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

# Diplomová práce

# Zdokonalení a automatizace nástrojů pro podporu vývoje softwaru

Plzeň 2018 Bc. Lukáš Rostás

**Místo této strany bude zadání práce.**

# Prohlášení

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

V Plzni dne 28. června 2018

Bc. Lukáš Rostás

# Abstract

The goal of this thesis is to design and implement a web application that triggers configurable actions. Actions are triggered after saving work items in the Rational Team Concert tool. There are two types of actions that can be triggered. The first are email notifications and the second are RESTful API calls. The second action allows us to remotely control UrbanCode Deploy server and run its application processes. The system will allow us to define rules that have to be satisfied before the actions are triggered. The implemeted system consist of a web application and RTC Eclipse plugin that extends the RTC server. The plugin invokes the application that evaluates the rules and triggers the actions.



# Abstrakt

Hlavním cílem této práce je navržení a implementace webové aplikace, která umožní po uložení pracovní položky v nástroji Rational Team Concert vyvolat nakonfigurovatelné akce. Akce existují dvojího druhu, a to emailové notifikace a volání RESTful API. Použitím druhé jmenované akce je možné volat vzdálený UrbanCode Deploy server, spouštět jeho aplikační procesy a vzdáleně tak spustit například nasazení. Aplikace umožní definovat pravidla, po jejichž splnění jsou akce spuštěny. Součástí programového vybavení je i RTC Eclipse plugin, který rozšiřuje funkčnost na straně RTC o volání aplikace, která vyhodnocuje pravidla a spouští akce.

# Poděkování

Rád bych poděkoval Ing. Lukáši Holému, Ph.D. za vstřícnost, ochotu a cenné rady při vedení této práce.

# Obsah

# 1 Úvod

Předložená diplomová práce se zabývá moderním pojetím dodávky softwaru, což nabývá v době neustálé digitalizace stále větší význam. Společnosti jsou nuceny zefektivňovat svoje procesy a zrychlovat dodávky informačních řešení. Jedním z moderních přístupů k výrobě software se zabývá i tato teoretická práce.

V teoretické části bude čtenář seznámen s problematikou nasazování software a s principy přístupu DevOps. Poté budou představeny nástroje společnosti IBM pro podporu DevOps, zejména pak nástroje Rational Team Concert a UrbanCode Deploy, se kterými se bude pracovat v praktické části.

Cílem praktické části práce je navrhnout a implementovat systém, který by v reakci na uložení pracovní položky v Rational Team Concert umožnil spouštět konfigurovatelné akce. Aby akce byla spuštěna, musí být splněna pravidla, která jsou uživatelem nakonfigurovatelná.

Díky systému by tak například bylo možné okamžitě spustit nasazení prostřednictvím nástroje UrbanCode Deploy.

V kapitole čtyři budou identifikovány požadavky na vylepšení schopnosti RTC. Dále bude diskutována architektura aplikace, rozdělení do modulů a programovací techniky použité pro implementaci.

Pátá kapitola se zabývá implementací systému. Dále bude aplikace otestována, zhodnocena a budou představena možná vylepšení.

# 2 Nasazování software

V následující kapitole bude představen přístup DevOps a bude rozebrána problematika nasazování softwarových produktů.

## 2.1 Přístup DevOps

Hnutí *DevOps* spatřuje světlo světa koncem prvního desetiletí nového tisíciletí. Samotný název je složenina dvou zkratek pro *development* a *operations*. Termín tedy naznačuje propojení dříve často separovaných oblastí v rámci životního cyklu software, a to vývoje – development a provozu – operations. Za termínem operations se skrývá tým pracovníků, kteří se starají o správný chod poskytovaného software, řešení klientských požadavků a dodržování SLA[1].

S nástupem agilních metodik vývoje software, kdy je kód dodáván v kratších časových intervalech, bylo nutné překlenout pomyslnou propast mezi vývojem a provozem podporou vzájemné spolupráce a prolnutí obou oblastí. Spolu s automatizací nasazování měl tento přístup vést k tomu, aby výsledný produkt byl dodán rychlejší a efektivnější cestou, což bylo původním záměrem DevOps [11].

Je třeba zmínit, že přístup DevOps není exaktně definovanou metodikou jako je např. ITIL[2], ale pouze zastřešující koncept, který napomáhá hladké spolupráci vývoje a provozu [3]. Pojetí DevOps se tak může lišit podle společnosti, která DevOps adaptuje.

DevOps uplatňuje principy agile a lean napříč celým procesem dodání software. Maximalizuje rychlost dodávky produktu nebo služby od počáteční myšlenky přes release až po zpětnou vazbu od zákazníka a vylepšení na základě zpětné vazby [10].

### 2.1.1 Cíle DevOps

DevOps si klade následující cíle:

- Větší spokojenost zákazníků,
- větší možnosti inovace,

[1]Termín označuje smlouvu sjednanou mezi poskytovatelem služby a jejím uživatelem. Přesně vymezuje rozsah a úroveň služeb poskytovaných dodavatelem zákazníkovi.

[2]Soubor zdokumentovaných postupů a metodik pro řízení a správu IT služeb.

- rychlejší přínos užitku z produktu.

### 2.1.2 Principy DevOps

#### Vývojové prostředí

Vývoj a testování by měly probíhat v prostředí co nejpodobnějšímu **produkčnímu**. Jedná se o koncept *Shift left*, kdy je cílem posunout možná rizika v rámci vývojového cyklu software na pomyslné časové ose co nejvíce doleva a odhalit tak případné problémy ještě před nasazováním do provozu [4].

#### Proces nasazení

Proces nasazování by měl být opakovatelný a spolehlivý.

#### Sledování provozu

Metriky sledující vlastnosti systému v provozu by měly být sledovány a vyhodnocovány i v průběhu nasazení a testování nejen v produkčním prostředí. To může pomoci včasnému odhalení chyb.

#### Zpětná vazba

Cílem je zesílit zpětnou vazbu od zákazníka k dodavateli tak, aby dodavatel mohl pružněji reagovat a provádět požadované změny.

### 2.1.3 DevOps podle IBM

IBM přichází se svým vlastním pojetím přístupu DevOps 2.1.


Obrázek 2.1: DevOps podle IBM

### 2.1.4 Techniky DevOps

Zásadními koncepty pro DevOps jsou *Continuous integration* a *Continuous delivery*. Obě techniky výrazně redukují čas od zachycení požadavku do doby než je požadovaná funkčnost využívaná zákazníkem [11].

#### Continuous Integration

Continuous Integration neboli kontinuální integrace je metoda k urychlení vývoje softwaru a včasnému odhalení chyb. Zajišťuje průběžnou integraci práce jednotlivých vývojářů, nebo týmů. Součástí jsou automatické testy, kdy je ověřeno, že aplikace funguje jako celek.

Cílem je zmenšit riziko neúspěchu dodávky software. Dodat produkt vysoké kvality pomocí transparentního, snadno kontrolovatelného procesu a umožnění rychlé zpětné vazby od zákazníka. V konečném důsledku to znamená samozřejmě úsporu nákladů, a to zejména automatizací stále se opakujících kroků prováděných ručně, eliminací prostojů a předělávání hotového díky redukci a včasnému odhalení chyb napříč celým vývojovým procesem [4].

#### Continuous Delivery

Průběžná integrace vede k principu Continuous Delivery (CD). Zatímco průběžná integrace zahrnuje automatizovanou fázi sestavení a testů, Continous Delivery umožňuje testovat také business logiku, což jednotkové testy neumožňují. Na konci každého integračního sestavení je produkt předkládán k akceptačnímu testování. To umožňuje odhalit případné chyby v business vrstvě včas.

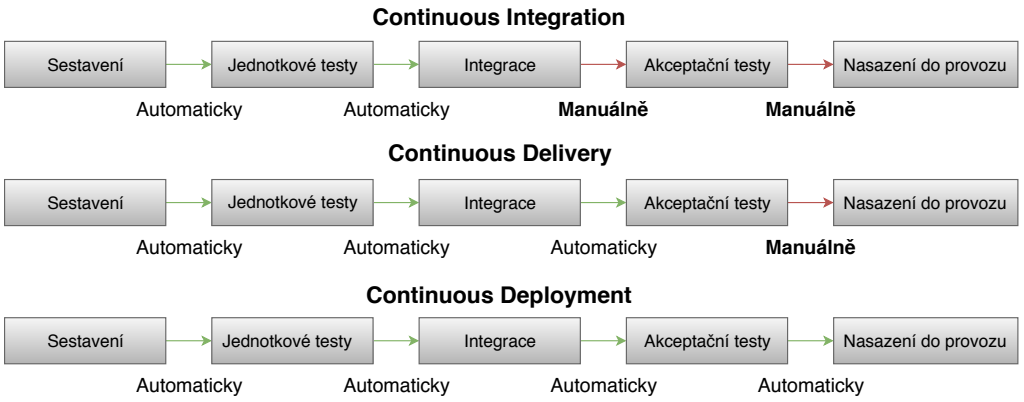
#### Continuous Deployment

Continous Deployment – průběžné nasazování je další krok po průběžné dodávce. Principem je nasazování změn, které prošly automatickými testy, rovnou do provozu v co nejkratším čase.

#### Porovnání technik

Jednotlivé techniky se liší mírou toho, kolik kroků z celého procesu nasazení dokážou automatizovat. To je vyjádřeno na obrázku 2.2.

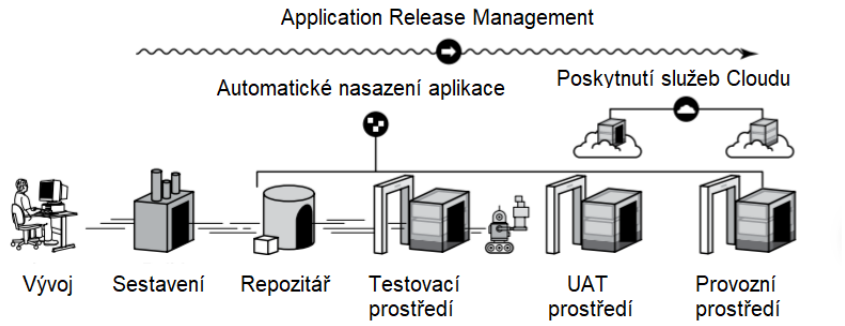
Zatímco metodika Continuous Integration končí fází sestavení, Continuous Delivery je automatická dodávka na testovací prostředí a Continuous Deployment zahrnuje automatizaci nasazení na produkci.




Obrázek 2.2: Porovnání technik pro DevOps

## 2.2 Delivery pipeline

Delivery pipeline neboli doručovací linka se skládá z fází (stage), kterými prochází aplikace od vývoje až po zařazení do produkce. Delivery pipeline lze definovat jako automatizovaný proces zahrnující sestavení, nasazení, testování a release [4]. Na obrázku 2.3 je znázorněn typický soubor fází. Tyto fáze se mohou lišit nejen v rámci jednotlivých organizací, ale také v závislosti na potřebách organizace. Kromě fází se může lišit i úroveň automatizace. Některé organizace své doručovací linky plně automatizují, jiné kontrolují software pouze ručně [10].




Obrázek 2.3: Delivery Pipeline [10]

Typická Delivery pipeline sestává z následujících fází [11]:

- *Vývojové prostředí* – Kromě nástrojů integrovaného vývojového prostředí (IDE), které používají vývojáři k psaní kódu, obsahuje tato fáze nástroje, které umožňují spolupráci. Konkrétně se jedná o systém

správu nástroje verzí, správu pracovních položek, spolupráci, testování jednotek a plánování projektů.

- *Sestavení* – Ve fázi sestavení je kód kompilován do binarních souborů a jsou vykonány jednotkové testy.
- *Repozitář* – jedná se o společné úložiště binárních souborů a konfiguračních souborů.
- *Testovací prostředí* – testovací prostředí je místo výkonu ověřování kvality.
- *UAT prostředí* – místo výkonu akceptačních testů.
- *Produkční prostředí* – produkční prostředí. Ostrý provoz.

# 3 Nástroje IBM pro podporu DevOps

V této kapitole budou popsány nástroje od společnosti IBM, které slouží pro podporu vývoje v duchu DevOps. Nejprve bude představena skupina nástrojů Rational Collaborative Lifecycle Management (CLM), poté budou detailněji popsány aplikace Rational Team Concert (RTC) a UrbanCode Deploy (UCD), se kterými se pracuje v praktické částí práce.

## 3.1 CLM

Rational Collaborative Lifecycle Management je název pro skupinu nástrojů, pomocí kterých lze uplatnit principy DevOps, tak jak je vidí společnost IBM. Jedná se o sadu důsledně integrovaných nástrojů založených na architektuře sdílených aplikačních služeb, která se jmenuje Jazz Team Server. Řešení Rational pro systémové a softwarové inženýrství nabízí produkty, služby a doporučené postupy pro aktivity systémového inženýrství a vývoje softwaru napříč správou požadavků, návrhem, vývojem a testováním. Společně s UCD pak podporují DevOps přístup. CLM nabízí řešení v těchto oblastech [5]:

- správa požadavků,
- plánování, spolupráce a vývoj,
- konfigurační management,
- reporting,
- monitoring.

CLM se skládá z následujících nástrojů [5]:

- Rational Quality Manager (RQM),
- Rational Doors Next Generation (RDNG),
- Rational Team Concert (RTC).

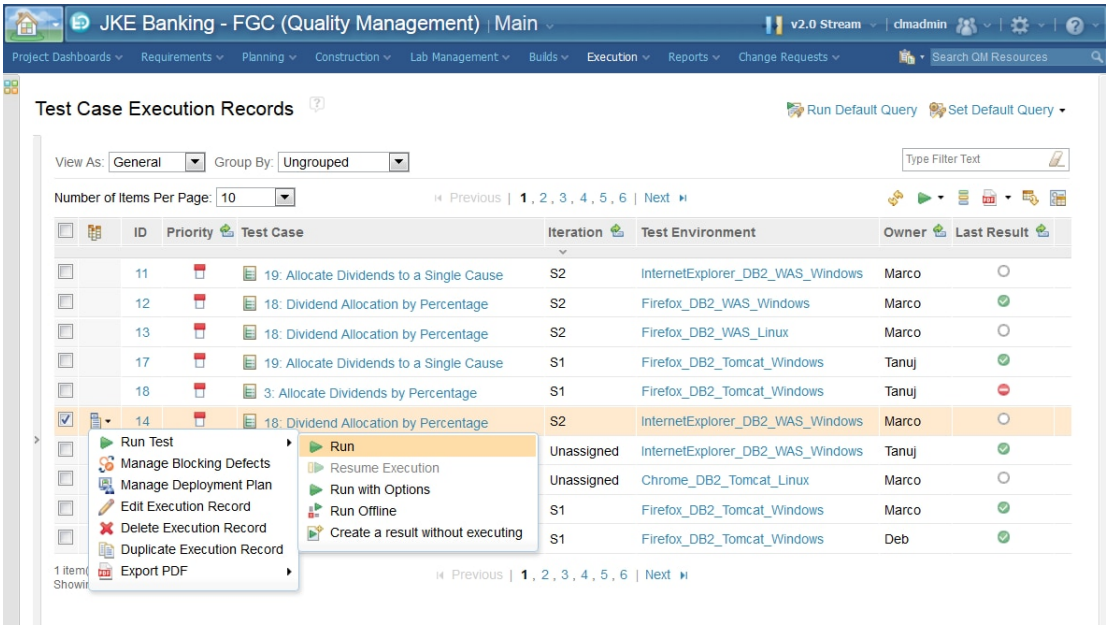
### 3.1.1 Jazz Team Server

Jazz Team Server (JTS) je platforma, která poskytuje základní služby, které umožňují, aby sady nástrojů spolupracovaly jako jeden logický server. Stará se o správu uživatelů, autorizaci. JTS funguje jako jádro integrace jednotlivých CLM aplikací. Každý nástroj CLM využívá určité prostředky poskytované prostřednictvím služeb Jazz Foundation Services k interakci v rámci serveru Jazz Team Server. Každý nástroj vystavuje své vlastní služby a data přes webové služby RESTful.

### 3.1.2 Rational Quality Manager

Nástroj RQM poskytuje schopnosti **testování** a správy testování, plánování jakosti, automatické testování, správu defektů, správu uživatelů zapojených do testování a jejich rolí. To vše pomáhá dosáhnout kvality produktu již od návrhu.

Mezi funkce správy testů patří dynamické testovací plány, řízené sledy prací, efektivita testovacího pracoviště, analýzy pokrytí testu a ruční tvorba testů. Tyto funkce lze integrovat s jinými artefakty životního cyklu, jako jsou např. pracovní položky a požadavky nebo s tvorbou sestav. Je možné využít panely *dashboard*, které poskytují podrobné a vysoce konfigurovatelné analýzy, které pomáhají monitorovat stav a průběh projektu. Na obrázku 3.1 se nachází ukázka uživatelského rozhraní nástroje RQM [6].



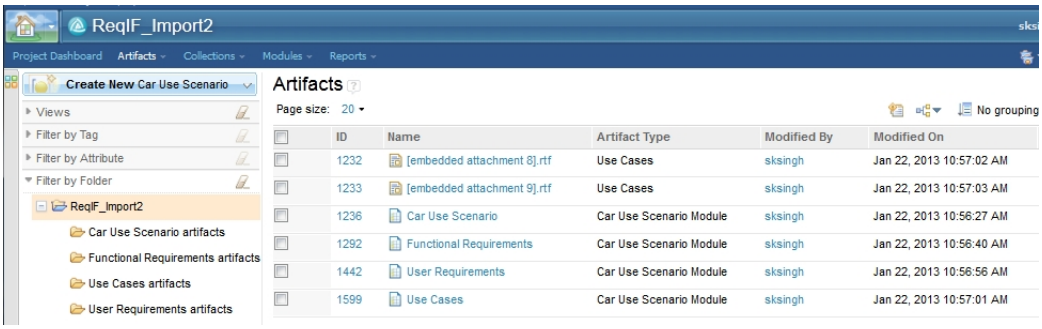

Obrázek 3.1: Uživatelské rozhraní nástroje RQM

### 3.1.3 Rational Doors Next Generation

Nástroj Rational Doors Next Generation slouží k definování a správě **požadavků** ve vývojovém projektu životního cyklu. Nástroj umožňuje definovat, vyvolat, zachytit, zpracovat, dokumentovat, řídit, diskutovat a posoudit požadavky a podpůrné artefakty s ostatními členy všech zúčastněných stran vývoje. Týmy mohou spravovat požadavky pomocí atributů, značek, filtrovaných pohledů, typů typů artefaktů a typů odkazů, indikátorů změn a dashboardů projektu.

RDNG pomáhá uspořádat požadavky, stanovit jejich prioritu, trasovat vztahy mezi nimi a trasovat změny, které je ovlivňují. Po integraci produktu IBM Rational Quality Manager s produktem Rational DOORS je možné přidružením požadavků z produktu Rational DOORS k pohledu Požadavky v plánu testování nebo testovacím případu pokrýt průběh implementace jednotlivých požadavků [7].

Ukázka uživatelského rozhraní se nachází na obrázku 3.2.




Obrázek 3.2: Uživatelské rozhraní nástroje RDNG

### 3.1.4 Integrace v CLM

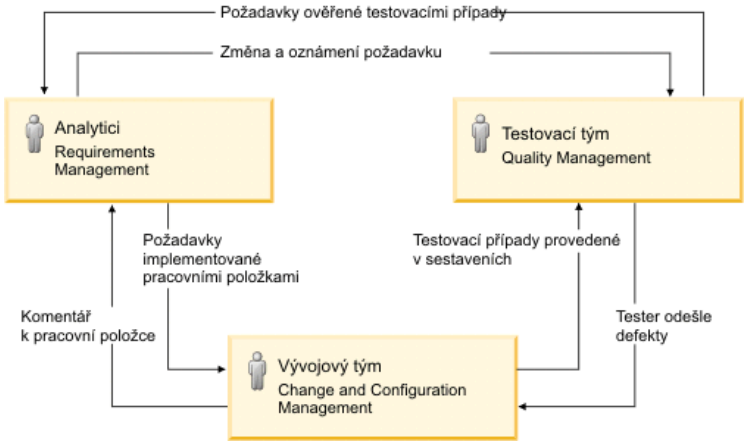
Produkt Collaborative Lifecycle Management nabízí integraci mezi aplikacemi Rational Team Concert, RDNG a RQM Management založenými na platformě Jazz za účelem spojení práce analytiků s vývojovými a testovacími týmy. Integrace CLM je postavena na platformě Jazz Foundation a poskytuje společný přístup k propojení artefaktů, panelů dashboard, zabezpečení a rámců uživatelských rozhraní.

Integrace CLM poskytuje následující funkce [1]:

- Propojení mezi artefakty napříč aplikacemi – například testovací případy lze propojit s pracovními položkami a požadavky.

- Podržení ukazatele myši nad odkazy – umožňuje rychlé zjištění podrobností o cíli odkazu. Například testeři mohou monitorovat stav defektu, který oznámili vývojovému týmu.
- Přidání widget modulů – z různých aplikací je možné přidat widget na panel dashboard ke sledování stavu položky napříč projekty. Je tedy možné například přidat widget zobrazující defekty, které blokují testery.

Členové týmu mohou použít integrace CLM k dosažení obchodních cílů v rámci životního cyklu aplikace, jak je uvedeno na následujícím obrázku a příkladech:




Obrázek 3.3: Spolupráce analytiků, vývojářů a testerů

### 3.1.5 OSLC

Aplikace RM je postavena na integrační platformě Jazz. Softwarová architektura pro aplikaci RM je založena na specifikaci Open Service for Collaboration Lifecycle Collaboration (OSLC), která používá společnou sadu zdrojů, formátů a architektonických služeb REST umožňující sdílení dat mezi aplikacemi CLM. OSLC poskytuje otevřenou škálovatelnou sadu specifikací pro integraci nástrojů do heterogenních vývojových prostředí. Sdílení dat podporuje propojení založené na protokolech HTTP, identifikaci zdrojů podle identifikátorů URI a získávání informací pomocí standardních typů médií. Sdílení dat je zabezpečeno pomocí standardních ověřovacích mechanismů HTTP a autorizačního protokolu OAuth.

## 3.2 Rational Team Concert

Produkt Rational Team Concert je nástroj určený pro podporu týmové spolupráce. RTC zajišťuje funkce, které integrují úlohy vývoje projektů včetně plánování iterací, definice procesů, aplikace Change Management, sledování defektů, aplikace konfiguračního řízení, automatizace sestavení a tvorby sestav.

Následuje výpis čtyř hlavních oblastí použití, které nástroj RTC poskytuje [8]:

- *Change Management* – Hlavním předmětem oblasti Change Management jsou **pracovní položky**, které sledují a koordinují děje, defekty, položky plánu a obyčejné úlohy. Pracovní položky a proces sledu prací, kterým procházejí, lze upravit a přizpůsobit konkrétnímu projektu.

- *Plánování* – Schopnost Plánování poskytuje nástroje, které pomohou s plánováním, sledováním a vyrovnáváním pracovní zátěže pro celé projekty, pro týmy v rámci těchto projektů a pro jednotlivé vývojáře. Plány jsou dostupné každému v týmu a zobrazují postup v rámci release a iterací k libovolnému okamžiku v čase.

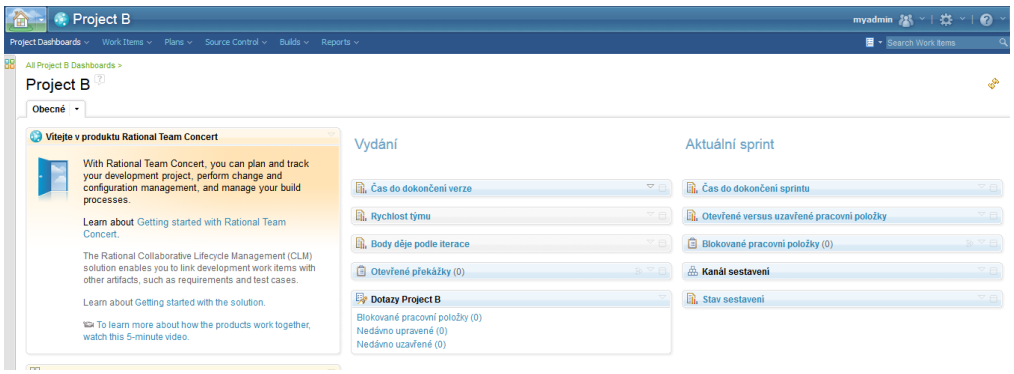
- *Konfigurační management* – Systém řízení zdrojů založený na komponentách poskytuje silnou podporu paralelního vývoje, agilního vývoje a geograficky rozdělených týmů. Je úzce integrován s trasováním defektů, sestaveními a automatizací procesů.

- *Automatizace* – Schopnost Automatizace poskytuje řízení správy sestavení pro vývojové a testovací týmy. Členové týmu mohou sledovat průběh sestavení, nechat si zobrazit výstrahy a výsledky sestavení, vyžádat sestavení a trasovat vztahy sestavení k artefaktům, jako jsou sady změn a pracovní položky.

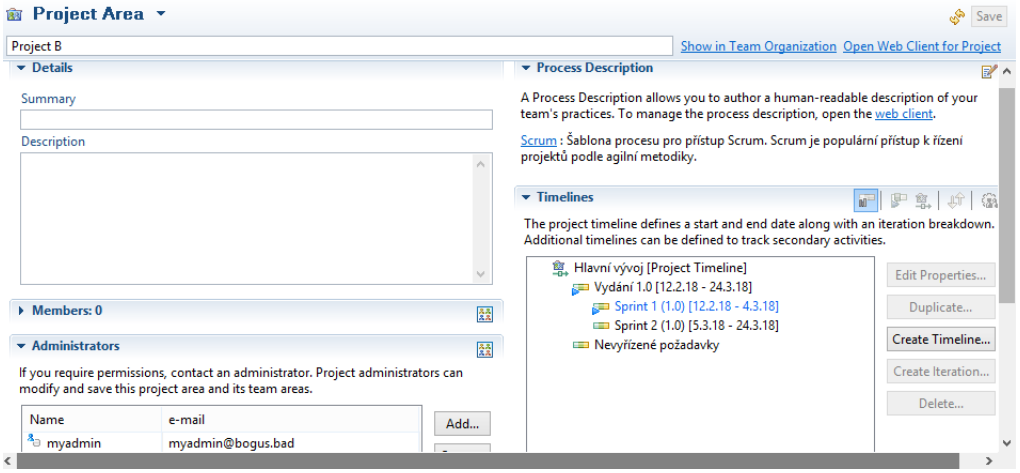
### Uživatelské rozhraní

Produkt Rational Team Concert nabízí standardní webové rozhraní, ale také klientské rozhraní vycházející z platformy Eclipse. Klientská rozhraní poskytují vývojářům bohaté integrované vývojové prostředí pro sestavování a doručování artefaktů. Webové rozhraní se hodí k administraci serverů a projektů a umožňuje uživatelům přístup k oblastem projektu, procházení informací o úložištích, zobrazování a úpravu plánů, aktualizaci úloh a čtení údajů o nedávných událostech. Pro uživatelská rozhraní platí, že ne všechny operace, které se dají vykonat v klientském rozhraní v rámci IDE, jdou provést také přes webové rozhraní a naopak.

Na obrázku 3.4 je zobrazeno webové rozhraní RTC následováno ukázkou uživatelského rozhraní v IDE RTC Eclipse client.




Obrázek 3.4: Webové rozhraní nástroje RTC




Obrázek 3.5: Uživatelské rozhraní v RTC Eclipse Client

### 3.2.1 Oblast projektu

Oblast projektu[1] je základní struktura, ve které jsou uloženy informace o softwarových projektech. Oblast projektu definuje předměty plnění projektu, strukturu týmu, plánování a samotný proces. Všechny artefakty projektu jako jsou plány, pracovní položky, požadavky, případy užití, testy, soubory spravované SCM jsou svázány k oblasti projektu. Projekt životního cyklu typicky seskupuje více oblastí projektu, přičemž členové jednotlivých oblastí projektu mezi sebou spolupracují.

[1] V originále Project Area.

#### Proces

Každá oblast projektu má definovaný proces projektu, který určuje práci členů týmu. V platformě Jazz lze pomocí procesu definovat role uživatelů a jejich oprávnění k provádění operací v rámci nástroje, jako např. ke změně stavu pracovní položky. Každá komponenta serveru Jazz zohledňuje procesy. Proces oblasti projektu definuje [8]:

- uživatelské role,
- oprávnění k rolím,
- harmonogram a iterace,
- předběžné podmínky a *následné akce*,
- typy pracovních položek,
- změny stavu pracovních položek.

Proces lze přenášet mezi oblastmi projektu.

### 3.2.2 Pracovní položky

Pracovní položky (Work items) reprezentují úlohy a problémy, které se vyskytují v průběhu vývojového cyklu, a které je třeba vyřešit. Každá pracovní položka má množinu **atributů**, které ji charakterizují. Základním atributem je typ pracovní položky.

#### Typ

Typ pracovní položky určuje její atributy a její workflow – tedy životní cyklus položky a její stavy. Pro představu budou představeny typy pracovních položek ze šablony procesu Scrum [5]:

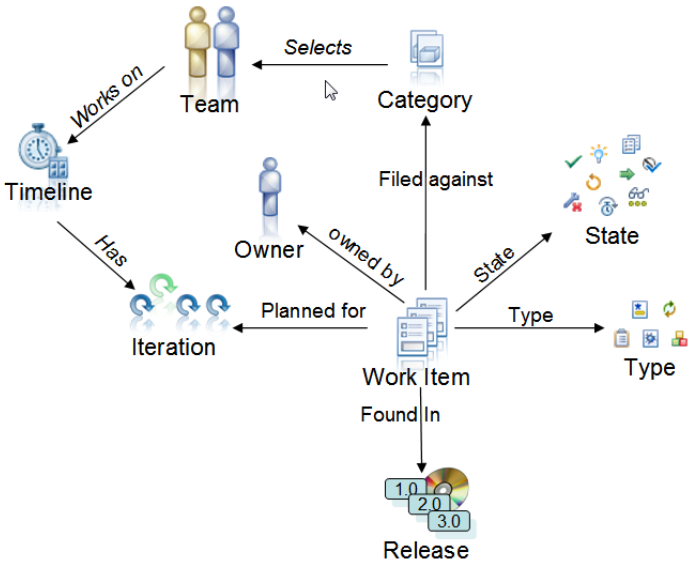
- *Defekt* – Identifikuje programovou chybu.
- *Úloha* – Popisuje specifickou část práce.
- *Děj* – Popisuje část případu užití.
- *Retrospektiva* – Zaznamenává, co se v dokončené iteraci dařilo a co se nedařilo.
- *Překážka* – Sleduje věci, které překážejí provádění procesu.

- *Epic* – Používá se v případě, že je děj příliš veliký na to, aby byl dokončen v jednom sprintu, nebo když je obtížné odhadnout množství práce. Epic lze rozdělit na několik samostatných dějů.
- *Položka schválení* – Použití v případě nutnosti, aby změny provedené jedním týmem přijal jiný tým.

Administrátor projektu může definovat vlastní typy pracovních položek a přiřadit k ní libovolnou množinu atributů, včetně vlastních[2] atributů.

### Atributy

Pracovní položky mají množinu atributů, které jsou nositeli informací o pracovní položce. Tato množina je definována typem pracovní položky, což je zároveň jeden z atributů. Mezi nejtypičtější atributy patří: id, typ, stav, souhrn, datum vytvoření, vlastník nebo oblast projektu. Každý atribut je určitého typu, např. řetězec, logická hodnota, timestamp, apod. Administrátor má možnost vytvořit svoje vlastní atributy. Na obrázku 3.6 jsou zobrazeny základní atributy pracovní položky a jejich vztahy.




Obrázek 3.6: Základní atributy pracovní položky [8]

[2]Tzv. *custom* atributy

## 3.3 IBM UrbanCode Deploy

IBM UrbanCode Deploy je nástroj pro podporu vývoje software – zahrnuje automatizaci nasazení na specifikované prostředí. UCD je navrženo tak, aby byla umožněna rychlá zpětná vazba a nepřetržitá dodávka. V konečném důsledku se jedná o urychlení dodání produktu cílovému subjektu. Důležitým aspektem je možnost auditu při procesu nasazování, možnost verzování a mechanizmus schvalovacího procesu[3][2].

UrbanCode Deploy umožňuje orchestraci změn napříč servery, vrstvami a komponentami a poskytuje ucelený přehled o tom „co je kde nasazeno“ a který uživatel provedl jakou změnu.

### 3.3.1 Přehled pojmů v UCD

#### Agenti

Agenti jsou procesy, které vykonávají nasazení na cílovém systému. Agenti mohou být spuštěni na fyzickém nebo virtuálním stroji.

#### Komponenty

Komponenty jsou ústředním prvkem UCD. Reprezentují nasaditelné jednotky – artefakty. Artefakty mohou být soubory, obrázky, databáze, konfigurační soubory a další soubory, které souvisejí se softwarovým projektem. Při každé změně souborů, které tvoří komponentu, je vytvořena nová verze komponenty. Verze komponent jsou zaznamenávány a uživatel tak může rozhodnout, která verze komponenty bude použita při nasazení.

#### Pluginy

Pluginy rozšiřují možnosti pro komponentové procesy. UCD je tak například schopno změnit stav pracovní položky v RTC a přidat k pracovní položce komentář.

#### Resources

Zdroje reprezentují vztahy mezi komponentami, mezi agenty, kteří je nasazují, a cílovým prostředím. Je umožněno vytvářet skupiny, které reprezentují cílové servery a prostředí. Ke skupině jsou připojeni agenti, kteří nasazují na prostředí určené komponenty.

[3]Možnost specifikovat u úlohy nutnost schválení uživatelem s odpovídající rolí.

Nejčastějším zdrojem je agent, program, který umožňuje komunikaci mezi serverem a cílovým prostředím. Agent spouští procesy nasazující komponenty.

### Aplikace

Aplikace je skupina komponent, které jsou nasazovány společně. Aplikace mají aplikační procesy. Aplikace obsahuje prostředí, které popisuje všechna umístění nasazení, která jsou potřebná v procesu zavádění a spouští procesy aplikací pro nasazení komponent do prostředí.

### Komponentový proces

Komponentový proces je posloupnost kroků, která pracuje s artefakty jedné komponenty. Každá komponenta může obsahovat více procesů. Typickou operací procesu může být instalace komponenty. Příklad komponentového procesu je uveden na obrázku 3.7.

### Prostředí

Prostředí je množina zdrojů, kde je nasazena aplikace. Typicky prostředí zahrnuje hostitelský systém a UCD agenty, kteří provádějí nasazení. Aplikace může být nasazena na více prostředích. Zatímco prostředí je množina zdrojů, zdroje samy o sobě se mohou lišit podle prostředí.

### Aplikační proces

Aplikační procesy slouží k instalování a odinstalování aplikací. Aplikační procesy jsou vytvářeny v editoru procesu. Aplikační proces se vždy vztahuje ke konkrétnímu prostředí.

Typické workflow pro použití UCD je znázorněno na obrázku 3.8:

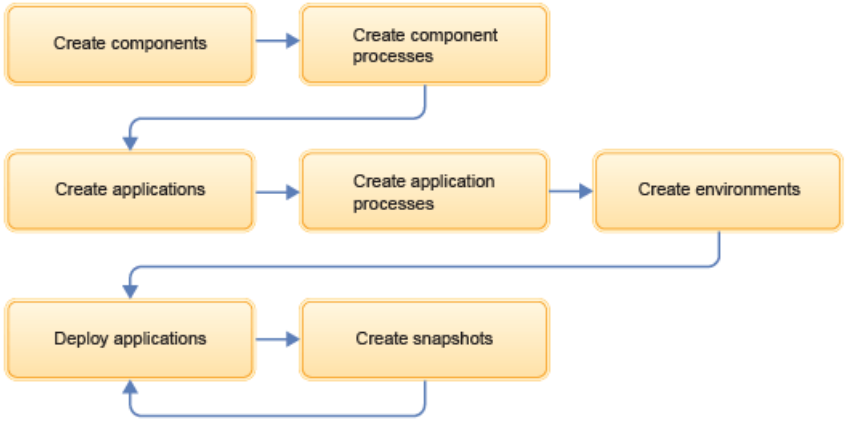
1. Prvním krokem při nasazování aplikace pomocí produktu IBM UrbanCode Deploy je importovat součásti aplikace jako komponenty.
2. Vytvoření komponentového procesu pro každou komponentu, která nasazuje další komponenty.
3. Vytvoření aplikace pro seskupení komponent.
4. Vytvoření aplikačního procesu, který spouští komponentové procesy.


Obrázek 3.7: Vytváření komponentového procesu prostřednictvím webového rozhraní.

5. Vytvoření jednoho nebo více prostředí, kam se budou nasazovat komponenty.

6. Spouštění aplikačního procesu k nasazení komponent.

7. Vytvoření funkčního *snapshotu*, aby mohlo být pohodlně nasazeno do jiných prostředí.




Obrázek 3.8: Sled prací při použití UCD

### 3.3.2 REST API

Ovládat UCD je možné přes webové rozhraní, prostřednictvím REST API a přes řádkového klienta (CLI), který je však pouze nadstavbou nad voláním REST API. Prostřednictvím API je umožněno provádět širokou škálu operací od vytváření, editování a mazání agentů, aplikačních procesů, komponent, přes spouštění procesů. Jako příklad bude uvedeno tělo příkazu pro spuštění aplikačního procesu „ap1“ na prostředí „Test“. Předmět procesu je komponenta „Comp1“ ve verzi 1.1.

```
{
  "application": "InterconnectionTest",
  "description": "Request for deployment",
  "applicationProcess": "ap1",
  "environment": "Test",
  "onlyChanged": "false",
  "versions": [
    {
      "version": "1.1",
      "component": "Comp1"
    },
  ]
}
```

# 4 Analýza a řešení

V této kapitole bude diskutován výběr a účel praktické části práce a uveden celkový pohled na vyvíjený systém.

## 4.1 Požadavky na vylepšení

Jednou z možností propojení RTC a UCD bylo zajistit vytváření odkazů mezi RTC pracovními položkami a UCD deploymenty. Při spouštění nasazení v UCD by uživatel musel zadat identifikátor RTC pracovní položky, která by reprezentovala release. Po skončení nasazování by pak byl výsledek nasazení zapsán do pracovní položky společně s URL v sekci *Odkazy* na aplikační proces v UCD, který provedl nasazení.

Druhým diskutovaným vylepšením byla možnost rozšíření pluginu v UCD, který umožňuje přidávat komentáře do pracovních položek a měnit jejich stavy. Úkolem by bylo rozšířit funkčnost tak, aby mohl měnit a nastavovat atributy *všech* typů včetně vlastních atributů a zakládat nové pracovní položky, protože současné možnosti pluginu jsou velice omezené.

Výše uvedená rozšíření však byla zamítnuta, jejich přidaná hodnota by nebyla příliš vysoká. Bylo rozhodnuto, že práce bude směřovat cestou vytvoření nástroje, který po uložení pracovní položky v RTC umožní spouštět nakonfigurovatelné akce. Systém nazvaný **RTC Action & Notification Engine**[1] by měl uživateli umožnit definovat pravidla, po jejichž splnění se akce vykoná. Pravidlem se rozumí výraz složený z názvů atributů pracovní položky, hodnot atributů, některých logických operátorů a z komparačních operátorů.

Spustitelné akce budou existovat dvojího typu:

1. volání `RESTful API`,
2. emailová notifikace.

Možnost volání REST API je primárně určená pro vzdálené ovládání nástroje UCD. Tím tak například bude umožněno automaticky po změně stavu pracovní položky vyvolat proces nasazení určité aplikace v UCD. Naopak UCD umí změnit stav pracovní položky v RTC a přidat do ní komentář, což bude použito pro zahrnutí oznámení výsledku operace v UCD do pracovní položky.

[1] Zkráceně RANE.

Systém bude zahrnovat pokročilejší možnosti notifikací v systému RTC. Emailovou notifikaci bude možné odeslat libovolnému počtu příjemců. Obsah zprávy by měl být parametrizovatelný, bude umožněno vložit do zprávy hodnoty atributů dané pracovní položky.

Pro další text bude RTC Action & Notification Engine pojmenován jako notifikační engine, a nebo jednoduše jako aplikace. Pokud se bude mluvit o „pluginu“, je tím myšlen RTC plugin, který volá notifikační engine.

## 4.2 Návrh řešení

Ke splnění výše uvedených požadavků bylo možné se vydat několika různými cestami. Jedna ze zásadních otázek byla, jakou formou ukládat pravidla. Jednodušší variantou bylo pro každou oblast projektu vytvořit soubor s pravidly a soubory pak umístit na server. Aplikace by pravidla načetla ze souborového systému a pak používala při vyhodnocování. Jedná se však o nepraktické řešení a v jistém smyslu jako zastaralé. Nakonec bylo rozhodnuto pro ukládání pravidel využít relační databázi a využít tak všechny výhody, které toto řešení nabízí, jako například transakční zpracování, strukturalizace dat, bezpečnost, apod.

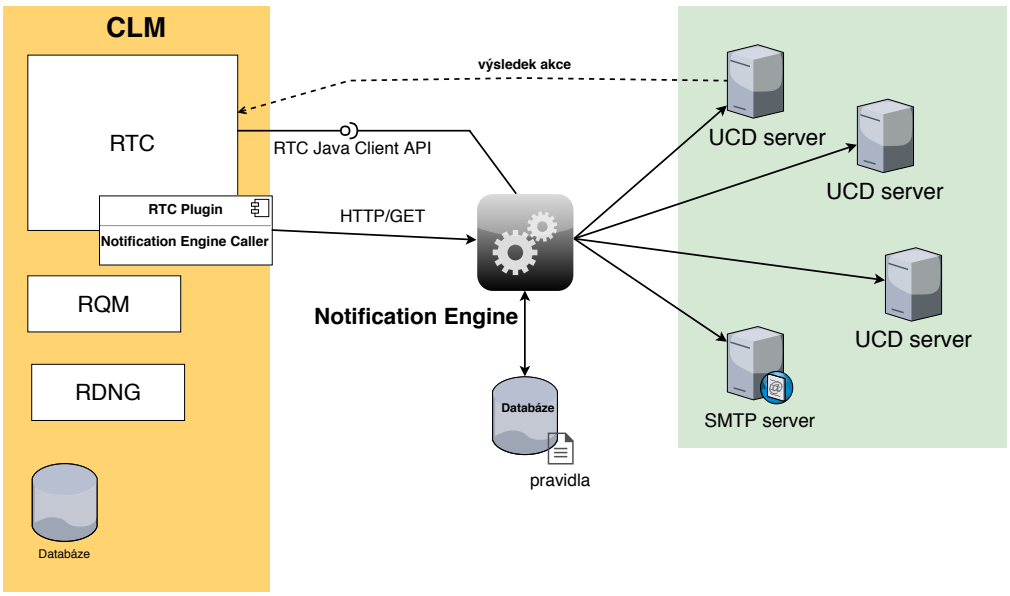
Další otázkou bylo, jak vlastně koncipovat architekturu notifikačního engine. První varianta počítala s vytvořením pluginu, který by rozšiřoval RTC server a spouštěl by engine při ukládání pracovní položky. Engine by byl součástí pluginu. Takové řešení by však způsobilo blokování ukládání pracovní položky do té doby, dokud by neproběhl celý proces výběru pravidel, jejich vyhodnocení a v neposlední řadě vykonání souvisejících akcí. To může být z principu časově náročná operace a v krajním případě by mohlo vyústit v neuložení změn v pracovní položce. Řešení s umístěním celé logiky do pluginu tedy nepřipadá v úvahu.

Jako správné řešení se ukázalo implementaci rozdělit na dvě části. První z nich je RTC plugin, jehož jediným úkolem je odeslání požadavku do notifikačního engine. Poté jeho činnost skončí. Druhou částí je webová aplikace, samotný notifikační engine, který po vyhodnocení pravidel spustí nadefinované akce. Obě komponenty budou popsány v dalších částech práce.

## 4.3 Architektura

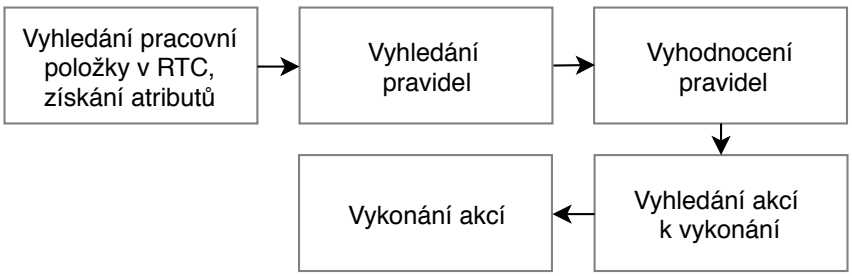
Širší kontext vyvíjené aplikace a pluginu je znázorněn na obrázku 4.1. Notifikační engine je webová aplikace, která má dvě logické části: administrační a výkonnou. Administrační část slouží uživateli k ovládání aplikace. Výkonná

část se stará o vyhodnocování pravidel a spouštění akcí vázaných k pravidlům.




Obrázek 4.1: Kontext aplikace

Výkonná část aplikace je vyvolána po uložení pracovní položky v RTC. Po uložení pracovní položky je spuštěn RTC plugin nazvaný *Rational Action and Notification Engine Caller*, který odešle HTTP požadavek obsahující číslo pracovní položky do aplikace. Poté se spustí proces nalezení a vyhodnocení pravidel potenciálně vedoucí ke spuštění nakonfigurovaných akcí. Posloupnost akcí v procesu je uvedena na obrázku 4.2.

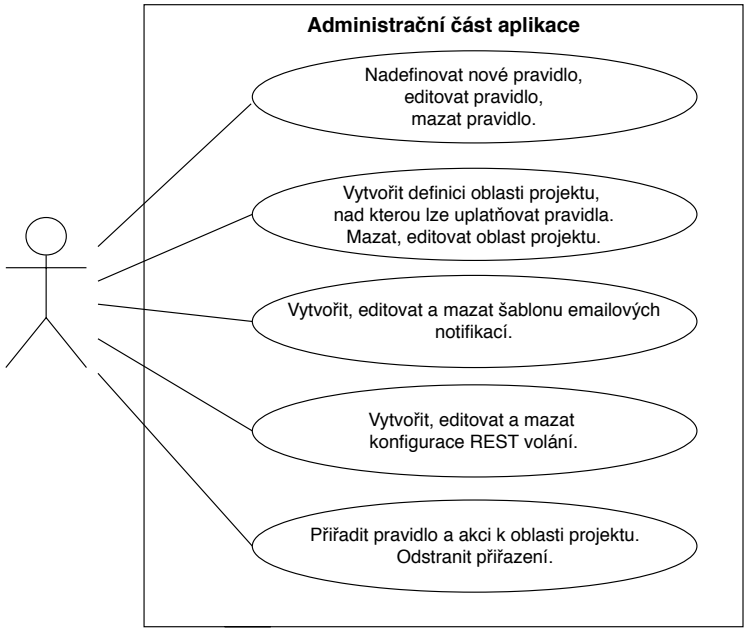



Obrázek 4.2: Sled akcí v notifikačním engine

V prvním kroku je získána pracovní položka z RTC prostřednictvím RTC Java Client API a jsou získány její atributy. Poté jsou vyhledána existující pravidla k vyhodnocení podle oblasti projektu, ze které pochází zkoumaná pracovní položka. Poté jsou nalezená pravidla vyhodnocena na základě získaných atributů. Posledním krokem je vyhledání a vykonání akcí, které jsou přiřazeny ke kladně vyhodnoceným pravidlům.

### 4.3.1 Administrační část aplikace

Administrační část aplikace je v podstatě rozhraní mezi uživatelem a databází. Prostřednictvím webových formulářů bude možné vytvářet, editovat a mazat jednotlivé entity vyskytující se v aplikaci. Uživatel bude moci vykonat akce, které jsou zachyceny na diagramu případů užití 4.3.




Obrázek 4.3: Diagram případů užití administrační části aplikace

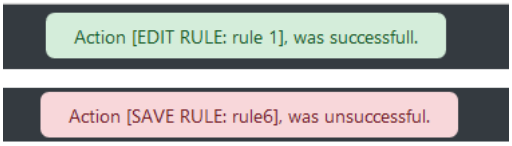
Prostřednictvím webového uživatelského rozhraní bude umožněno vytvářet pravidla, editovat je a mazat je. Na příslušné stránce bude zobrazen seznam existujících pravidel. Stejná množina operací bude umožněna pro definice oblastí projektu a správu obou druhů akcí.

Pro vytvoření přiřazení pravidel k projektu a svázání s vykonávanou akcí platí, že nebude možné přiřazení editovat, ale pouze odstranit. Pracnost implementace editace by neodpovídala skutečnému přínosu funkčnosti. Uživatel snadno může odstranit stávající přiřazení a vytvoření nového je otázkou chvíle.

Výsledek každé uživatelské akce je zobrazen uživateli ve formě boxu v horní části obrazovky. Příklad výsledku úspěšné a neúspěšné uživatelské akce v administrační části aplikace je zobrazen na obrázku 4.4.

### 4.3.2 Asynchronní servlety

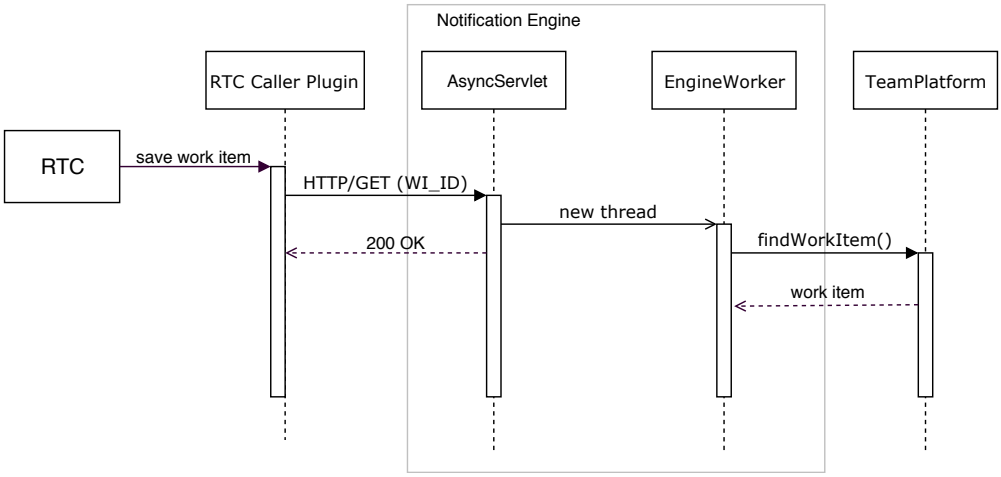
Již bylo řečeno, že RTC plugin odesílá HTTP požadavky do notifikačního engine. V aplikaci začne požadavek zpracovávat servlet, který začne vykonávat




Obrázek 4.4: Informace uživateli o výsledku uživatelské akce

workflow uvedené na obr. 4.2. Problém je ten, že vlákno pluginu je blokováno, dokud neobdrží odpověď, takže při použití klasického Java servletu by bylo ukládání pracovní položky blokováno do té doby, než by proběhl celý proces od výběru pravidel po spuštění odpovídajících akcí. Navíc, přístup k pracovní položce je v momentě ukládání pracovní položky uzamčený a není tak možné zjistit hodnotu atributů pracovní položky. Běh pluginu tedy musí skončit v co nejkratším čase.

Řešením je použití asynchronních servletů. V momentě, kdy asynchronní servlet obdrží požadavek od pluginu, je požadavek uložen do fronty ke zpracování a pluginu odeslána odpověď `200 OK`. V tu chvíli plugin končí svoji činnost a je dokončeno uložení pracovní položky. Komunikace je znázorněna na zjednodušeném sekvenčním diagramu 4.5. Sekvenční diagram znázorňuje komunikaci od uložení pracovní položky v RTC po získání reprezentace pracovní položky v notifikačním engine.

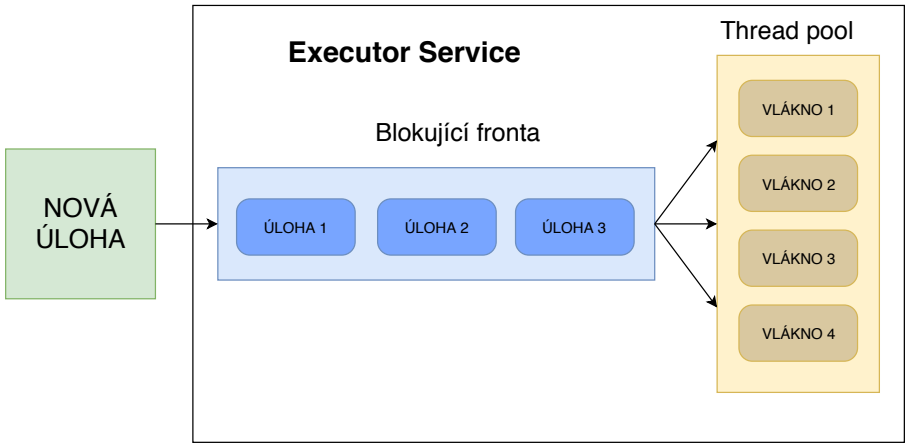



Obrázek 4.5: Sekvenční diagram

K realizaci asynchronnosti byly použity prostředky z balíku `java.util.concurrent`, který je součástí JDK od verze 5. Byl využit návrhový vzor Thread pool neboli fond vláken. Po spuštění aplikace je vytvořen tzv. *thread pool*, což je struktura spravující skupinu znovupoužitelných vláken organizovaných ve frontě. Vytvoření vlákna je drahá operace a měla by být prováděna

co nejméně, proto je znovupoužitelnost vláken velice vítaným prostředkem. Další výhodou je minimalizace režie spojené se správou vláken.

Jako konkrétní implementace thread poolingu je použitá třída ThreadPoolExecutor. Součástí je blokující vláknově bezpečná fronta, do které jsou ukládány požadavky v případě nedostupnosti volného vlákna. Funkce thread poolingu je znázorněna na obrázku 4.6.




Obrázek 4.6: Zpracování úloh

### Konfigurace thread poolu

Z konfiguračního souboru jsou nastavitelné následující parametry pro nastavení thread poolingu:

1. Počet vláken, které budou stále přítomné v poolu. V případě nečinnosti nejsou ukončena, ale čekají na přidělení práce.

2. Maximální počet vláken, které mohou být vytvořeny v poolu. Udává tedy vlastně maximální počet paralelně zpracovávaných pracovních položek.

3. Doba, po které jsou ukončena nečinná vlákna nad rámec fixního počtu vláken v poolu v bodě 1.

4. Kapacita fronty požadavků, které čekají na zpracování. Pokud je zadána nula, nebo je vlastnost v konfiguračním souboru zakomentována, tak má fronta neomezenou kapacitu. Kvůli možnosti neomezené[2] kapacity je v implementaci použitá blokující fronta `LinkedBlockingQueue`

[2]Kapacita fronty je ve skutečnosti omezena hodnotou `Integer.MAX_VALUE`.

reprezentovaná spojovým seznamem. Pokud by byla použita `ArrayBlockingQueue` reprezentovaná polem, kapacita fronty by nemohla být neomezená.

V tabulce 4.1 se nachází jména parametrů z konfiguračního souboru, kterým je umožněno přizpůsobit počty vláken prostředí, kde je aplikace spuštěna. V tabulce jsou také uvedeny výchozí hodnoty těchto parametrů, které jsou použity, pokud nejsou v konfiguračním souboru nastaveny.

| Parametr | Význam | Výchozí |
|---|---|---|
| `core_pool_size` | Počet stále udržovaných vláken v poolu. | 2 |
| `max_pool_size` | Maximální počet vláken v poolu. | 4 |
| `keep_alive` | Doba, po které jsou ukončena nečinná vlákna nad rámec prvního parametru v milisekundách. | 50000 |
| `queue_capacity` | Kapacita fronty. | 0 |

Tabulka 4.1: Konfigurace thread poolu

Pro produkční nasazení je tímto způsobem možnost optimalizovat počty aktivních vláken v aplikaci.

### Statistiky vláken

Po dokončení zpracování požadavku je vypsána doba běhu celého procesu zpracování pracovní položky od přijetí asynchronním servletem do dokončení spuštění akcí a také průměrná doba zpracování požadavku. Protože proměnné pro uchování celkové doby zpracování a celkového počtu požadavků jsou sdíleny mezi vlákny, bylo nutné použít synchronizačních prostředků, které Java nabízí. Konkrétně se jedná o atomické proměnné[3], které zaručují atomické operace nad proměnnými. Ukázka jejich použití následuje na následujícím výpisu.

```
1 static AtomicInteger processed = new AtomicInteger(0);
2 static AtomicLong totalTime = new AtomicLong(0);
3 (...)
4 long endTime = System.currentTimeMillis();
5 log.info(this.ctx + "thread finish in "
6         + (endTime - this.startTime) + " ms.");
7 totalTime.getAndAdd(endTime - this.startTime);
```

[3]Opět prostředek balíčku `java.util.concurrent.atomic`.

```
8  long timeTotal = EngineWorker.totalTime.get();
9  int completed = processed.incrementAndGet();
10 log.info("Completed: " + completed + " tasks, avg time:"
11         + (timeTotal / completed) + " ms");
```

Výpis 4.1: Statistiky zpracování vláken

Na řádce 7 je vidět atomické přičtení doby zpracování požadavku k celkové době. Na řádce 9 je pak atomická inkrementace celkového počtu zpracovaných požadavků.

Ostatní pole nejsou sdílená mezi vlákny, ani přístupná zvenčí instance třídy `EngineWorker`, není proto třeba synchronizovat přístup k nim. V rámci vypsání statistik ohledně vláken je po každém dokončeném požadavku navíc vypsán počet požadavků aktuálně čekajících na obsloužení ve frontě `Executoru`.

### 4.3.3 Databáze

Databáze pro aplikaci je tvořena pěti tabulkami. Tabulky a vztahy mezi nimi jsou zachyceny na obrázku 4.7. Schéma tvoří pomyslnou hvězdu s centrální tabulkou `Assignment`, která reprezentuje právě jedno přiřazení pravidla k projektu a k této dvojici přidruženou akci, která má být vykonána po splnění pravidla. Model splňuje požadavky na znovupoužitelnost pravidel a akcí, které tak nemusí být opětovně vytvářeny pro jednotlivé oblasti projektu.

Vývoj a testování aplikace proběhlo s využitím databáze SQLite[4]. Pro reálné použití pak bude využita databáze DB2[5] od IBM, která je dodávána k systému CLM zdarma.

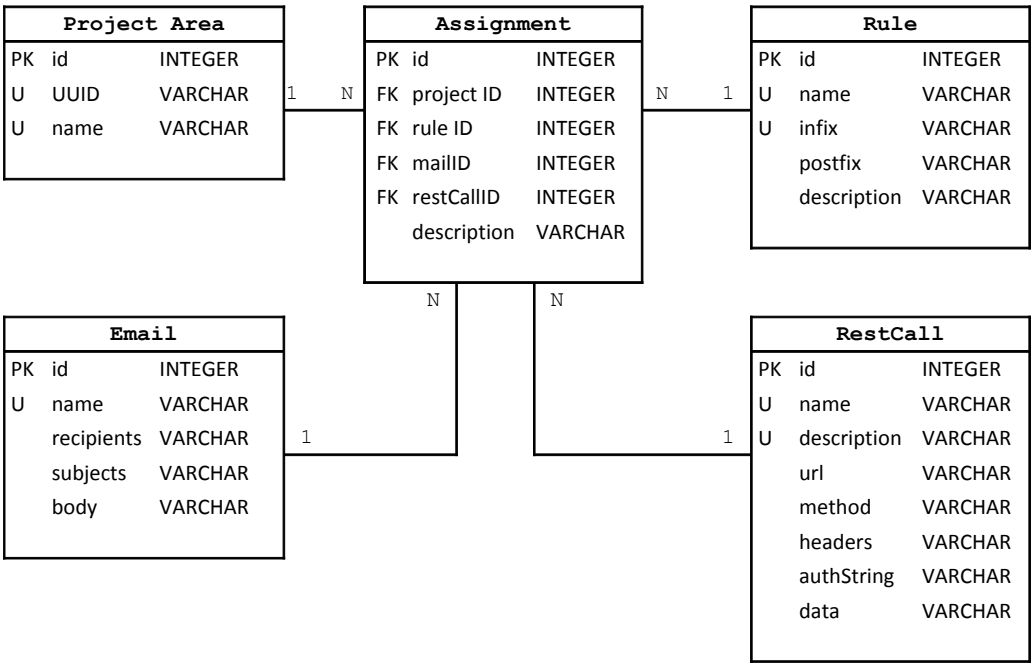
Jméno ovladače pro připojení, URI databáze a přihlašovací údaje jsou konfigurovatelné ze souboru `engine.properties`.

#### Připojení do databáze

Připojení do databáze může být časově náročná operace. Takže situace, kdy pro každý přístup do databáze je vytvářeno nové připojení, které je po skončení databázové operace uvolněno, je plýtvání prostředky, které může mít výkonnostní dopady na aplikaci. Z tohoto důvodu se používá tzv. *connection pooling*. Tento mechanizmus vytvoří „zásobník“ znovupoužitelných spojení do databáze. V případě potřeby je připojení poskytnuto uchazeči. Po skon-

[4] Jednoduchá souborová databáze `https://www.sqlite.org/`.

[5] Relační databázový systém klasické architektury klient-server. `https://www.ibm.com/analytics/cz/cs/technology/db2/`




Obrázek 4.7: Relační schéma databáze

čení práce připojení není uvolněno, ale může být použito dalším žadatelem. Jedná se o podobný princip jako dříve zmiňovaný *thread pooling*.

### Popis tabulek

- *ProjectArea* – tabulka reprezentuje Oblast projektu, ke které jsou přiřazována pravidla.
- *Rule* – tabulka obsahuje jméno a popis pravidla. Hlavním obsahem tabulky jsou samotná pravidla, a to jak v čitelné podobě pro uživatele – v tzv. infixovém tvaru, tak i v postfixovém tvaru, který umožňuje pravidla vyhodnotit.
- *Email* – tabulka obsahuje šablony emailových notifikací.
- *RestCall* – tabulka obsahuje konfigurace volání `REST API`.
- *Assignment* – přiřazení pravidla k Oblasti projektu a spřažení s akcí, která bude vyvolána při splnění pravidla. Tabulka obsahuje cizí klíče do ostatních čtyř tabulek.

### 4.3.4 Přístup do databáze (DAO)

Pro přístup k databázi je využit návrhový vzor Data access object (DAO). Návrhový vzor využívá rozhraní pro přístup k datům v databázi, souborovém systému a jiným zdrojům dat, aniž by se klient musel starat o to, jak jsou data uložena a jak je z úložiště dostat. Použitím DAO návrhového vzoru je odstíněna aplikační logika programu od zdrojů dat a veškerá logika přístupu k datům je shromážděna do jedné vrstvy. Konkrétní implementace DAO rozhraní mohou být pak odlišné pro různé zdroje dat.

Diagram tříd z balíku `dao` je zachycen na obrázku 4.8. Hlavním prvkem je generické rozhraní `IGenericDao<T>`, které specifikuje společné operace pro všechny tabulky databáze. Generické rozhraní implementuje abstraktní třída `GenericDaoJdbc<T>`. V abstraktní třídě se nachází implementace operací odstranění záznamu z tabulky `remove(int)` a nalezení záznamu podle identifikátoru `findOneById(int)`. Tyto metody jsou pak dále děděny a je zamezeno opakování kódu.

Generické rozhraní je rozšířeno specifickými rozhraní o další operace. Například `IRuleDao` definuje dvě operace navíc, například chceme mít možnost najít nejen podle identifikátoru, ale také podle oblasti projektu, ke které náleží. Třídy implementující jednotlivá rozhraní dědí od `GenericDaoJdbc<T>` výše zmíněné metody a implementují specifická rozhraní pro danou entitní třídu. U těchto tříd nejsou pro větší přehlednost v diagramu uvedeny metody.

### 4.3.5 Webové uživatelské rozhraní

Pro implementaci webového uživatelského rozhraní administrační části aplikace byl použit frontendový framework Bootstrap[6]. Jako základ vzhledu stránek byla použita šablona dostupná na `https://startbootstrap.com/template-categories/all/`. Nepředpokládá se využití aplikace na mobilních zařízeních, proto nebyl kladen důraz na responsivitu stránek.

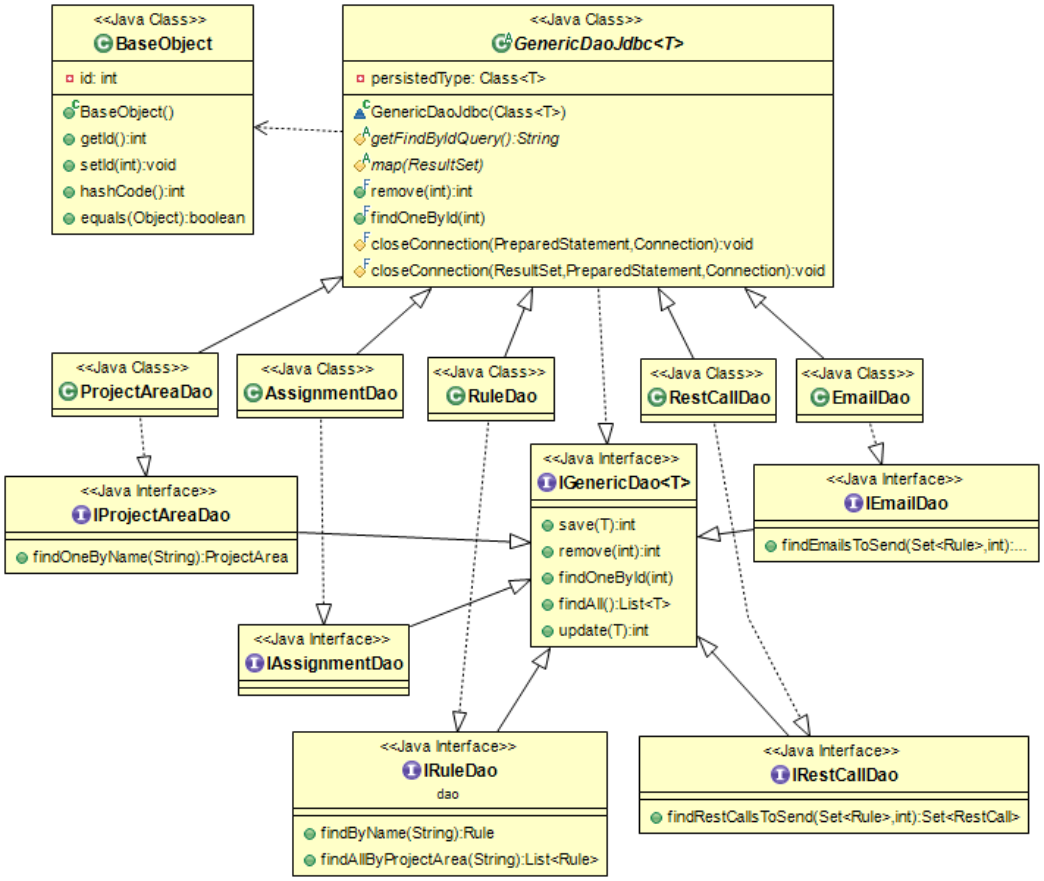
Webové rozhraní se skládá z osmi stránek a poskytuje funkce uvedené na obrázku 4.3. V levé části všech stránek se nachází navigační menu, které slouží jako rozcestník do ostatních částí. Webové stránky jsou `HTML` stránky doplněné o dynamicky generovaný obsah za využití technologie `JSP`.

Při vývoji byla také použita knihovna `JSTL`[7], která rozšiřuje značky `JSP` o běžné často opakující se operace, jako jsou iterace v kolekcích, větvení na základě podmínek, značky pro práci s databází, apod. K usnadnění přístupu k datům v aplikaci jsou použity tzv. `EL`[8] výrazy. Ukázka uživatelského

[6] https://getbootstrap.com/

[7] JavaServer Pages Standard Tag Library

[8] Expression Language

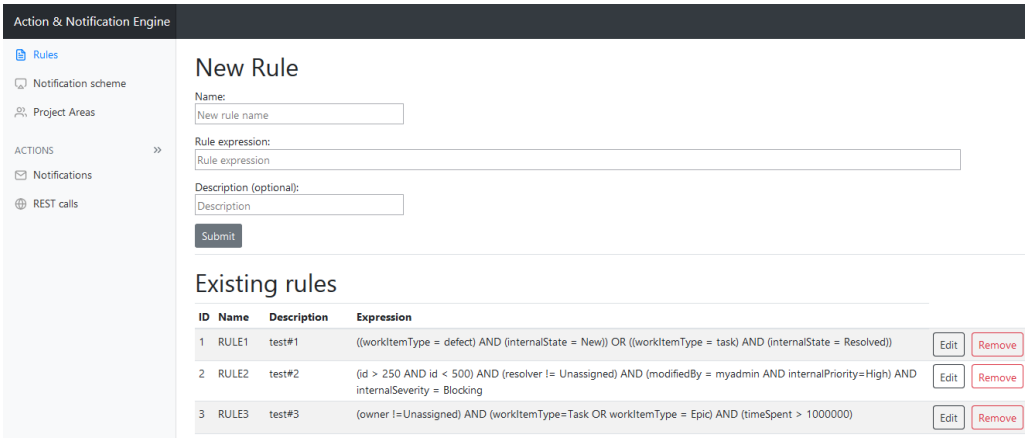



Obrázek 4.8: Diagram tříd balíku DAO

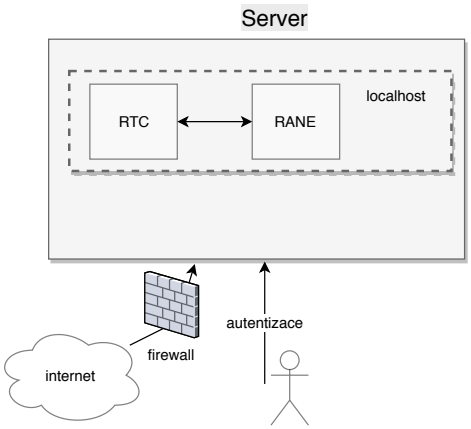
rozhraní se nachází na obrázku 4.9.

### 4.3.6 Zabezpečení

Aplikace bude nasazená v servletovém kontejneru a díky firewallovým pravidlům nebude přístupná mimo localhost. K aplikaci má tedy přístup jen uživatel, který je v momentě přístupu již autentizován. Není tedy nutné implementovat další zabezpečení. Situace je znázorněna na obrázku 4.10.




Obrázek 4.9: Uživatelské rozhraní




Obrázek 4.10: Přístup k aplikaci – zabezpečení

# 5 Implementace

V následující kapitole je popsána implementace pluginu a notifikačního engine. Dále je popsán mechanizmus ukládání a vyhodnocování pravidel, poté jsou popsány oba druhy spustitelných akcí.

## 5.1 RTC Java Client API

RTC Java Client API je Java API založené na Eclipse frameworku. Prostřednictvím tohoto API je možné přistupovat k datům uloženým v RTC. Z výkonnostních důvodů není vždy přístup k datům jednoduchý a přímočarý. Ve většině případů data nejsou „vystavena“ přímo, ale jsou napřed získány tzv. *handly*, pomocí kterých jsou dosažitelná konkrétní data. Příkladem je základní entita v RTC – pracovní položka a její atributy. Po programovém vyhledání pracovní položky objekt, který ji reprezentuje, sám o sobě nabízí pouze základní informace, jako je ID, datum vytvoření nebo termín vyřešení.

Pro hodnoty dalších atributů poskytuje objekt pracovní položky pouze *handly*, ze kterých je možné přistoupit k požadovaným datům pomocí specifických rozhraní jednotlivých typů atributů.

## 5.2 RTC Server plugin

Rozšíření funkčnosti RTC na straně serveru se realizuje přidáním tzv. Předběžných podmínek nebo Následných akcí[1]. Jedná se o Eclipse plugin, který je uživatelem přiřazený k určité oblasti projektu. Činnost pluginu je pak vykonána vždy, když dojde k uložení pracovní položky v rámci dané oblasti projektu.

Plugin se skládá ze tří komponent, každá z nich je Eclipse projekt:

- *Plugin project* – vlastní výkonný kód pluginu.
- *Feature project* – slouží k popisu souvisejících komponent a vyžadovaných závislostí na ostatních komponentách. *Feature* je popsána souborem `feature.xml`.
- *Update site* – repozitář pro *features* a *pluginy* – slouží k sestavení pluginu.

[1] Preconditions, Follow-up actions

Výsledkem sestavení pluginu jsou JAR soubory reprezentující *feature* a *plugin*.

Ke každé oblasti projektu, která bude chtít využívat notifikační engine, musí být nasazen *Rational Action and Notification Engine Caller*. Jeho jedinou funkcí je odeslání HTTP požadavku do notifikačního engine a spustit tak proces nalezení a vyhodnocení pravidel.

#### Časový limit

Ukládání pracovní položky nesmí být blokováno příliš dlouhou dobu. Poté, co plugin odešle HTTP požadavek notifikačnímu engine, čeká na odpověď. Při případném zpoždění nebo nedostupnosti aplikace je programově zajištěno, že po stanoveném časovém limitu bude HTTP požadavek terminován a nedojde tak k blokování ukládání pracovní položky. Časový limit je konfigurovatelný z `properties` souboru `conf.properties`. Programová realizace časového limitu je uvedena na následujícím výpise.

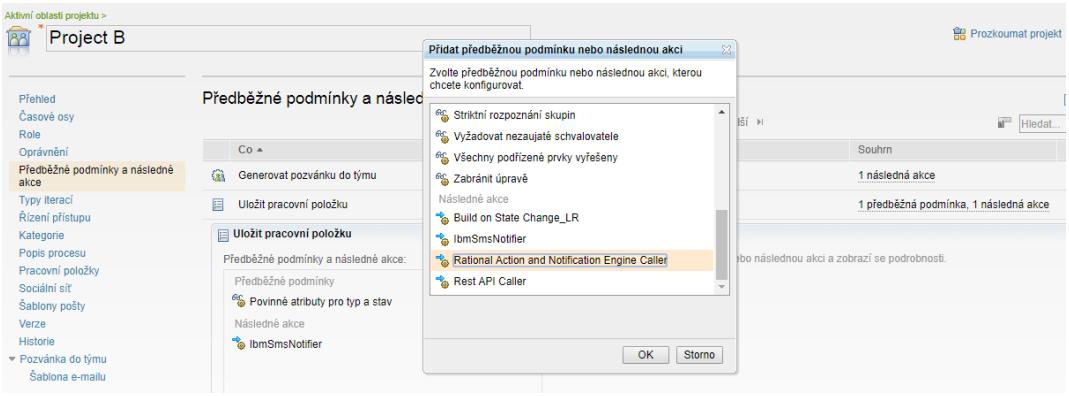
```
1  final HttpGet request = new HttpGet(uri);
2  TimerTask task = new TimerTask() {
3      @Override
4      public void run() {
5          if (request != null) {
6            request.abort();
7          }
8      }
9  };
10 new Timer(true).schedule(task, TIMEOUT);
11 HttpResponse response = client.execute(request);
```

Výpis 5.1: Časový limit HTTP požadavku

### 5.2.1 Nasazení

- V projektu Update site sestavit plugin v souboru `site.xml` stiskem tlačítka `Build all`.
- Zkopírovat vygenerované adresáře features a plugins společně se souborem `site.xml` do adresáře, kde je nainstalovaný Jazz Team Server, konkrétně do adresáře `sites`.
- Do adresáře `/server/conf/ccm/provision_profiles` vložit soubor provision profile. Soubor slouží serveru pro identifikaci komponent, které se má pokusit nasadit.

- Restartovat server.
- V konfiguraci oblasti projektu v sekci Předběžné podmínky a následné akce vložit novou Následnou akci. V seznamu dostupných rozšíření jsou zobrazena jména pluginů, viz obrázek 5.1.
- Uložit nastavení.




Obrázek 5.1: Nasazení pluginu do RTC

## 5.3 Action & Notification Engine

Kapitola popisuje jádro notifikačního engine. Vysvětlena bude reprezentace pravidel, způsob jejich uložení, převod do formy, která umožňuje pravidla vyhodnotit a mechanizmus vlastního vyhodnocení.

### 5.3.1 Struktura aplikace

Aplikace notifikačního engine se skládá z následujících balíčků:

- `actions`: třídy spojené s vykonáváním akcí.
- `attribute_resolver`: třídy sloužící k získání hodnot atributů z pracovní položky.
- `config`: třída, která načte konfigurační soubor.
- `core`: třídy spojené s vyhodnocováním pravidel pro spouštění akcí.
- `dao`: třídy DAO vrstvy. Třídy poskytují přístup k datům z databáze.

- `domain`: třídy vrstvy modelu. Na tyto třídy jsou mapována data z databáze.
- `domain.actions`: třídy vrstvy modelu související s akcemi. Reprezentují data z databáze.
- `servlet`: servlety pro administrační část aplikace.
- `servlet.async`: balíček obsahuje asynchronní servlet, který přijímá požadavky od pluginu a vkládá nové úlohy do fronty požadavků. Také obsahuje třídu pro vykonání celého procesu od nalezení pravidel po spuštění akcí.
- `test`: jednotkový test vyhodnocování pravidel.
- `utils`: pomocné třídy. Třída následuje návrhový vzor jedináček.

### 5.3.2 Atributy

Atributy jsou nositeli informací o pracovní položce. Existují různé typy atributů. Mezi základní typy patří např. řetězec, logická hodnota, časová značka nebo číselné typy. Aby mohl být atribut používaný v pravidlech, musí jeho hodnota být snadno reprezentovatelná textovým řetězcem. V této části práce bude popsáno získání hodnot atributů z RTC a typy atributů, které mohou být součástí pravidel.

#### Získání atributu z úložiště

RTC nabízí pro přístup k pracovním položkám a jejich atributům API. Přístup k hodnotám atributů základních typů je vcelku snadný. Nad objektem reprezentujícím pracovní položku se zavolá metoda `Object getValue(IAttribute attribute)`. Nad získaným objektem poté stačí zavolat metoda `String toString()`, nebo v případě řetězcových typů přetypovat hodnotu na `String`. Příklad získání `String` hodnoty atributu pro řetězcové typy je uveden na následujícím výpise.

```
1 if (workItem.hasAttribute(attributeId)){
2   Object value = workItem.getValue(attributeId);
3   if (value instanceof String) {
4     String attributeValue = (String) value;
5     return attributeValue;
6   }
7 }
```

Výpis 5.2: Získání hodnoty atributu typu řetězec

Obdobně u číselných typů stačí rozhodnout příslušnost objektu `value` k odpovídajícímu datovému typu v Javě a zavolat nad objektem metodu `toString()`.

Kromě takových základních typů existují také komplexnější typy atributů, u kterých je získání hodnoty, která bude vhodně reprezentovat atribut, obtížnější. Jako příklad bude uveden atribut *Priorita*, který nabývá výčtového typu *Priorita*[2]. Pokud by měla pracovní položka hodnotu tohoto atributu např. „Vysoká“, tak by metoda `workItem.getValue(attributeId)` vrátila místo hodnoty „Vysoká“ například „com.ibm.team.workitem.common.model.IPriority:priority.literal.l11“, což je identifikátor literálu „Vysoká“. Taková hodnota je pro použití v pravidlech nevhodná. Metoda, která vrací hodnotu atributu v čitelném tvaru je uvedena v následujícím výpise. Z důvodu přehlednosti je metoda zkrácena o ošetření chybných vstupů.

```
1   private String calculateEnumerationAsString(Object value,
2       IAttribute attribute) throws TeamRepositoryException {
3
4     IEnumeration<? extends ILiteral> enumeration = wiCommon
5         .resolveEnumeration(attribute, getMonitor());
6
7     Identifier<? extends ILiteral> id =
8         (Identifier<? extends ILiteral>) value;
9
10    ILiteral literal = enumeration
11        .findEnumerationLiteral(id);
12    return literal.getName();
13  }
```

Výpis 5.3: Získání hodnoty atributu typu výčet

Metoda očekává na vstupu hodnotu atributu a *handle* atributu. Pomocí služby z RTC client API `wiCommon` je získán kompletní výčet literálů atributu Priorita. Poté je na základě identifikátoru získán literál a nakonec je vráceno jméno literálu, které je použitelné v pravidlech.

Určitou část typů reprezentovat řetězcem v přijatelné podobě nejde. Po prozkoumání vlastností jednotlivých typů bylo nutné vybrat z množiny existujících typů typy, které budou podporované v notifikačním engine.

K získání hodnoty atributů reprezentované řetězcem byla jako základ použita třída `ExportWorkItemsCommand` z nástroje RTC WorkItem Command Line (WCL). Nástroj WCL[9] umožňuje z příkazové řádky vytvářet,

[2]Výčtový typ nabývá omezené množiny pojmenovaných hodnot. Identifikátor atributu Priorita je `internalPriority`.

| Operátor | Název | poznámka |
|---|---|---|
| = | Rovná se | - |
| != | Nerovná se | - |
| > | Větší než | Pouze pro číselné typy |
| < | Menší než | Pouze pro číselné typy |
| AND | Logický součin | Pouze mezi výrazy |
| OR | Logický součet | Pouze mezi výrazy |

Tabulka 5.1: Podporované operátory

upravovat pracovní položky v RTC, importovat a exportovat pracovní položky použitím CSV souborů.

### 5.3.3 Pravidla

Hlavní ideou vyvíjeného systému je možnost definovat pravidlo, které když je splněno, je spuštěna nějaká akce. Pravidlem se rozumí výraz složený z názvů atributů pracovní položky, očekávaných hodnot atributů, závorek, některých logických operátorů a z relačních operátorů.

#### Syntaxe pravidel

Základním prvkem pravidel je trojice: `jméno_atributu` *operátor* `hodnota_atributu`. Výraz složený z takové trojice porovnává hodnotu atributu pracovní položky se specifikovanou hodnotou. Výstupem porovnání je logická hodnota `true` nebo `false`. *Jméno atributu* je identifikátor atributu v rámci oblasti projektu, ke které pracovní položka patří. V průběhu vyhodnocování pravidla je za jméno atributu dosazena jeho skutečná hodnota a ta je porovnána s *hodnotou atributu*. *Operátor* specifikuje typ logické operace, která je aplikovaná na dvojici hodnot. Akceptované operátory jsou uvedeny v tabulce 5.1.

Operátor „rovná se“, podle očekávání, porovná dvě hodnoty a pokud jsou totožné, je výsledek porovnání `true`. Analogicky operátor „nerovná se“ poskytuje výsledek `true` v případě odlišných hodnot. Operátory „větší“ a „menší“ lze uplatnit pouze na celočíselné typy.

Aby bylo možné vyhodnocovat atributy řetězcových typů, které mohou nabývat hodnot složených z více slov, je nutné hodnoty atributů ohraničit rovnými uvozovkami (`"`), podobně jako řetězcové konstanty u programovacích jazyků.

Výrazy je možné dále skládat do složitějších struktur jejich uzávorkováním a použitím logických operátorů `AND` a `OR`. Následuje ukázka tří pravidel.

```
1. (workItemType = "defekt") AND (internalState = "Nový")

2. (id > "250"AND id < "500") AND (resolver != "Una-
   ssigned")

3. (workItemType = "Úloha") OR ((workItemType = "Epic") AND
   (internalPriority = "Nízká"))
```

Výpis 5.4: Ukázka pravidel

První pravidlo je vyhodnoceno jako splněné v případě, že zkoumaná pracovní položka je typu „defekt“ a její stav je „Nový“. Druhé pravidlo splňují pracovní položky s hodnotou identifikátoru mezi 250 a 500, přičemž atribut „řešitel“ nesmí mít hodnotu „nepřiřazeno“. Třetí pravidlo je kladně vyhodnoceno pro všechny pracovní položky typu „Úloha“ a pro položky typu „Epic“, jejichž priorita není „Nízká“.

### 5.3.4 Podporované atributy

V této části se nachází přehled typů, kterých mohou nabývat atributy používané v pravidlech. Seznam podporovaných typů je uveden v tabulce 5.2. První sloupec tabulky představuje typ atributu, druhý sloupec obsahuje příklad atributu daného typu, respektive jméno atributu. V posledním sloupci se nachází identifikátor atributu. Právě identifikátory atributů mohou být používány v pravidlech.

Pro typy atributů uvedených v tabulce 5.2 bylo testováním ověřeno získání jejich `String` hodnoty.

## 5.4 Operace s pravidly

V této části bude popsána implementace ukládání pravidel a jejich vyhodnocování.

### 5.4.1 Vytváření pravidel

Pravidlo vytváří uživatel prostřednictvím webového rozhraní. Uživatel zadává pravidla v přirozeném tvaru s operátorem mezi operandy, v tzv. *infixovém* tvaru. Aby bylo možné pravidlo vyhodnotit, je nutný jeho převod

| Typ atributu | Název atributu | Id atributu |
|---|---|---|
| Boolean | Archivováno | archived |
| html | Popis | description |
| Integer | ID | id |
| Kategorie (oblast projektu) | Zařazené vzhledem k | category |
| Malý řetězec | Stav | internalState |
| Obecný výčet | – | – |
| Priorita (výčet) | Priorita | internalPriority |
| Přispěvatel | Vlastník | creator |
| Release | Nachází se v | foundIn |
| Timestamp | Datum vytvoření | creationDate |
| Typ | Typ | workItemType |
| Výčet | Priorita | internalPriority |
| Závažnost (výčet) | Závažnost | internalSeverity |
| Značka | Značky | internalTags |

Tabulka 5.2: Podporované typy atributů

do postfixového tvaru. Postfixová notace má navíc tu vlastnost, že k vyhodnocení výrazů nejsou potřeba závorky. Následuje ukázka algoritmu převodu pravidla do postfixové notace. Algoritmus využívá tři datové struktury:

1. fronta tokenů v infixovém tvaru (vstup),
2. zásobník,
3. řetězec tokenů v postfixovém tvaru.

```
1  vstup:= retezec, pravidlo v infixovem tvaru
2  vystup:= seznam tokenu v postfixovem tvaru
3
4  fronta = ziskej frontu tokenu(vstup)
5
6  WHILE fronta.size != 0:
7      token = fronta.pop();
8      IF token.type=terminal
9          vystup.add(token)
10
11     IF token.type=operator //o1
12      dokud je na zasobniku operator (o2), tak pokud je
13      priorita(o1) >=  priorita(o2), vyjmi operator o2
14      ze zasobniku a vloz ho do vystupniho seznamu.
```

```
15          zasobnik.push(op1)
16      IF token.type = L_zavorka
17          zasobnik.push(token)
18      IF token.type = P_zavorka
19          WHILE stack.top() != L_zavorka
20              vystup.add(stack.pop()) //vloz operatory
21          END WHILE
22  END WHILE
23
24  //vloz zbytek obsahu zasobniku do vyst. seznamu
25  WHILE stack.size() !=0:
26          vystup.add(stack.pop())
27  END WHILE
28  RETURN vystup;
```

Výpis 5.5: Převod infixového pravidla na postfix

Algoritmus pro převod je naznačen na výpisu 5.4.1. Seznam tokenů v postfixovém tvaru je nakonec převeden na řetězec, který může být následně uložen do databáze. Jednotlivé tokeny je nutné oddělit oddělovačem. Jako oddělovač byla zvolena dvě zpětná lomítka: „\\“. Příklad pravidla a jeho různých forem následuje:

Infixový tvar:
`(workItemType = "Úloha") OR ((workItemType = "Epic") AND (id > "200"))`

Seznam tokenů v infixovém tvaru:
`L_PAREN, workItemType, EQUAL, "Úloha", R_PAREN, OR, L_PAREN, L_PAREN, workItemType, EQUAL, "Epic", R_PAREN, AND, L_PAREN, id, GREATER, "200", R_PAREN, R_PAREN`

Seznam tokenů v postfixovém tvaru:
`workItemType, "Úloha", EQUAL, workItemType, "Epic", EQUAL, id, "200", GREATER, AND, OR`

Do databáze je pravidlo uloženo ve formátu, kdy jednotlivé tokeny v postfixovém tvaru jsou navíc odděleny oddělovačem.

### Převod do Postixové notace

Pro práci s pravidly slouží rozhraní `IRuler`. Rozhraní deklaruje tři metody

Metoda `convertRuleToPostfixString` převádí pravidlo z infixové notace do postfixové. Vstupem i výstupem je řetězec. Metoda je volána po vytvoření nového pravidla, nebo editaci již existujícího pravidla. Po vytvoření

pravidla ve webovém formuláři je voláním této metody pravidlo převedeno na tvar vhodný k uložení do databáze.

Druhá metoda `convertPostfixStringToTokenList` je volána při vyhodnocování pravidel. Z databáze je pravidlo načteno jako řetězec s tokeny oddělenými oddělovačem. Metoda rozdělí řetězec podle oddělovače a určí typ tokenu jednotlivých slov.

Pravidlo reprezentované seznamem tokenů v postfixové formě je již možné vyhodnotit.

### Vyhodnocování pravidel

V této části je popsán algoritmus vyhodnocení pravidel. Pro vyhodnocení je použita datová struktura zásobník.

```
Vstup: řetězec - seznam tokenů v postfixové formě.
Tokeny jsou odděleny oddělovačem.
Výstup: pravdivostní hodnota true/false.

seznam := převeď řetězec pravidla na seznam tokenů

přes všechny tokeny v seznamu:
    pokud je token terminál:
       vlož token do zásobníku
    pokud je token operátor:
       prověď logickou operaci nad dvěma
       terminály z vrcholu zásobníku a
       výsledek vlož do zásobníku

pokud v zásobníku není právě jeden prvek, zaloguj chybu,
jinak vyjmi hodnotu ze zásobníku. To je výsledek
vyhodnocení pravidla.
```

## 5.5 Akce

V minulých kapitolách byl popsán způsob reprezentace pravidel, jejich uložení a způsob jejich vyhodnocování. V této části budou popsány akce, jež jsou důsledkem kladně vyhodnocených pravidel.

V současné podobě systém umožňuje použít dva druhy akcí:

1. Volání `RESTful` API.

2. Emailová notifikace.

Programový kód byl vytvářen s ohledem na možné rozšíření především o možnost přidání dalšího druhu akce.

### 5.5.1 Volání REST API

První ze jmenovaných akcí byla představena jako volání RESTful API. Ve skutečnosti je ale možné nakonfigurovat odesílání libovolného HTTP požadavku. Hlavní předpokládané využití však spočívá ve volání API **UrbanCode Deploy** serveru.

#### Konfigurace akce

Administrační část webové aplikace umožňuje nakonfigurovat libovolné volání pomocí protokolu HTTP. Uživatel ve webovém formuláři vloží následující údaje pro vytvoření HTTP požadavku:

- *URL* – Adresa volaného zdroje.
- *Metoda* – Metoda přístupu ke zdroji, zároveň metoda odesílaného HTTP požadavku.
- *Hlavičky* – hlavičky volaného HTTP požadavku. Každou hlavičku je třeba přidat na novou řádku vstupního pole ve formuláři. Hlavička obsahuje jméno a obsah. Tyto dvě hodnoty jsou odděleny dvojtečkou.
- *Uživatelské jméno a heslo* – Uživatelské jméno a heslo slouží k autentizaci pro volání REST příkazů. Jméno a heslo jsou zakódovány *Base64* kódováním a výsledný řetězec je nastaven jako hodnota hlavičky `Authorization`. Pokud cílový server neakceptuje přihlašovací token, tak server vrací chybový kód `401`. Pro obě položky se ve formuláři nachází samostatné pole.
- *Data* – tělo požadavku, které obsahuje data. Při volání REST API se jedná typicky o zprávu ve formátu JSON, nebo XML.

Do pole „Data“ je možné vkládat hodnoty atributů z pracovní položky. Pokud typ atributu patří mezi podporované a pokud ho pracovní položka obsahuje v odeslané zprávě je zobrazena hodnota atributu. Tím je možné konfigurovat obsah JSON zprávy na základě hodnot atributů pracovní položky. Atribut se vloží do těla zprávy ve formátu: `${id_atributu}`.

Ke snadnější identifikaci nakonfigurovaného volání je součástí formuláře také jméno konfigurace a volitelně popis toho, jaký je účel volání.

#### Použití

Typická situace pro využití akce tak může vypadat například takto: Pokud je uživatelem změněn stav pracovní položky typu `Build and deployment` na stav *Deploy*, spusť aplikační proces na konkrétním UCD serveru, který vyvolá nasazení konkrétního artefaktu na požadované prostředí.

#### Bezpečnost v UCD

Použití Action & Notification Engine umožňuje vzdáleně ovládat UCD, respektive odesílat libovolné HTTP požadavky na server. Bezpečnost na straně UCD je ošetřena tím, že pro každý projekt bude existovat konkrétní uživatel, který bude mít omezená práva. Takový uživatel tak může mít povoleno spouštět pouze procesy „Run deployment“ jen na konkrétní prostředí.

### 5.5.2 Emailová notifikace

Druhou implementovanou akcí, která může být vyvolána, je odeslání emailové notifikace na dané emailové adresy. Implementována je třída `BasicMailer`, která demonstruje odesílání notifikací. Pro odesílání emailů využívá balíček `javax.mail`.

#### Konfigurace akce

Administrační část webové aplikace umožňuje nakonfigurovat šablonu emailové notifikace. Při vytváření šablony uživatel ve webovém formuláři vkládá následující údaje:

- *Seznam příjemců* – seznam emailových adres, na které bude notifikace odeslána při vyvolání akce. Jednotlivé emailové adresy jsou odděleny čárkou.
- *Předmět zprávy* – Předmět odesílané emailové zprávy.
- *Tělo zprávy* – obsah notifikace. Do těla zprávy je možné vkládat atributy z pracovní položky, viz dále.

Ke snadnější identifikaci šablony notifikace je součástí formuláře také jméno šablony.

### Parametrizace zpráv

Implementace umožňuje parametrizovat obsahy notifikací. Do těla emailové zprávy je možné vkládat hodnoty atributů pracovní položky. Atributy a jejich hodnoty jsou získány před vyhodnocováním pravidel. Pokud typ atributu patří mezi podporované a pokud ho pracovní položka obsahuje, tak je hodnota atributu v době vykonávání akce dostupná. Hodnota atributu se vloží do těla zprávy vložením identifikátoru atributu ve formátu: `${id_atributu}`. Příklad notifikační šablony a skutečně odeslané zprávy následuje: Hodnotou atributu může být nastaveno také pole příjemců emailu.

```
Dear Colleagues,
within the Project area \${category} there an issue occured.
Work Item id=\${id}
category=\${category}
creationDate=\${creationDate}
creator=\${creator}
internalPriority=\${internalPriority}
internalSeverity=\${internalSeverity}
internalState=\${internalState}
summary=\${summary}
workItemType=\${workItemType}

Please take an appropriate action
```

Do výše uvedené šablony jsou dosazeny skutečné hodnoty atributů:

```
Dear Colleagues,
within the Project area Project A there an issue occured.
Work Item id=200  :
category=Project A
creationDate=2018-03-28 19:46:12.649
creator=myadmin
description=testování 24.4.2018
internalPriority=Vysoká
internalSeverity=Blokování
internalState=Nový
summary=testovani 24.4.2018
workItemType=Defekt

Please take an appropriate action
```

# 6 Ověření řešení

## 6.1 Jednotkové testy

Součástí programu je i jednotkový test umístěn ve třídě `EvalTest.java`. Test usnadnil ověření správného vyhodnocování pravidel a převodů pravidel do postfixové formy. Test byl vytvořen tak, aby zahrnul všechny podporované operátory, které se mohou v pravidlech vyskytnout.

## 6.2 Testování pluginu

Na straně pluginu bylo ověřeno, že při případném zpoždění nebo nedostupnosti notifikační webové aplikace, bude po stanoveném časovém limitu pokus o spojení ukončen a nedojde tak k blokování ukládání pracovní položky. Informace o neúspěchu volání notifikačního engine je na straně pluginu zaznamenána do logu.

## 6.3 Funkční testování

V této části bude popsáno použití aplikace a ověření výsledků v rámci kompletního případu užití: Vytvoření pravidla, vytvoření šablony spouštěné akce a přiřazení pravidla s akcí k oblasti projektu. Poté vytvoření pracovní položky, které vyvolá spuštění pluginu, který odešle požadavek do notifikačního engine, ověření vyhodnocení pravidel a vykonání požadovaných akcí.

### 6.3.1 Test volání UCD

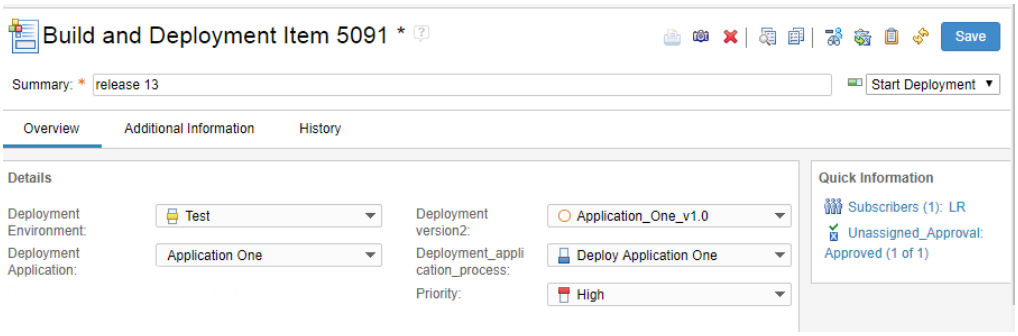
**Očekávaný výsledek testu:**

Spuštění procesu nasazení aplikace „Application One“ na prostředí „Test“ po přechodu pracovní položky typu Build and Deployment ze stavu *Draft* do stavu *Deploy* a pokud priorita pracovní položky není nízká – „Low“. Proces v UCD musí být úspěšně dokončen a do pracovní položky musí být vložen komentář o úspěšném nasazení.

### Příprava v RTC

V RTC pro oblast projektu „DEMO RTC-GSN“ mějme, nebo vytvořme pracovní položku typu „Build and Deployment Item“. Pro pracovní položku vytvoříme vlastní atributy *deployment_environment, deployment_version, deployment_application a deployment_application_process*. Tyto atributy budou definovat spustitelný proces v UCD. K atributům vytvoříme výčtové typy s hodnotami, jichž mohou atributy nabývat. K jednotlivým atributům přiřadíme odpovídající výčty a přidáme atributy do prezentace editoru. Pokud by se tak nestalo, atributy by sice byly přiřazeny k pracovní položce, ale v uživatelském prostředí by nebyly vidět při zakládání pracovní položky.

Pracovní položka ve stavu před uložením je zachycena na obrázku 6.1.




Obrázek 6.1: Pracovní položka před uložením

### Příprava v UCD

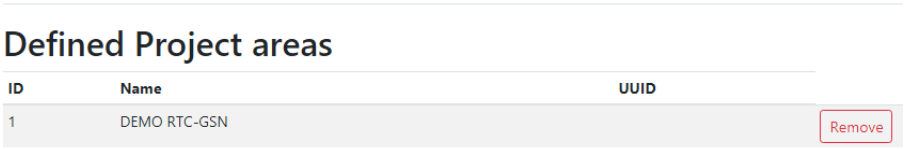
V UCD připravíme Aplikační proces, který po spuštění vyvolá nasazení konkrétní komponenty na požadované prostředí.

### Příprava v notifikačním engine

V prvním kroku je nutné vytvořit v aplikaci reprezentaci **oblasti projektu**. Oblastem projektu jsou přiřazována jednotlivá pravidla a k nim vázané akce. Vytvoření oblasti projektu se provede na adrese `http://localhost:8080/notifications/areas`[1][2]. Jednoduchý webový formulář obsahuje jedno povinné pole, a to název oblasti projektu. Ten musí být shodný s názvem oblasti projektu v RTC.

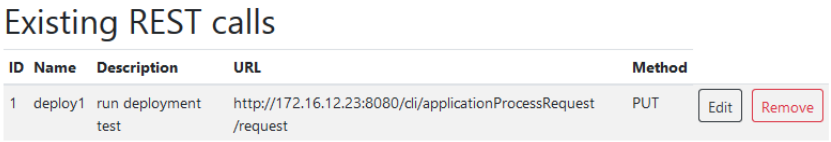
[1]Dále bude uváděna pouze namísto celé URL včetně kontextu a domény pouze konkrétní stránka.

[2]Doména, port a kontext se samozřejmě liší podle místa a způsobu nasazení.



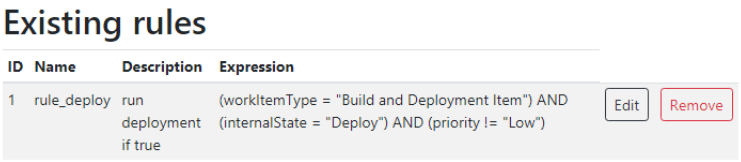

Obrázek 6.2: Vytvořený záznam pro oblast projektu

V druhém kroku bude vytvořena **šablona pro vytvoření HTTP požadavku**, který je odeslán na požadovaný daný server. Vytvoření šablony se provede na adrese `/rest`. Ve formuláři jsou postupně zadány údaje, které byly popsány v kapitole 5.5.1. Formulář s příkladem vyplněných hodnot se nachází v příloze A. Výsledek uložení šablony je znázorněn na obrázku 6.3.




Obrázek 6.3: Vytvořená šablona volání REST

Dalším krokem je **vytvoření pravidla**, které bude ověřovat typ pracovní položky, její stav a prioritu. Pravidlo bude kladně vyhodnocené pouze v případě, že pracovní položka je typu „Build and Deployment Item“, je ve stavu „Deploy“ a priorita není „Low“. Vytvoření pravidla se provede na adrese `/rules`.

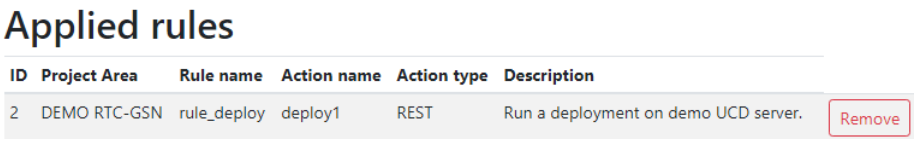



Obrázek 6.4: Uložené pravidlo

Poslední krok vykonaný v notifikačním engine je **přiřazení pravidla k oblasti projektu** a nastavení akce, která bude vyvolána po splnění pravidla. Ve formuláři dostupném na adrese `/scheme` vybereme oblast projektu „DEMO RTC-GSN“. K oblasti přiřadíme pravidlo a akci vytvořené v předchozích bodech. Výsledek přiřazení je zobrazen na obrázku 6.5.

### Spuštění

V RTC vytvoříme pracovní položku typu Build and Deployment a nastavíme jí požadované hodnoty atributů pro volání UCD. Po jejím uložení je pra-




Obrázek 6.5: Aplikované pravidlo: oblast projektu, pravidlo a akce.

covní položka stále v počátečním stavu „Draft“. Uložením pracovní položky je spuštěn Notificaion Engine Caller plugin, který spustí notifikační engine. Engine vyhodnotí pravidlo negativně, pracovní položka se nenachází v požadovaném stavu. V dalším kroku pracovní položce nastavíme akci „Start deployment“. V tomto stavu je pracovní položka na obrázku 6.1. Po uložení pracovní položky je tato převedena do stavu „Deploy“. Uložení pracovní položky opět spustí plugin. Tentokrát je pravidlo vyhodnoceno pozitivně.

### Ověření

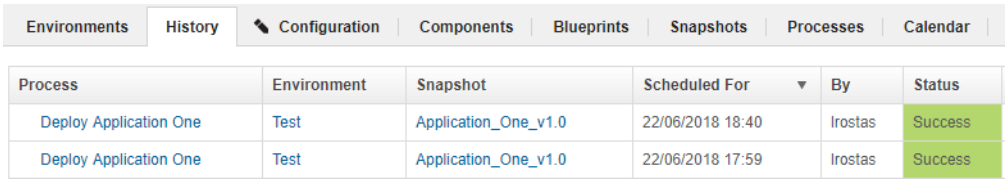
Kladné vyhodnocení pravidla by mělo spustit akci vytvořenou v předchozích bodech a v konečném důsledku spustit proces nasazení v UCD. Následuje výpis logu ze kterého je vidět, že došlo k odeslání HTTP požadavku a příjmu odpovědi.

Z logu je vidět, že bylo pozitivně vyhodnoceno pouze jedno pravidlo „rule_deploy“. K pravidlu byla vyhledána přiřazená akce „deploy“. Následuje spuštění akce a také obsah těla HTTP požadavku odeslaného do UCD. Je vidět, že parametry ve zprávě byly správně nahrazeny skutečnými hodnotami atributů. Černou barvou je pak vypsána odpověď od UCD serveru a informace o ukončení vlákna, které vykonalo kompletní workflow od získání atributů pracovní položky po spuštění akce.

```
INFO: ExecutorThread[ID=117::Name=http-bio-8080-exec-26::workItem=5091] evaluated rule id=1,name=rule_deploy with result: true
22.6.2018 18:40:53 servlet.async.EngineWorker run
INFO: ExecutorThread[ID=117::Name=http-bio-8080-exec-26::workItem=5091] fulfiled rules:
22.6.2018 18:40:53 servlet.async.EngineWorker run
INFO: 1:rule_deploy
22.6.2018 18:40:53 servlet.async.EngineWorker run
INFO: ExecutorThread[ID=117::Name=http-bio-8080-exec-26::workItem=5091] Action to perform: REST, deploy1
22.6.2018 18:40:53 actions.ActionHandler performAction
INFO: Performing action [type=REST, name=deploy1]
22.6.2018 18:40:53 actions.RestCallIer perform
INFO: HTTP request body: {
  "application": "Application One",
  "applicationProcess": "Deploy Application One",
  "environment": "Test",
  "onlyChanged": "false",
  "description": "Called from RANE.",
  "snapshot": "Application_One_v1.0",
  "properties": {
        "wiId": "5091",
  "rfcNumber":"RFC1268727"
  }
}
{"requestId":"bbb2b10a-6eb0-4b31-bac4-ec1c209f9df2"}
22.6.2018 18:40:54 servlet.async.EngineWorker run
INFO: ExecutorThread[ID=117::Name=http-bio-8080-exec-26::workItem=5091]thread finish in 550 ms.
```

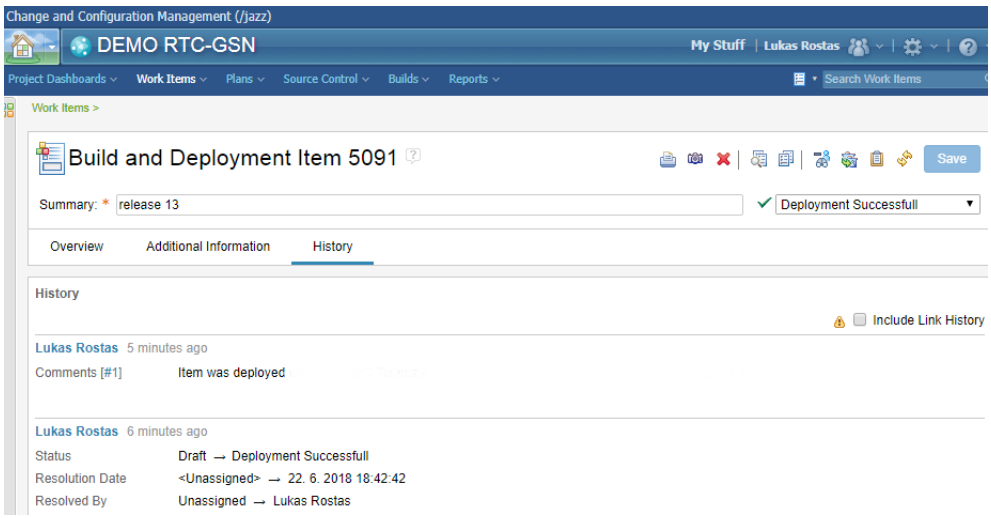
Obrázek 6.6: Evaluace pravidla a vyvolání akce

Zbývá ověřit, že proces v UCD byl skutečně spuštěn a úspěšně dokončen. Úspěšný proces je zobrazen na obrázku.




Obrázek 6.7: Výsledek procesu v UCD

UrbanCode Deploy může být nakonfigurován tak, aby vložil komentář do pracovní položky a dokonce změnil její stav. Po úspěšném vykonání procesu nasazení, UCD provedlo změny v pracovní položce, což je vidět na obrázku 6.8.




Obrázek 6.8: Zpráva o úspěšném nasazení v pracovní položce

## 6.4 Zátěžové testování

Pro zjištění výkonnostních limitů aplikace byla na straně RTC serveru provedena hromadná simultánní změna 20 pracovních položek. Každá změna v pracovní položce vyvolala Následnou akci v podobě volání Notification engine a vyvolání celého workflow uvedeného v kapitole 4.2. Opět bylo ověřeno, že jako splněná byla vyhodnocena správná pravidla a že byly úspěšně provedeny požadované akce.

# 7 Možnosti rozšíření

Aplikace byla navržena s ohledem na možné budoucí rozšíření. Tato kapitola představuje některá z nich.

### Monitoring událostí

V současné podobě aplikace zaznamenává svoji činnost do logu. K tomu je použita knihovna `Apache Commons Logging`. V rámci rozšíření aplikace by bylo možné ukládat zprávy o nejdůležitějších událostech v aplikaci také do databáze a zobrazovat je v prezentační vrstvě. To by umožnilo pohodlnější sledování provozu aplikace. Uživatelsky příjemnější formou, v porovnání se sledováním logu, by bylo možné sledovat historii vyhodnocení pravidel a ověřovat vyvolání správných akcí. Tímto způsobem by uživatel snadno identifikoval případné chyby v pravidlech nebo samotných akcích.

### Zneplatnění záznamů v databázi

S předchozím bodem souvisí další možnost vylepšení. Pokud uživatel v aplikaci provede odstranění pravidel, akcí nebo konfigurace pravidel, tak tato operace způsobí skutečné odstranění záznamů z databáze. Jiným přístupem by bylo pro inkriminované tabulky zavést nový sloupec, jehož hodnota by indikovala platnost záznamu. Při mazání v aplikaci by byl ve skutečnosti záznam ponechán v databázi, ale byl by zneplatněn nastavením hodnoty nového sloupce, tzv. flagu. To by umožnilo auditovat historickou činnost aplikace a dohledat informace, které by v případě odstranění z databáze byly ztraceny.

### Dokonalejší integrace s UCD

Dalším možným bodem rozšíření je zdokonalení integrace se systémem UrbanCode Deploy. Při zakládání akce volání REST API by bylo možné ve formuláři dynamicky nabízet uživateli dostupné Aplikace, Komponenty a Prostředí z konkrétního UCD serveru a operace nad nimi. UCD nabízí API pro zjištění takových informací. To by bylo jistě uživatelsky pohodlnější než ruční zadávání ve formátu JSON.

# 8 Závěr

Cílem této práce bylo navrhnout a implementovat systém, který by v reakci na uložení pracovní položky v RTC, umožnil spouštět konfigurovatelné akce.

V prvním kroku bylo nutné rozhodnout, jak systém koncipovat. V úvahu přicházelo celou funkcionalitu umístit do pluginu, který by rozšiřoval funkčnost na straně RTC. Plugin by byl spouštěn po uložení pracovní položky.

Nakonec byla zvolena implementace dvou oddělených komponent. První komponentou je RTC plugin, který rozšiřuje funkčnost na straně RTC o volání webové aplikace, která vyhodnocuje pravidla a spouští akce.

Pro realizaci systému bylo nutné se seznámit s platformou RTC, s prostředky RTC SDK, nainstalovat platformu Jazz a RTC Eclipse Client. Dalším krokem bylo vymyslet způsob jakým reprezentovat pravidla, po jejichž vyhodnocení mohou být spuštěny akce.

V teoretické části práce je čtenář seznámen s problematikou nasazování software a s principy přístupu DevOps. Poté jsou představeny nástroje společnosti IBM pro podporu DevOps, zejména pak nástroje Rational Team Concert a UrbanCode Deploy, které úzce souvisí s praktickou částí práce.

Implementace byla realizována s ohledem na budoucí rozšíření, zejména o možnost přidání další akce. Obě komponenty byly náležitě otestovány jak funkčními testy, tak zátěžovými testy a zčásti jednotkovými testy.

Práce také obsahuje návrh budoucího rozšíření s ohledem na lepší použitelnost vyvinutého software. Vytvořený systém je připravený k testování na produkčním prostředí. Cíl práce byl splněn, podařilo se vytvořit funkční systém, který splnil očekávání a bude dále vyvíjen.

# Přehled použitých zkratek

| | |
|---|---|
| **API** | Application Programming Interface |
| **CLI** | Command Line Interface |
| **CCM** | Change and Configuration Management |
| **CLM** | Rational Collaborative Lifecycle Management |
| **CSV** | Comma Separated Values |
| **DAO** | Data Access Object |
| **EL** | Expression Language |
| **HTTP** | Hypertext Transfer Protocol |
| **IBM** | International Business Machines |
| **IDE** | Integrated Development Enviroment |
| **ITIL** | Information Technology Infrastructure Library |
| **JDK** | Java Development Kit |
| **JSON** | JavaScript Object Notation |
| **JSP** | JavaServer Pages |
| **JSTL** | JavaServer Pages Standard Tag Library |
| **JTS** | Jazz Team Server |
| **OSLC** | Open Services for Lifecycle Collaboration |
| **QM** | Quality Management |
| **RANE** | RTC Action & Notification Engine |
| **RDNG** | Rational Doors Next Generation |
| **REST** | Representational State Transfer |
| **RM** | Requirements Management |
| **RQM** | Rational Quality Manager |
| **RTC** | Rational Team Concert |
| **SCM** | Supply Chain Management |
| **SDK** | Software Development Kit |
| **SLA** | Service-level agreement |
| **SMTP** | Simple Mail Transfer Protocol |
| **SW** | Software |
| **UAT** | User Acceptance Test |
| **UCD** | UrbanCode Deploy |
| **UML** | Unified Modeling Language |
| **URI** | Uniform Resource Identifier |
| **URL** | Uniform Resource Locator |
| **WCL** | WorkItem Command Line |
| **XML** | eXtensible Markup Language |

# Literatura


[1] *Collaborative lifecycle management linking* [online]. IBM. [cit. 05/2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/SSYMRC_6.0.4/com.ibm.help.common.jazz.calm.doc/topics/c_calm_common.html.

[2] *IBM Knowledge Center* [online]. IBM. [cit. 03/2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SS4GSP_6.2.7/com.ibm.udeploy.tutorial.doc/topics/webapp_application_process.html.

[3] *DevOps* [online]. bestpractice.cz. [cit. 05/2018]. Dostupné z: https://www.bestpractice.cz/cs/Best-practice/DevOps.alej.

[4] Humble, J. – Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley Signature Series (Fowler). Pearson Education, 2010. ISBN 9780321670229.

[5] *IBM Knowledge Center* [online]. IBM, 2015. [cit. 04/2018]. Rational solution for CLM documentation. Dostupné z: https://www.ibm.com/support/knowledgecenter/cs/SSJJ9R_6.0.1/com.ibm.rational.clm.doc/helpindex_clm.html.

[6] *Přehled produktu Rational Quality Manager* [online]. IBM, . [cit. 06/2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/cs/SSYMRC_6.0.4/com.ibm.rational.test.qm.doc/topics/c_qm_overview.html.

[7] *Přehled produktu Rational Quality Manager* [online]. IBM, . [cit. 06/2018]. Dostupné z: https://www.ibm.com/support/knowledgecenter/cs/SSYMRC_6.0.4/com.ibm.rational.test.qm.doc/topics/c_qm_overview.html.

[8] Schoon, R. *Rational Team Concert process fundamentals* [online]. IBM, 2016. [cit. 04/2018]. Dostupné z: https://jazz.net/wiki/bin/view/Deployment/RTCProcessFundamentals.

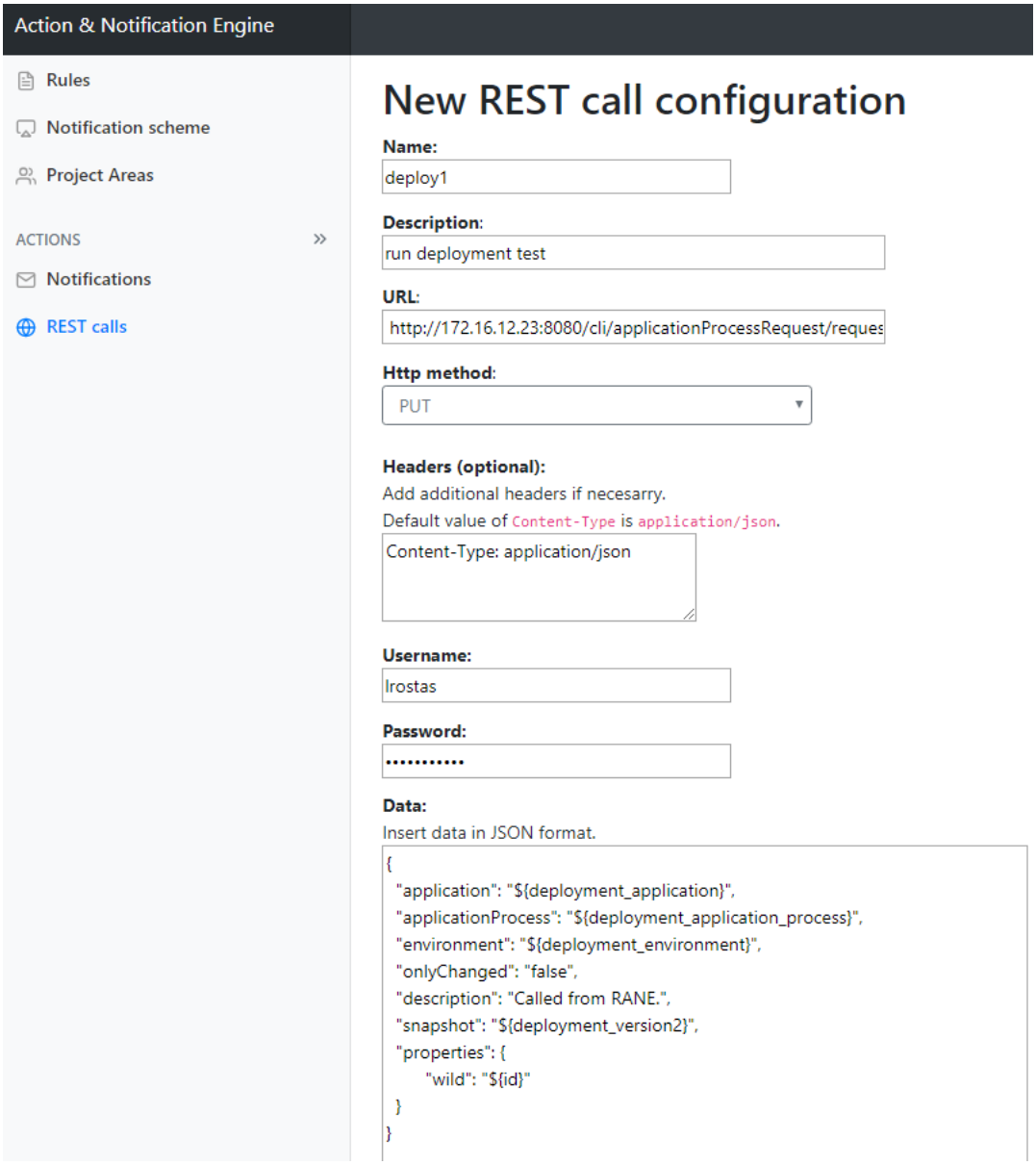
[9] Schoon, R. *A RTC WorkItem Command Line Version 3.0* [online]. IBM, 2015. [cit. 05/2018]. Dostupné z: https://rsjazz.wordpress.com/2015/11/03/a-rtc-workitem-command-line-version-3-0/.

[10] Sharma, S. *DevOps for Dummies.* Wiley, 2014. ISBN 139781118734704.

[11] Sharma, S. *The DevOps Adoption Playbook: A Guide to Adopting DevOps in a Multi-Speed IT Enterprise.* Wiley, 2017. ISBN 9781119308744.

# Přílohy

# A Formulář vytvoření šablony volání REST




Obrázek A.1: Formulář šablony volání REST API