckých nabídek dochází ke správnému rozpadu jednotlivých dotazů.

Jakmile bylo celkové řešení naimplementováno a otestováno, došlo k uživatelským testům, kterých se zúčastnila zaměstnankyně firmy AARON GROUP, která nemá žádné znalosti jakéhokoliv programovacího jazyka, a která bude v budoucnu daná pravidla vytvářet. Před samotným testováním jí bylo vysvětleno, jaký problém se snaží nová funkcionalita vyřešit. Následně jí byla představena aplikace Business Central, aby pochopila, jakým způsobem je možné pravidla vytvářet. Dále měla k dispozici seznam jednotlivých klíčových slov, které nyní lze zadat do objektu typu DroolsWrapper a poté mohla testovat.

Zaměstnankyně firmy AARON GROUP měla za úkol vytvořit jedno lehčí a jedno složitější pravidlo. Dostala níže uvedený text, který popisoval, čeho by daná pravidla měla dosáhnout. Následně obě pravidla vytvořila a zkontrolovala, že se obě uložila do databáze. V poslední řadě otestovala funkčnost pravidel pomocí webové aplikace Hub, která je určena pro vyhledání, respektive pro nákup letenek. V ní upravila vyhledávání tak, aby se do systému Symphony poslal dotaz obsahující parametry, na které daná pravidla zareagují. Po vyhledání nabídek si zobrazila záznamy událostí, ke kterým v systému Symphony došlo, a podle nich ověřila, že rozpad dotazů proběhl správně.

Zadání pro vytvoření pravidel s BaCabin a SqAfDiscount

1. Úkolem je vytvořit pravidlo BaCabin, které bude mít následující význam: pokud se jedná o dotaz pro dopravce British Airways, a v dotazu je specifikován jakýkoliv typ kabiny, je požadováno, aby se dotaz rozpadnul na dva – jeden bez definice typu kabiny a druhý s definicí typu kabiny.

2. Úkolem je vytvořit pravidlo SqAfDiscount, které bude mít následující význam: pokud se jedná o dotaz pro dopravce Singapore Airways nebo pro dopravce Finnair a zároveň se v dotazu vyskytuje Discount, který v sobě má FareType nastaven na TOUR_OPERATOR, dotaz se rozpadne na dva – na jeden bez proměnné typu Discount a druhý s proměnnou typu Discount.

Výsledek prvního zadání je zachycen na *Obrázek 10.1* a *Obrázek 10.2*. Výsledek druhého zadání je následně zachycen na *Obrázek 13.2* a *Obrázek 13.3*.

Parametry použitelné v podmínce

IATA Airline code – Kód aerolinky podle standardu IATA – například „AF“ nebo „BA“.

Discount – neboli Corporate Discount – Obsahuje FareType, identifikační kód a kód aerolinie, ke které se vztahuje. Pokud je zadán a poslán poskytovateli dat, tak se vrátí speciální nabídky pro korporátní firmy či cestovní agentury.

FareType – může mít být v následujících tvarech: PUBLIC, PROMO, CORPORATE, TOUR_OPERATOR, VISIT_FRIENDS_AND_RELATIVES, CONSOLIDATOR, EMIGRANT

FrequentFlyer – Jedná se o Frequent flyer program neboli o věrnostní kartu zákazníka.

CorporateBonus – Zde se jedná o Korporátní bonus program, ke kterému se vztahuje i konkrétní ProgramId.

ProgramId – obsahu každý korporátní bonus program.

Cabin – specifikuje rozpad podle typu cestovní kabiny. - FIRST, BUSINESS, ECONOMY_ALL, PREMIUM_ECONOMY, ECONOMY, ECONOMY_DISCOUNTED, ALL

Parametry použitelné ve výsledku

IATA Airline code – kód aerolinky podle IATA

OnlySpecific – Definuje skutečnost, že nebude volán dotaz bez speciálních dat.

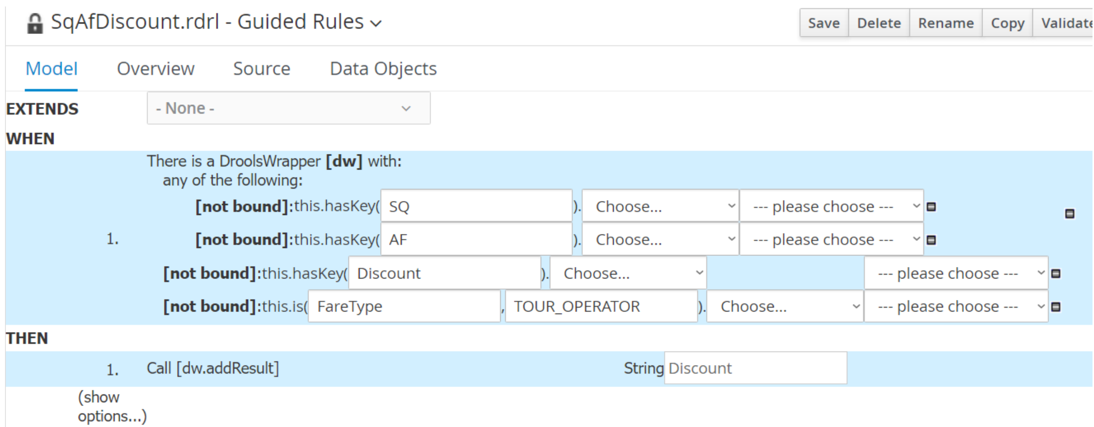
Cabin – Dotaz bude rozdělen podle typu kabin.

Discount – Dotaz bude rozdělen podle typu speciálních nabídek.

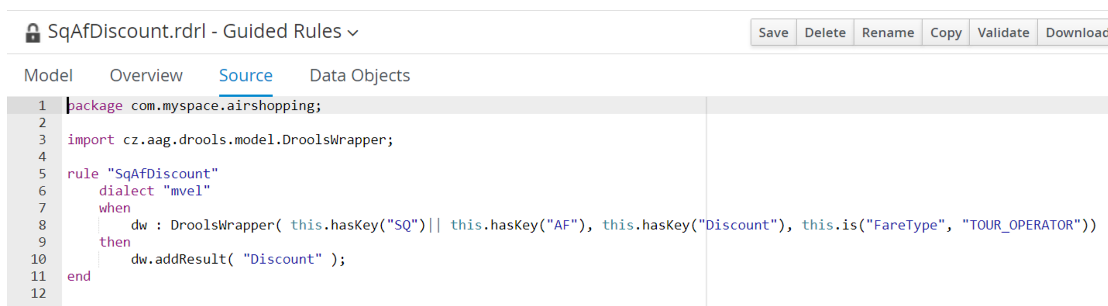
FrequentFlyer – Dotaz bude rozdělen podle Frequent flyer programu.

CorporateBonus – Dotaz bude rozdělen podle Corporate Bonus Programmes.

Obrázek 13.1: Seznam klíčových slov, které lze požít v při vytváření business pravidel



Obrázek 13.2: Ukázka z aplikace Business Central, kde jde vidět vytváření pravidel ze zadání č.2



Obrázek 13.3: Ukázka z aplikace Business Central, kde je vidět výsledný rdl file pro zadání č.2

Testující s vytvořením prvního pravidla neměla žádné problémy, ale u vytváření druhého pravila se musela doptat na to, kde najde možnost „nebo“. Po vytvoření obou pravidel otestovala, zda se pravidla přidala do databáze a jestli jsou obě pravila funkční. Poté celé testování zhodnotila. Aplikace Business Central jí přišla intuitivní a dokázala se v ní dobře orientovat. Samotné vytváření pravidel jí přišlo trochu složité, ale předpokládá, že po napsání několika pravidel již s jejich vytvářením nebude mít žádné problémy. Vznesla požadavek na vytvoření manuálu, ve kterém by byl přesný popis tvorby pravidel pomocí aplikace Business Central, aby se do něj v případě potřeby mohla podívat.

13.4. Shrnutí

Veškerá naimplementovaná logika byla otestována jednotkovými a integračními testy i pomocí uživatelského testování. Při testování bylo objeveno několik chyb, které byly odstraněny. Například byla objevena chyba, která způsobovala pomalejší vyhledávání nabídek. Tato chyba byla nalezena v knihovně BusinessRequestSplitter a byla způsobena tím, že byl rule engine vytvářen při každém volání dotazu na vyhledávání nabídek. Po zjištění byl problém ihned vyřešen přesunutím vytváření rule engine do konstruktoru třídy RequestSplitterControllerImpl.

Uživatelské testování prokázalo, že pomocí aplikace Business Central může vytvářet pravidla i uživatel bez znalosti jakéhokoliv programovacího jazyka. Dále ukázalo, že orientace v aplikaci Business Central je intuitivní, a uživatelé nebudou mít problémy s jejím používáním. Také přineslo podnět k vytvoření krátké příručky, ve které bude popsán proces vytváření pravidel. Daná příručka nemusí sloužit pouze jako tahák při vytváření pravidel, ale mohla by novým zaměstnancům pomoct při učení vytváření pravidel.

14. Závěr

V diplomové práci bylo navrženo, implementováno a otestováno řešení, které je schopno řídit počty transakcí pro získávání dat od poskytovatele, jež není schopen zaslat v jednom dotazu. Funkčnost řešení byla otestována tím, že bylo zapojeno do již existujícího rezervačního systému, kde je použito pro řízení počtu dotazů, jež jsou zasílány poskytovatelům dat. Při návrhu a implementaci byl kladen důraz na to, aby vyvinuté řešení bylo možné využít i v jiném systému pro řešení podobného problému. Systém byl také vyvinut tak, aby jej dokázal nakonfigurovat i uživatel bez znalosti jakéhokoliv programovacího jazyka.

Při vývoji byly nejprve prozkoumány jednotlivé možnosti řešení. Aby bylo možné dané řešení jednoduše zapojit i do jiného systému, který byl napsán v jazyce Java, byl zvolen implementační přístup založený na vytvoření nové knihovny, která bude řídit počty transakcí. Následně byly analyzovány přístupy, jakými lze danou knihovnu vytvořit, a z navrhovaných způsobů byl zvolen přístup pomocí systému založeném na business pravidlech. Díky tomuto přístupu je rozhodovací logika oddělena od systému a je ji možné změnit bez nutnosti zásahu do kódu daného systému. Jelikož standardů a systémů založených na pravidlech existuje mnoho, bylo nutné z nich vybrat ten nejvhodnější. Po důkladné úvaze byl zvolen systém Drools, který je stále vyvíjen a vylepšován. Poskytuje rule engine, který se dá jednoduše zapojit do jakéhokoliv systému napsaném v jazyce Java. Pro tvorbu pravidel, která mohou být zpracována pomocí rule engine od Drools, je možné využít aplikaci Business Central. Velký důraz byl kladen na to, aby danou konfiguraci dokázal vytvářet i uživatel, který nemá zkušenosti s žádným programovacím jazykem. Daný požadavek byl dodržen právě díky aplikaci Business Central, která poskytuje uživatelské rozhraní pro tvorbu business pravidel.

Po analýze dostupných řešení byl vytvořen návrh, který počítal s vytvořením knihovny obsahující rule engine, jež by podle business pravidel rozhodovala o řízení počtu transakcí. Tato knihovna by měla být zapojena do již existujícího systému Symphony. Pro vytváření business pravidel byla využita aplikace Business Central a pro ukládání pravidel do databáze byla vytvořena aplikace nová. Před samotnou implementací byl systém Drools prozkoumán a popsán. Byl představen základní algoritmus, na kterém rule engine stojí, dále pak byla popsána syntaxe business pravidel, které rule engine zpracovává, a nakonec byla představena aplikace Business Central, ve které se pravidla budou vytvářet.

Při implementaci byl kladen důraz především na to, aby bylo možné dané řešení použít i v jiném systému pro řešení podobného problému. Proto byla i nově vytvořená knihovna BusinessRequestSplitter navržena a naimplementována tak, aby ji bylo možné

bez jakýchkoliv úprav zapojit i do jiného systému. To je také dáno díky univerzálnímu objektu DroolsWrapper, který je navržen tak, aby do něj šlo přenést data, podle kterých se mají dané business pravidla rozhodovat. Po zapojení knihovny BusinessRequestSplitter je potřeba do daného systému doimplementovat logiku, která bude převádět data z objektu, ve kterém jsou data uložena, na objekt DroolsWrapper. Následně je nutno také doimplementovat logiku, která výsledky vrácené z knihovny BusinessRequestSplitter vyhodnotí a na jejich základě provede požadované změny. Tímto způsobem byla knihovna zapojena do systému Symphony, respektive do komponenty NdcPrice, kde slouží k řízení počtu transakcí pro získávání dat od poskytovatele.

Aplikace Business Central, jež slouží pro vytváření business pravidel, pro ukládání dat využívá Git. Proto byla vytvořena aplikace Bridge, která pomocí Git transakcí dokáže získat pravidla z Git repozitáře, a následně je uložit do databáze. Aby se aplikace Nridge nemusela manuálně spouštět při každé změně pravidla, byla využita funkcionalita Git hook tak, že jakmile dojde k úpravě jakéhokoliv pravidla v aplikaci Business Central, spustí se aplikace Bridge a upravené, popřípadě nově vytvořené, pravidlo stáhne a uloží do databáze. Podobně jako vytvořenou knihovnu BusinessRequestSplitter i aplikaci Bridge je možné použít v jiném systému pro řešení podobného problému. Po nastavení správné konfigurace a drobné úpravě v kódu je možné ji použít tak, aby z určitého Git repozitáře stahovala data a ukládala je do nadefinované databáze.

Systém byl celkově otestován, jak pomocí jednotkových a integračních testů, tak i pomocí uživatelského testování, které prokázalo funkčnost systému, a také ověřilo, že pomocí aplikace Business Central může business pravidla vytvářet i uživatel bez znalosti jakéhokoliv programovacího jazyka. Testování pomohlo odhalit několik chyb v systému, které byly odstraněny a také přineslo několik podnětů pro další zlepšení. Například bude nutno vytvořit krátkou příručku, ve které by byl popsán postup vytváření pravidel pomocí aplikace Business Central. V dalším vývoji bude upraveno logování v knihovně BusinessRequestSplitter i v aplikaci Bridge tak, aby výsledné záznamy o událostech byly přehlednější. Tabulka *drl_file* se rozšíří o nové sloupce, které budou obsahovat další informace o pravidlu. Při vytváření pravidel v aplikaci Business Central je možné ukládat nejen samotné pravidlo, ale i čas, kdy pravidlo bylo změněno, nebo kdo pravidlo vytvořil. Dané informace pro tuto práci nebyly důležité, ale v budoucím využití by mohly být užitečné. Je plánováno také využití knihovny BusinessRequestSplitter v rezervačním systému společnosti AARON GROUP, který zprostředkovává prodej hotelových a jiných ubytovacích nabídek.

15. Literatura

1. **AARON GROUP.** *AARON GROUP* [online]. [cit. 2022-05-29]. Dostupné z: https://www.aarongroup.net/

2. **Global distribution system.** In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2022-05-29]. Dostupné z: https://en.wikipedia.org/wiki/Global_distribution_system

3. **IATA.** *IATA* [online]. [cit. 2023-01-08]. Dostupné z: https://www.iata.org/

4. **HOFFMANNOVÁ, Bc. Dominika. Distribuce a prodej letenek v současné letecké dopravě.** 2021. Diplomová práce. Vysoká škola obchodní v Praze, o.p.s. Vedoucí práce PhDr. Jean-Jacques Radosa.

5. **Low cost airline channels.** *Travel Daily* [online]. 2019 [cit. 2022-05-29]. Dostupné z: https://www.traveldailymedia.com/low-cost-airline-channels/

6. **SCHILDT, Herbert.** *Mistrovství Java*. Computer Press, 2014, 1224 s. ISBN 978-80-251-4145-8.

7. **Interpretovaný jazyk.** In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2022-05-29]. Dostupné z: https://cs.wikipedia.org/wiki/Interpretovan%C3%BD_jazyk

8. **Rule-based system.** In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2022-05-29]. Dostupné z: https://en.wikipedia.org/wiki/Rule-based_system

9. **Business rules**. *IBM* [online]. [cit. 2023-01-09]. Dostupné z: https://www.ibm.com/topics/business-rules

10. **VOJÍŘ, Stanislav. Výběr specifikace business rules pro praktické nasazení v podnikové informatice** [online]. 2015 [cit. 2022-05-29]. Dostupné z: https://www.researchgate.net/publication/286256277_Vyber_specifikace_business_rules_pro_prakticke_nasazeni_v_podnikove_informatice. Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky.

11. **FORGY, Charles.** Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. *Artificial Intelligence*. 1982, str. 17-37.

12. **Drools.** *Drools* [online]. [cit. 2022-09-25]. Dostupné z: https://www.drools.org/

13. **Apache Jena.** *Apache Jena* [online]. c2011-2023 [cit. 2023-01-08]. Dostupné z: https://jena.apache.org/index.html

14. **Jess Rule Engine and JSR 94**. *Baeldung* [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://www.baeldung.com/java-rule-engine-jess-jsr-94

15. **OpenL Tablets.** *OpenL Tablets* [online]. c2004-2022 [cit. 2022-05-29]. Dostupné z: http://openl-tablets.org/

16. **Microsoft Excel.** *Microsoft* [online]. c2023 [cit. 2023-01-08]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/excel?rtc=1

17. **Microsoft Word.** *Microsoft* [online]. c2023 [cit. 2023-01-08]. Dostupné z: https://www.microsoft.com/cs-cz/microsoft-365/word?rtc=1&activetab=tabs%3afaqheaderregion3

18. **Apache Kafka.** *Kafka* [online]. c2023 [cit. 2023-01-08]. Dostupné z: https://kafka.apache.org/

19. **Git--fast-version-control.** *Git* [online]. [cit. 2023-01-08]. Dostupné z: https://git-scm.com/

20. **Easy Rules.** *GitHub* [online]. [cit. 2022-05-29]. Dostupné z: https://GIThub.com/j-easy/easy-rules

21. **Jess Language Basics.** *Alvarestech* [online]. [cit. 2022-05-29]. Dostupné z: http://alvarestech.com/temp/fuzzyjess/Jess60/Jess70b7/docs/basics.html

22. **Decision Intelligence Framework.** *Open Rules Decision Manager* [online]. 2022 [cit. 2022-05-29]. Dostupné z: https://openrulesdecisionmanager.com/

23. **Thinking Machines Corporation**. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2022 [cit. 2023-01-08]. Dostupné z: https://en.wikipedia.org/wiki/Thinking_Machines_Corporation

24. **Business Rules Management Systems 101.** *IBM* [online]. 2021 [cit. 2023-01-08]. Dostupné z: https://www.ibm.com/cloud/blog/business-rules-management-systems-101

25. **JBPM Documentation.** *JBPM Documentation* [online]. 2017 [cit. 2023-01-08]. Dostupné z: https://www.ibm.com/cloud/blog/business-rules-management-systems-101

26. **OptaPlanner.** *OptaPlanner* [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://www.optaplanner.org/

27. **Why use a Rule Engine?.** *Jboss* [online]. [cit. 2023-01-08]. Dostupné z: https://docs.jboss.org/drools/release/6.0.0.CR1/drools-expert-docs/html/ch01.html#d0e384

28. **Prolog** [online]. [cit. 2023-01-08]. Dostupné z: https://www.kiv.zcu.cz/studies/predmety/uzi/Folie2_Prolog/Prolog.pdf

29. **ŽÁKOVÁ, Bc. Iva.** *Drools Fusion and Utilization of Complex Event Processing in Web Applications*. 2013. Diplomová práce. MASARYK UNIVERSITY FACULTY OF INFORMATICS. Vedoucí práce Doc. RNDr. Tomáš Pitner, Ph.D.

30. **Decision engine in Red Hat Decision Manager.** *Red Hat Customer Portal* [online]. c2023 [cit. 2023-01-08]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_decision_manager/7.4/html/decision_engine_in_red_hat_decision_manager/index

31. **SALATINO, Mauricio a Esteban ALIVERTI.** *Mastering JBoss Drools 6*. Packt Publishing Limited, 2016, 330 s. ISBN 9781783288625.

32. **Integration with Aries Blueprint.** *Jboss* [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://docs.jboss.org/drools/release/7.73.0.Final/drools-docs/html_single/#_ch.kie.blueprint

33. **SILVERMAN, Richard.** *Git Pocket Guide: A Working Introduction*. O'Reilly Media, 2013, 231 s. ISBN 9781449325862.

34. **What is Git?.** *Microsoft* [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://learn.microsoft.com/en-us/devops/develop/git/what-is-git

35. **Know Business Central: Authoring, business monitoring, and more.** *Karin Avarela* [online]. 2020 [cit. 2023-01-08]. Dostupné z: https://karinavarela.me/2020/05/13/business-central-authoring-business-monitoring-and-more/

36. **Business Central Storage.** *Alex Porcelli* [online]. 2018 [cit. 2023-01-08]. Dostupné z: https://porcelli.me/rhba/business-central/git/intro/2018/10/26/business-central-git-intro.html

37. **Eclipse JGit™.** *Eclipse* [online]. [cit. 2023-01-08]. Dostupné z: https://www.eclipse.org/jgit/

38. **JGit - Tutorial.** *Vogella* [online]. 2019 [cit. 2023-01-08]. Dostupné z: https://www.vogella.com/tutorials/JGit/article.html

39. **What's the Difference? Creating Diffs with JGit.** *Code Affine* [online]. 2016 [cit. 2023-01-08]. Dostupné z: https://www.codeaffine.com/2016/06/16/jgit-diff/

40. **Customizing Git - Git Hooks.** *Git* [online]. [cit. 2023-01-08]. Dostupné z: https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

41**. Chapter 19. Git hooks and remote Git repository integration.** *Red Hat Customer Portal* [online]. c2023 [cit. 2023-01-08]. Dostupné z:

https://access.redhat.com/documentation/zh-cn/red_hat_process_automation_manager/7.5/html/configuring_business_central_settings_and_properties/managing-business-central-git-hooks-con

42. **What is JDBC?.** *IBM* [online]. 2021 [cit. 2023-01-08]. Dostupné z: https://www.ibm.com/docs/en/informix-servers/12.10?topic=started-what-is-jdbc

43. **Class Awaitility.** *Javadoc* [online]. c2010-2017 [cit. 2023-01-08]. Dostupné z: https://www.javadoc.io/doc/org.awaitility/awaitility/3.0.0/org/awaitility/Awaitility.html

44. **Interface Future<V>.** *Oracle* [online]. 2020 [cit. 2023-01-08]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Future.html

45. **Guide to java.util.concurrent.Future.** *Baeldung* [online]. 2021 [cit. 2023-01-08]. Dostupné z: https://www.baeldung.com/java-future

46. **JUnit 5.** *JUnit* [online]. c2023 [cit. 2023-01-08]. Dostupné z: https://junit.org/junit5/

47. **JMockit.** *JMockit* [online]. [cit. 2023-01-08]. Dostupné z: https://jmockit.github.io/index.html

48. **Docker.** *Docker* [online]. c2022 [cit. 2023-01-08]. Dostupné z: https://www.docker.com/

16. Seznam obrázků

1.1 Sekvenční diagram komunikace s poskytovatelem dat

5.1 Přehled základních komponent Decision engine neboli Rule Engine

zdroj: *Chapter 1. Decision engine in Red Hat Decision Manager. In: Red Hat Customer Portal [online]. c2023 [cit. 2023-01-09]. Dostupné z: https://access.redhat.com/documentation/en-us/red_hat_decision_manager/7.4/html/decision_engine_in_red_hat_decision_manager/decision-engine-con_decision-engine*

5.2 Přehled cyklu při provádění pravidel

zdroj: *SALATINO, Mauricio a Esteban ALIVERTI. Mastering JBoss Drools 6. Packt Publishing Limited, 2016, 330 s. ISBN 9781783288625.*

5.3 Síť Rete pro pravidlo „Black cat"

5.4 Síť Rete pro pravidla „flight with baggage" a „person has a ticket for flight"

5.5 Diagram reprezentující vztahy mezi pojmy potřebnými při implementaci aplikace s Drools engine

zdroj: *SALATINO, Mauricio a Esteban ALIVERTI. Mastering JBoss Drools 6. Packt Publishing Limited, 2016, 330 s. ISBN 9781783288625.*

7.1 Přehled nástrojů pro tvorbu business aktiv v aplikaci Business Central

7.2 Struktura projektů v aplikaci Business Central

zdroj: *Integration with Aries Blueprint. Jboss [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://docs.jboss.org/drools/release/7.73.0.Final/drools-docs/html_single/#_ch.kie.blueprint*

7.3 Příklad vytváření business pravidla v nástroji Guided Rules

zdroj: *Integration with Aries Blueprint. Jboss [online]. 2022 [cit. 2023-01-08]. Dostupné z: https://docs.jboss.org/drools/release/7.73.0.Final/drools-docs/html_single/#_ch.kie.blueprint*

7.4 Stránka v aplikaci Business centrál s názvem Overview

8.1 Diagram ukazující zapojení knihovny BusinessRequestSplitter systému Symphony a využití aplikace Business Central

9.1 Diagram nastiňující problematiku využití aplikace Business Central

10.1 Ukázka obrazovky z aplikace Business Central při vytváření business pravidla

10.2 Ukázka výsledného business pravidla vytvořeného v aplikaci Business Centra

11.1 Sekvenční diagram pro využití mechanismu fireUntilHalt při zastavení engine

11.2 Sekvenční diagram pro využití mechanismu Future při zastavení engine

12.1 Sekvenční diagram komunikace s poskytovatelem dat

12.2 Sekvenční diagram komunikace s AirFrance bez zapojení knihovny BusinessRequestSplitter

12.3 Sekvenční diagram komunikace s AirFrance po zapojení knihovny BusinessRequestSplitter

12.4 Sekvenční diagram zachycující použití knihovny BusinessRequestSplitter

13.1 Seznam klíčových slov, které lze požít v při vytváření business pravidel

13.2 Ukázka z aplikace Business Central, kde jde vidět vytváření pravidel ze zadání č. 2

13.3 Ukázka z aplikace Business Central, kde je vidět výsledný rdl file pro zadání č.2

17. *Přílohy*

Knihovna Business Request Splitter

- cz.aag.drools.businessrequestsplitter

Aplikace Bridge

- cz.aag.drools.bridge

Light NdcPricer

- cz.aag.light.ndcpricer
- cz.aag.light.ndc.api

Business Central

- conf/bridge.jar
- conf/standalone.xml
- conf/post-commit
- dockerfile
- ReadMe

- BusinessRules.txt
- CreateDrlFile.sql
- ReadMe
- SeznamParametru.txt