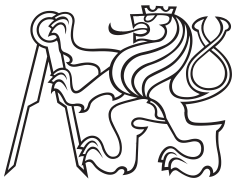
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

Vývoj integračního adaptéru pro IBM RTC a Blueworks Live

Ruslan Bakeyev

Školitel: Ing. Martin Samek
Obor: Kybernetika a Robotika
Zaměření: Robotika
Květen 2016

Poděkování

Rád bych poděkoval Ing. Martinu Samkovi a RNDr. Danielu Prusovi za jejich čas a cenné rady po dobu psání této práce.

Také bych rád poděkoval rodině za obrovskou podporu během celého studia na Fakultě elektrotechnické.

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, 27. května 2016

Abstrakt

Tato bakalářská práce popisuje jeden z moderních způsobů integrace dat mezi cloudovými a tradičními infrastrukturami. Jedná se o standardy vyvinuté OSLC (Open Services for Lifecycle Collaboration) komunitou. Pro zajištění integrace byla v rámci práce vyvinuta aplikace v jazyce Java, která je založena na projektu Eclipse Lyo. V závěru je znázorněna dosažená výkonnost a finální struktura adaptéru.

Klíčová slova: OSLC adaptér, SaaS, Eclipse Lyo, Blueworks Live, RTC, REST

Školitel: Ing. Martin Samek

Abstract

This thesis describes the modern way of data integration between cloud and traditional infrastructures. These are the standards defined by OSLC (Open Services for Lifecycle Collaboration) comunity. In order to ensure integration between these two platforms, Java application was developed. The written program is based on Eclipse Lyo framework. The conclusion illustrates both the final structure and the achieved performance of adapter.

Keywords: OSLC adapter, SaaS, Eclipse Lyo, Blueworks Live, RTC, REST

Title translation: Integration adapter development for IBM RTC and Blueworks Live

Obsah

Kapitola 1
Úvod

Tato bakalářská práce je vytvořená podle požadavků z průmyslu a zabývá se otázkou integrace cloudové služby IBM Blueworks Live a tradičního IT řešení IBM Rational Team Concert. Zvolená integrační metoda je založena na standardech vyvinutých OSLC (Open Services for Lifecycle Collaboration) komunitou.

Kapitola 2 se převážně zabývá problematikou cloudového světu. Patří tam i popis základních distribučních modelů a na příkladu aplikace IBM Blueworks Live je proveden detailnější rozbor SaaS (Software as a service) platformy. Jelikož ne každý podnik si může dovolit přemístění svých software řešení do cloudu, otázka integrace se stává aktuálnější. Jedním s možných řešení je přijetí OSLC standardů popsaných v sekci 2.3.

Výsledkem této práce je integrační nástroj psaný v jazyce Java. Avšak nejde o implementaci úplně od začátku. Předané zdrojové kódy měli zjevné vady. Z toho důvodu byly zákazníkem stanoveny požadavky na optimalizaci a základní funkčnost adaptéru. Kapitola 3 poskytuje detailnější přehled o hlavních cílech této práce.

Vyvíjený nástroj používá několik různých technologií a Java knihoven. Například projekt Eclipse Lyo usnadňuje přijetí OSLC standardů. Stručný přehled použitých technologií a postup při optimalizaci rychlosti adaptéru uvádí kapitola 4.

Poslední kapitola ukazuje konečnou strukturu vytvořené aplikace spolu s dosaženými výsledky během řešení optimalizačních úloh.

Kapitola 2

Integrace SaaS a on-premise systémů pomocí OSLC přístupu

V současně době jsou cloudové technologie jedním z nejperspektivnějších trendů. O Apple iCloud, Microsoft Azure, IBM Bluemix, Amazon EC2 určitě slyšel i ten, kdo neví, co se skrývá pod těmi názvy. Každý rok se zvětšuje počet korporací rozhodujících ve prospěch cloudu. Avšak ne každý podnik může dovolit přemístění svých software řešení do této platformy. Jedním z nejčastějších důvodů jsou migrační náklady, i když se s velkou pravděpodobnosti v budoucnu vyplatí. Otázka integrace s tradičním (tj. on-premise) softwarem se s rozšířením počtu cloudových služeb stává aktuálnější. Tato práce se zabývá řešením právě uvedeného problému. Pomocí standardů popsaných OSLC[1] komunitou si ukážeme, jak se dá propojit cloudovou službu IBM Blueworks Live a on-premise řešení IBM Rational Team Concert (RTC). Než se pustíme do implementace, stručně popíšeme základní pojmy, které budeme v této práci dále potřebovat.

2.1 Co je to cloud computing?

Cloud computing (dále v této práci jen cloud) je způsob poskytování počítačových prostředků v nejrůznějších formách[1]. Může to být sdílení aplikace, výpočetního výkonu serveru, diskového prostoru, internetového přípojení atd. Mezi nejznámější cloudy patří iCloud, Google Drive a Microsoft Azure. Všechny tyto služby plní roli úložiště a nabízejí vysokou efektivitu a spolehlivý model zabezpečení dat za malou cenu.

Základem cloudových služeb je princip virtualizace hardwaru. Zjednodušeně řečeno, tato technologie umožňuje rozdělit komponenty počítače (jako např. procesor, paměť atd.) do samostatných skupin, čímž se vytvoří nové prostředí s jinými výkonnostními vlastnostmi. Avšak lze virtualizovat nejenom hardware, ale i software. Virtualizace operačních systému je jedním z příkladů. Tím se umožňuje paralelní běh více na sobě nezávislých operačních systémů na jednom serveru. V současné době nejznámějším poskytovatelem této služby na trhu je společnost VMware.

[1] Open Services for Lifecycle Collaboration

Jedna z největších výhod použiti virtualizace je možnost jednoduše a rychle přizpůsobit prostředí pro požadavky podniku. Vysvětlíme to na prostém příkladu. Představme si, že kvůli naplánovanému výprodeji predikovalo analytické oddělení desetinásobné zvětšení počtu klientů internetového obchodu. Avšak současné výkonnostní možnosti serveru neumožňují zpracování takového množství objednávek. Pokud podnik využívá tradiční (tj. on-premise) IT řešení, pak musí dokoupit další servery, které pravděpodobně po skončení akce nebudou využívat ani 50% svých možnosti. Oproti tomu virtualizace nabízí možnost dočasného zvýšení výkonu (tj. přizpůsobení prostředí pro požadavky podniku). Zbývající část výpočetních prostředků se po skončení akce vrátí pro původní potřeby. Jelikož cloud je většinou postaven na principu virtualizace, možnost škálování fyzické infrastruktury platí i pro něj.

Dalšími značnými výhodami použiti cloudu jsou:

- Ekonomická výhoda. Odpadá nutnost nákladu na obsluhu serverové infrastruktury, je totiž na poskytovateli cloudu.
- Nezávislost na platformě klienta. Poskytované služby by měli být přístupné pomocí web prohlížeče z libovolného zařízení.
- Vyšší zabezpečení dat.

Avšak existují jisté nevýhody:

- Nutnost internetového přípojení.
- Náklady na migraci.
- V porovnání s tradičními aplikacemi cloudové služby nabízejí menší funkcionalitu.

2.1.1 Rozdělení cloudových systému

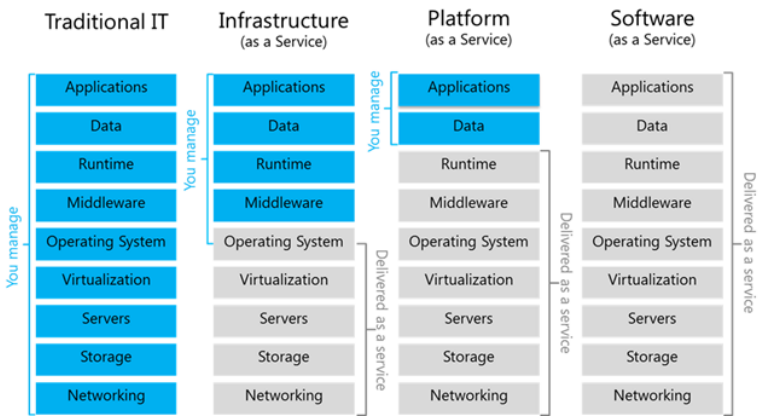
Každý podnik má svoje specifické požadavky na poskytované služby. Například někdo potřebuje pouze infrastrukturu, tj. výpočetní výkon, zatímco jiný podnik potřebuje již nastavené prostředí pro efektivní vývoj a testování aplikace. Z tohoto důvodu se vyvíjí různé typy cloudových prostředí. Tři základní jsou:[2]:

- Infrastruktura jako služba (IaaS) umožňuje klientovi detailní výběr komponent serveru (např. procesoru, typu a velikosti paměti atd.) pro vybudovaní svého vlastního cloudového řešení. Všechny problémy s hardware řeší poskytovatel služby. Volba operačního systému, aplikačního serveru a dalšího software je na zákazníkovi. Příklad poskytovatele IaaS je Microsoft Azure.
- Platforma jako služba (PaaS) poskytuje klientovi prostředí pro vývoj, testování a provoz vlastní aplikace bez nutnosti spravování jednotlivých komponent (jako např. databáze, systémové knihovny atd.). Příklad poskytovatele PaaS je IBM Bluemix.

- Software jako služba (SaaS) umožňuje klientovi používat aplikaci, která je nasazená v cloudovém prostředí. Velkou výhodou je jednoduché a přívětivé uživatelské rozhraní. Zákazník nemusí mít žádné specifické technické znalosti. Všechny problémy s poskytovanou aplikací, licencí a zabezpečením dat řeší provider. Nejčastěji se zákazník stará pouze měsíční poplatky nebo jinou formu platby za využíti služeb. Příkladem SaaS služby je aplikace pro vývoj procesních diagramů Blueworks Live.

Obrázek 2.1 názorně porovnává typy cloudových služeb s on-premise (tj. tradičním) přístupem.

Cloudové prostředí můžeme také roztřídit podle modelu nasazení (deployment model). Představme si, že podnik má vlastní bezpečnostní požadavky na poskytovanou službu, pak jedním z řešení je privátní model cloudu. Jedná se o infrastrukturu, která je upravená v souladu s potřebami zákazníka. Dokonce vlastníkem prostředí může být i sám klient. Opačným případem je veřejný model, který je určen pro širokou veřejnost a není adaptován pro jednotlivé potřeby zákazníků.



Obrázek 2.1: Porovnání základních cloudových služeb s tradiční infrastrukturou[3]

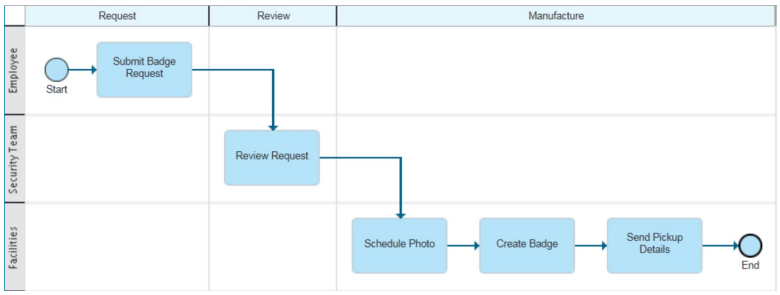
2.1.2 IBM Blueworks Live

IBM Blueworks Live je cloudový software (SaaS), který umožňuje vývoj a kreslení diagramů podnikových procesů[5]. Tento nástroj přebírá veškeré výhody způsobené distribučním modelem:

- Jednoduché a přívětivé uživatelské rozhraní.
- Absence nutnosti stahování a instalace aplikace.
- Snadný model předplacení.
- Prostředí pro společný vývoj procesních diagramů.

Jelikož se tato práce zabývá otázkou integrace služby Blueworks Live, je dobré mít představu o poskytované funkcionalitě.

Klíčovým pojmem je podnikový proces (dále již proces). Jedná se o překlad anglického termínu Business Process. Zjednodušeně řečeno, podnikový proces je množina kroků, které je potřeba vykonat pro splnění určitého úkolu. Výsledkem této činnosti může být poskytnutí služby nebo hotový produkt. Obrázek 2.2 uvádí příklad podnikového procesu. Je vidět, že proces tvoří jednotlivé kroky, kterým se říká aktivity. Sloupce diagramu neboli milestones reprezentují jednotlivé fáze procesu. Uvedený příklad se skládá ze třech etap: žádost (request), vyhodnocení (review) a výroba (manufacture). Řádky diagramu neboli swimlanes představují oddělení společnosti, které by měli tyto činnosti vykonávat.



Obrázek 2.2: Příklad podnikového procesu[4]

Tato práce se zabývá způsobem integrace dat mezi cloudovou službou Blueworks Live a on-premise řešením IBM Rational Team Concert (RTC), ovšem je nutné znát typ dat, která přesně chceme přenášet. Jako většina SaaS řešení, IBM Blueworks Live poskytuje rozhraní (API[2]) pro přístup k potřebným informacím. Především nás zajímají následující druhy dat:

- Název, jak procesu, tak i všech obsahovaných aktivit.
- Datum poslední úpravy a jméno autora procesu.
- Dokumentace k jednotlivým aktivitám.

Po obdržení požadované informace nasleduje proces transformace (viz. sekce 2.3) a ukládání výsledku do programu RTC (viz. sekce 2.2).

2.2 IBM Rational Team Concert

IBM Rational Team Concert (RTC) je nástroj umožňující efektivní řízení životního cyklu aplikací (application lifecycle management, viz. [6]). Zjednodušeně řečeno, jedná se o tradiční (tj. on-premise) řešení, poskytující sdílené prostředí pro plánovaní, reporting, automatizaci, vývoj a testování aplikací. Program se skládá ze 4 modulů neboli domén: Requirements management (RM), Change and configuration management (CCM), Quality management (QM) a Design management (DM). Každá doména je samostatnou aplikací,

[2]Application Programming Interface

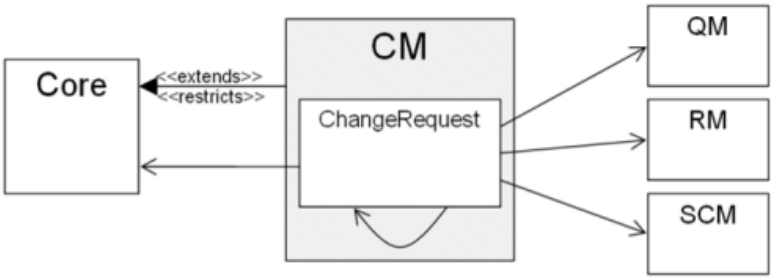
která plní svoji určitou roli v procesu vývoje softwaru. Avšak tyto moduly jsou navrženy tak, aby mezi sebou mohli jednoduše komunikovat.

Jedním z požadavků, který se řeší v této práci je rozšíření oblasti funkčnosti výsledného adaptéru na CCM doménu, tzn. možnost ukládaní informací o zvoleném procesu (příp. aktivitě) v CCM modulu nástroje.

2.3 OSLC přístup

V předchozích sekcích je popsán typ dat, která chceme přenášet mezi SaaS a on-premise platformami. Jsou to vlastnosti procesů a aktivit, jako např název nebo datum poslední editace. Avšak dosud nebylo nic řečeno o způsobů předávání dat.

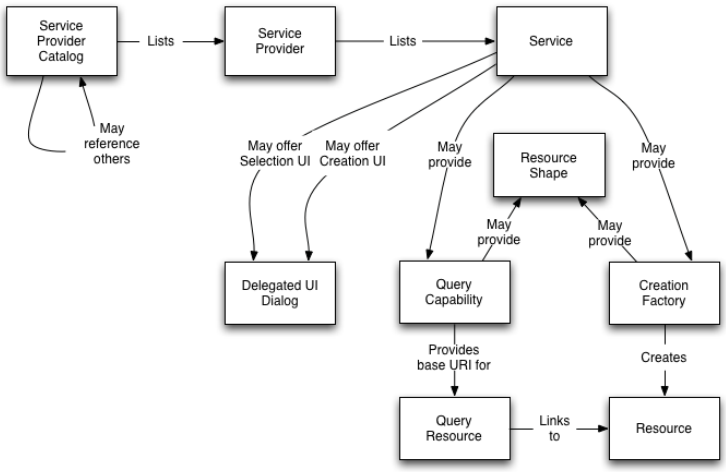
V roce 2008 byly OSLC (Open Services for Lifecycle Collaboration) komunitou vyvinuty integrační standardy pro produkty a služby, které podporují všechny fáze životního cyklu aplikace[8]. Každá část tohoto cyklu má svoji vlastní doménu a specifikaci, např. RM, CM nebo QM (viz. sekce 2.2 a [6]). Ovšem základ tvoří tzv. OSLC Core specifikace, která definuje zásadní funkcionalitu každé domény. Obrázek 2.3 znázorňuje vztahy mezi jednotlivými specifikacemi.



Obrázek 2.3: Vztah mezi Core a ostatními domény[8]

Na uvedeném diagramu je vidět, že CM doména rozšiřuje Core specifikaci a zároveň odkazuje na zdroje definované v ostatních doménách. Tato práce uvádí pouze stručné vysvětlení Core specifikaci, protože se jedná o striktně stanovený standard, který definuje zaprvé sadu povinných atributů datové struktury (OSLC properties) a zadruhé množinu přípustných operací nad zdrojem. Pro komfortní práci s OSLC specifikacemi stačí pochopit základní koncept, který je znázorněn na obrázku 2.4. Uvedený diagram zahrnuje 3 klíčové pojmy:

- Služba (OSLC Service) je množina operací, která umožňuje vytváření, mazání, aktualizaci a poskytování přístupu ke zdroji.
- Poskytovatel služeb (OSLC Service Provider) je nejčastěji web služba nebo produkt, který poskytuje implementaci jedné nebo více služeb.
- Katalog poskytovatele služeb (OSLC Service Provider Catalog) je soubor poskytovatele služeb.



Obrázek 2.4: Koncept OSLC Core domény[7]

Hlavním problémem při přijetí OSLC přístupu je adaptace nástroje pod definovaný model. To znamená, že je nutně rozhodnout o rozdělení aplikace do jednotlivých částí, které budou plnit rolí služeb, poskytovatele služeb a katalogu poskytovatele služeb. Jako příklad uvedeme nástroj Blueworks Live. V této práci role služeb plní poskytnuté rozhraní. Avšak není to jediný způsob příjetí OSLC modelu. Například je možné pro roli služby využít i podnikový proces, zatímco poskytovatelem služeb bude API nástroje Blueworks Live.

Kapitola 3

Základní funkcionalita adaptéru a analýza požadavků

Cílem této práce je vytvořit integrační adaptér pro produkty IBM RTC a Blueworks Live. Obě aplikace byly popsány v kapitole 2. Zvolenou metodou integrace je implementace OSLC standardů. Vyplývá to z toho důvodu, že tato specifikace je podporována IBM RTC. Více o OSLC standardech je popsáno v sekci 2.3.

Předpokládá se, že adaptér bude mít následující funkcionalitu:

1. Poskytování uživatelského rozhraní (vnořeného do nástroje RTC) pro hledání požadovaného procesu nebo aktivit.
2. Transformace nalezeného procesu (resp. aktivity) do objektu, dodržujícího OSLC normy.
3. Uložení odkazu na vytvořený objekt v programu RTC.
4. Generování náhledu (UI Preview), tj. vnořené HTML stránky, poskytující informaci o vybraném procesu (resp. aktivitě) v programu RTC.
5. Generování odkazu na vybraný proces (resp. aktivitu), který přesměruje uživatele do služby Blueworks Live.

Avšak nejde o implementaci úplně od začátku. Předaný zdrojový kód měl zjevné vady, jako např. nefunkčnost v doméně CCM[1] v aplikaci IBM RTC nebo špatná optimalizace kódu, volajícího API[2] serveru Blueworks Live. Na základě nalezených problémů byl sestaven seznam požadavků:

- Optimalizace rychlosti adaptéru, aby byl použitelný koncovým uživatelem.
- Rozšíření oblasti funkčnosti na CCM doménu.
- Možnost nastavení URL[3] serveru Bluework Live pro případ, pokud zákazník bude mít vlastní upravenou kopii této služby ve svém prostředí.

[1] Change and configuration management
[2] Application Programming Interface
[3] Uniform Resource Locator

- Zlepšení vzhledu vnořené stránky (UI preview).

První dva požadavky mají největší prioritu, protože zajišťují základní funkcionalitu adaptéru. Až ve chvíli, kdy je máme splněno, můžeme přejít k vypracování dalších úkolu.

Řekneme, že aplikace je použitelná, pokud odezva adaptéru nepřekračuje dobu 8 sekund. Jinak řečeno, doba obdržení uživatelem výsledku nesmí trvat déle, než 8 sekund. Tato hodnota je stanovená zákazníkem.

Při optimalizaci rychlosti adaptéru se zjistilo, že původní verze aplikace se zakládá na projektu OSLC4J Bugzilla Adapter[4] s mírnými úpravami. Důsledkem toho je nepřehlednost kódu a přítomnost zbytečných Java tříd, atributů a metod, což činí problémy při rozšíření funkcionality adaptéru a pochopení logiky aplikace. Takže dalším požadavkem bylo:

- Refaktorování aplikace, tj. úprava zdrojového kódu beze změny funkcionality adaptéru.

Splnění tohoto požadavku vede k zvýšení rychlosti vývoje programu a usnadňuje ladící proces.

Optimalizační proces vyžaduje dobré porozumění vnitřní logice adaptéru včetně použitých technologií. Vzhledem k tomu, že produkt IBM Blueworks-Live poskytuje REST API[5], pak je dobře znát strukturu a možnosti protokolu HTTP. Sám o sobě adaptér je také RESTful službou základ kterého tvoří implementace JAX-RS API, který je součástí Java Enterprise Edition. Dodržení požadavků OSLC specifikace a serializace datových struktur do RDF/XML formátu zajišťuje projekt Eclipse LYO. Technologie Java Server Pages (JSP) je zodpovědná za generovaní HTML stránek pro uživatele. Stručný rozbor použitých nástrojů je uveden v kapitole 4.

[4] `http://open-services.net/software/oslc4j-bugzilla-adapter/`

[5] Application Programming Interface

Kapitola 4
Implementace

4.1 Přehled použitých technologií

V kapitole 3 jsme zmínili o pomocných nástrojích, které jsou využité pro implementaci adaptéru. V následujících podsekcích stručně popíšeme každou z nich.

4.1.1 Protokol HTTP

Protokol HTTP (HyperText Transfer Protocol) je jeden z nejrozšířenějších způsobu předání dat, který původně byl navržen pro výměnu HTML souborů. Patří do zásobníku TCP/IP a je protokolem aplikační vrstvy podle OSI specifikace[1]. Základem HTTP je architektura klient-server, tzn. že se serveru od klienta posílá dotaz (request), který obsahuje označení potřebné informace, zatímco server vrátí odpověď (response) s požadovanými daty. Příklad nejjednoduššího dotazu na server cvut.cz vypadá následovně:

```
GET / HTTP/1.1
Host: www.cvut.cz
```

Ukázka kódu 4.1: Příklad HTTP dotazu

Struktura dotazu je poměrně jednoduchá a má tvar:

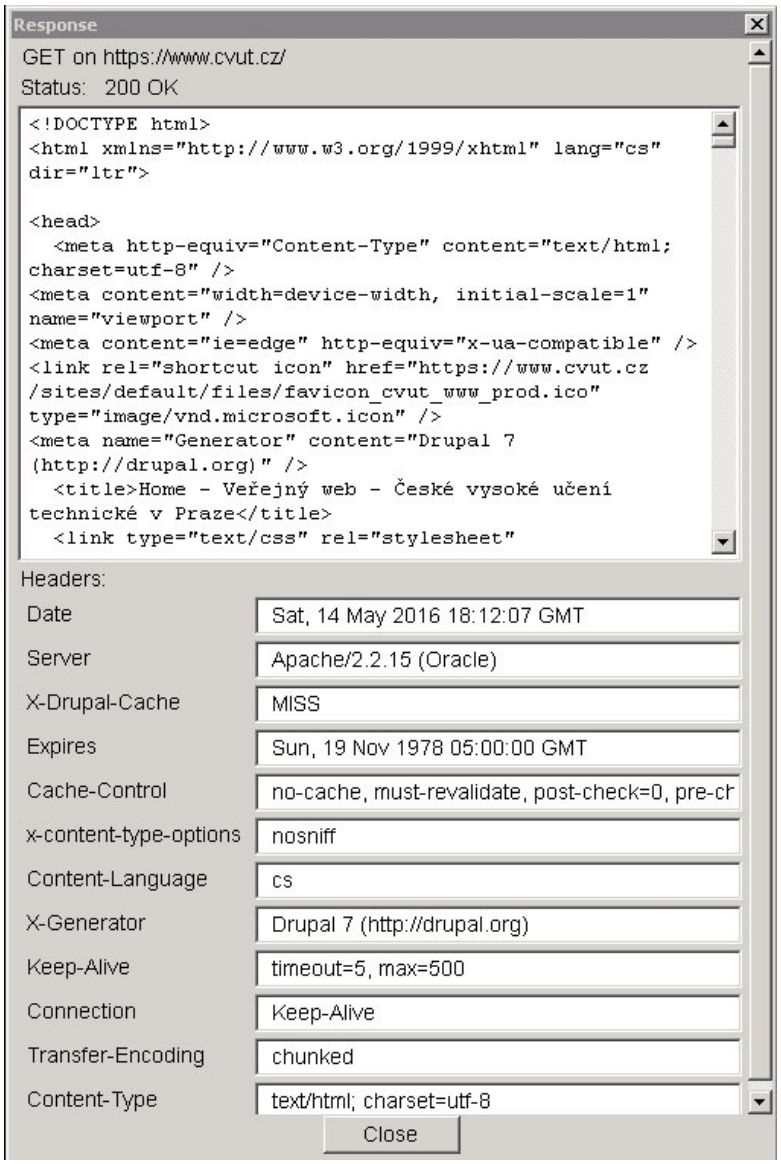
<Metoda> <URI> <Verze>

Důležitá je dotazovací metoda, což je posloupnost libovolných symbolů s výjimkou řídicích (např. "/0", "/t") a rozdělovacích (např. ","), která definuje činnost, která má být provedena ohledně zdroje informací, tj. URI. [10]

Po zaslání dotazu obdrží klient odpověď, která je znázorněna na obrázku 4.1. Jedním z klíčových údajů je návratová hodnota (Status Code) a její slovesná interpretace. V tomto případě server vrátil kód "200 OK". Další významné hodnoty jsou: "401 Unauthorized", "404 Not Found"a "502 Bad Gateway". Pak nasledují požadovaný HTML soubor a seznam hlaviček (Headers) obsahující

[1] `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=20269`

informaci o serveru, délce a typu vraceného dokumentu, času vytvoření odpovědi atd. V adaptéru se tyto hlavičky používají pro zjištění formátu dotazovaných dat (např. JSON nebo XML).



Obrázek 4.1: Příklad HTTP odpovědi

4.1.2 RESTful rozhraní

Technologie REST (Representational state transfer) definuje architekturu rozhraní pro návrh web služeb. Jedná se o styl, který nejčastěji využívá vlastností a metod protokolu HTTP. Velkou výhodou tohoto přístupu je nezávislost na programovacím jazyku aplikace klienta. Každá RESTful služba musí implementovat REST architekturu, tedy striktně dodržovat čtyři základní principy:

1. Definování a použití HTTP dotazovacích metod v souladu se standardem RFC 2616 [2]. Specifikace RFC 2616 jednoznačně definuje význam a způsoby použití následujících metod:
   - Pro obdržení zdroje informace ze serveru se používá GET metoda.
   - Pro vytváření zdroje informace na serveru se používá POST metoda.
   - Pro změnu stavu zdroje informace na serveru se používá PUT metoda.
   - Pro mazaní zdroje informace ze serveru se používá DELETE metoda.
2. Absence stavu (Be stateless) pro zvětšení výkonností web servisu.
3. Reprezentace zdroje informace v různých formách, např. XML, JSON, HTML atd. Klient by měl pracovat výhradně s těmito formami.
4. Přiřazení každému zdroji informaci unikátního identifikátoru, který by měl odpovídat struktuře katalogů.

Poslední bod znamená, že by URI adresa měla být intuitivní, pochopitelná a předvídatelná. Například následující URL splňuje tuto podmínku:[11]

`http://www.myservice.org/discussion/topics/{topic}`

Jak již bylo zmíněno v sekci 4.1, IBM BlueworksLive poskytuje RESTové služby pro obdržení informace o procesech nebo aktivitách. Detailní přehled API viz použitá literatura [12]. Nicméně, ukážeme si příklad volání REST API serveru s využitím dotazovací metody GET v jazyce Java. Jde o univerzální šablonu, kterou se dá využít pro libovolný server, který nabízí stejné (tj. RESTful) služby a vrací JSON datovou strukturu. Stačí pouze implementovat mechanismus validace uživatelů. Tentýž kód se používá i v našem adaptéru:

```
private JSONObject makeApiCall(String reqURL)
                throws UnauthorizedException, RestException {
JSONObject apiCallResult = null;
try {

        // Code for user validation ..

        String finalURL = new String(reqURL + "&accountId=" + accountId);
        // Call the provided api on server with GET method
        HttpURLConnection restApiURLConnection = getRestApiConnection(
            finalURL, username, password);
        // Controlling the status code
        if (restApiURLConnection.getResponseCode() != HttpURLConnection.
            HTTP_OK) {
                throw new RestException(restApiURLConnection.
                    getResponseCode());
        }
        InputStream restApiStream = restApiURLConnection.getInputStream();
```

[2] `https://www.ietf.org/rfc/rfc2616.txt`

```
                try {
                        // Getting the output from the stream object
                        apiCallResult = new JSONObject(restApiStream);
                } finally {
                        restApiStream.close();
                }
        }
                // Exception handling code..

                return apiCallResult;
        }
```

Ukázka kódu 4.2: Příklad (šablona) volání REST API IBM Blueworks Live

Vracený JSON objekt vypadá následovně:

```
// Output from https://ibm.blueworkslive.com/scr/api/FollowedProcesses
{
        "followedProcesses ":[
          {
            "processId ":"4aeae03b0d",
            "name":"Demo BPM diagram",
            "modifiedUserName":"Ruslan Bakeyev",
            "modifiedDate":"5\/4\/16"
          },
          {
            "processId ":"6eadc5cac",
            "name":"Hiring - Onboarding",
            "modifiedUserName":"Ruslan Bakeyev",
            "modifiedDate":"11\/8\/15"
          },
                // Zkraceno
        ]
}
```

Ukázka kódu 4.3: Odezva serveru BlueworksLive na HTTP GET dotaz

4.1.3 JAX-RS servlet

V sekci 4.1.2 byly rozebrány základy REST architektury spolu s hlavními principy pro návrh RESTful služeb. Demonstrovali jsme i příklad volání REST API serveru BlueworksLive pomocí jazyku Java. Nicméně, zatím se nic neřeklo o tom, jak se dá implementovat RESTful služby ze strany serveru. Pro řešení tohoto problému v této práci se používá technologie JAX-RS. Jedná se o Java EE[3] knihovnu, umožnující efektivní a snadný vývoj RESTful služeb[13]. Specifikace JSR-311[4] popisuje význam a způsob použíti JAX-RS anotací. My si ale ukážeme příklady kódu, které jsou použité v této práci.

Následující ukázka znázorňuje obsah konfiguračního souboru (Web-descriptor), který se používá pří nasazení JAX-RS servletu na web kontejner (např. Apache

[3] Java Enterprise Edition

[4] `https://jcp.org/en/jsr/detail?id=311`

Tomcat [5]). Podle příkladu vytvořený web servis má jméno "JAX-RS servlet". Inicializace aplikace je implementována ve třídě "BlueworksLiveApplication".

```
<servlet>
        <servlet-name>JAX-RS Servlet</servlet-name>
        <servlet-class>org.apache.wink.server.internal. servlet .RestServlet</
            servlet-class>
        <init-param>
            <param-name>javax.ws.rs.Application</param-name>
            <param-value>uk.ibm.com.oslc.services.
                BlueworksLiveApplication</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
        <servlet-name>JAX-RS Servlet</servlet-name>
        <url-pattern>/services/*</url-pattern>
</servlet-mapping>
```

Ukázka kódu 4.4: Web-deskriptor (zkracený)

Typická třída použité technologie vypadá následovně:

```
// The example of the class, which uses JAX-RS framework
@Path("catalog")
public class ServiceProviderCatalogService {

  @Context private HttpServletRequest httpServletRequest;
  @Context private HttpServletResponse httpServletResponse;
  @Context private UriInfo uriInfo;

  @GET
  @Path("{serviceProviderId}")
  @Produces(MediaType.TEXT_HTML)
  public void getHtmlServiceProvider(@PathParam("serviceProviderId") final String
      serviceProviderId) {

    // Setting request attrubutes

    // Return the HTML representation with Request Dispatcher
    }
}
```

Ukázka kódu 4.5: Příklad třídy JAX-RS servletu

Uvedený příklad demonstruje použití klíčových anotací:

- Anotace @Path se používá pro definici cesty k zdrojům. Pro náš případ to znamená, že všechny dotazy na relativní URL `/catalog` budou zpracované instancí třídy ServiceProviderCatalogService. Ovšem, tato anotace může být použitá i u metod. Pokud pošleme dotaz např. na `/catalog/{serviceProviderId}`, kde místo parametru serviceProvi-

[5] `http://tomcat.apache.org/`

derId dosadíme ID poskytovatele služeb, tak to bude znamenat, že náš požadavek bude zpracován metodou getHtmlServiceProvider().

- Anotace @GET odpovídá dotazovací metodě, která se používá v HTTP. To samé platí i pro anotace @POST, @PUT, @DELETE a další. Význam těchto metod byl popsán v sekci 4.1.2.
- Anotace @Produces (resp. @Consumes) udává typy dat (např. JSON nebo XML), které tato metoda je schopna přijmout pro zpracování (resp. použit při generaci odpovědi.).

4.1.4 Java Server Pages

Java Server Pages (JSP) je technologie, která zjednodušuje generování web stránek, obsahujících jak statické, tak i dynamické komponenty. Statická data jsou nejčastěji představena ve formátu HTML. Přidáním JSP elementů je pak možně generovat dynamický výstup. Java elementy se vkládají pomocí speciální konstrukce:

<% ... %>

V této práci technologie JSP je použitá např. pro zobrazení uživatelovi informace o procesech nebo aktivitách.

4.1.5 Eclipse LYO

Jedna z nejdůležitějších použitých technologií je Eclipse LYO. Jedná se o sadu Java knihoven, které usnadňují transformaci a serializaci datových struktur (pro nás případ jsou to instance třídy OSLCRequirement) do formátů XML, JSON a RDF/XML. Základní komponenty nástroje jsou: [14]

- OSLC4J - jádro aplikaci, poskytující množinu Java anotací.
- Apache Jena Provider - knihovna, umožňující serializaci OSLC anotovaných Java objektu z/do RDF/XML formátu.
- Apache Wink Json4J Provider - knihovna, umožnující jak serializaci, tak i zpracování JSON dat.

Následující příklad demonstruje definici třídy OSLCRequirement, která následuje vlastnosti datové struktury Requirement:

```
// Definition of OSLCRequirement class which belongs to RM domain
@OslcNamespace(Constants.
    REQUIREMENTS_MANAGEMENT_NAMESPACE)
@OslcName(Constants.REQUIREMENT)
@OslcResourceShape(title = "Requirements Management Resource Shape",
    describes = Constants.TYPE_REQUIREMENT)
public final class OSLCRequirement extends Requirement {
        // Set of both OSLC Core/RM and user defined attrubutes ..
}
```

Ukázka kódu 4.6: Definice třídy OSLCRequirement

Definice atributů třídy (např. již zmíněné OSLCRequirement) je jednoduchou úlohou, což znázorňuje následující ukázka:

```
@OslcDescription("Timestamp last latest resource modification.")
@OslcPropertyDefinition
        (OslcConstants.DCTERMS_NAMESPACE + "modified")
@OslcReadOnly
@OslcTitle("Modified")
public Date getModified() {
        return modified;
}
```

Ukázka kódu 4.7: Definice atributu "Modified"

4.2 Optimalizace rychlosti adaptéru

Jedna ze základních úloh, která se řeší v této práci, je optimalizace rychlosti adaptéru. Předaná verze programu zjevně nesplňovala požadavek použitelnosti z hlediska koncového uživatele (viz sekce 5.3). Je zřejmě, že v současné době zpracování dat o objemu stovky kilobajtů, což zhruba odpovídá velikosti vracené odpovědi serveru Blueworks Live ve formátu JSON, jednoduše zvládají i osobní počítače. Z toho důvodu, hlavní optimalizační úlohou je minimalizace počtu dotazů na server.

Pomocí metody getCurrentTime() ze třídy System bylo zjištěno, že střední doba mezi procesy odeslaní dotazu na server Blueworks Live a obdržením potřebných dat je zhruba 2000 ms. Bohužel, tuto hodnotu nejsme schopni žádným způsobem ovlivnit, protože se jedná o vnitřní implementaci volané služby. Nicméně, po zkoumání REST API serveru Blueworks Live se dalo najít efektivnější cesty pro obdržení informace jak o procesech, tak i aktivitách, které tyto procesy tvoří. Dál si ukážeme konkrétnější příklady.

Z hlediska funkčnosti adaptér nabízí dvě základní služby, jejichž rychlost vykonávání je potřeba optimalizovat:

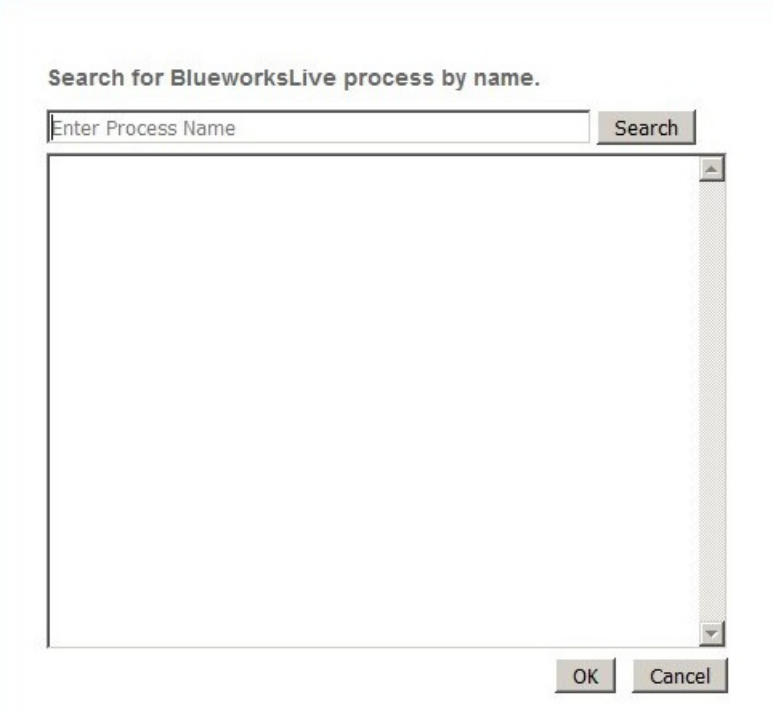
- Hledání procesu a aktivit, ke kterým uživatel má právo přístupu.
- Generování náhledové stránky (UI preview) zvoleného procesu nebo aktivity.

4.2.1 Optimalizace rychlosti hledání procesů a aktivit

V této a následující podsekcích budeme na konkretních příkladech ukazovat průběh optimalizace rychlosti hledání procesů a aktivit.

Celý proces začíná voláním metody changeRequestSelector() (resp. requirementsSelector() po refaktorování) ze třídy BugzillaChangeRequestService (resp. BlueworksRequirementsService), výsledkem které je přesměrování uživatele na HTML stránku, která je znázorněna na obrázku 4.2.

Po zadání názvu požádaného procesu (resp. vzoru, obsahujícího posloupnost symbolů, zakončené symbolem "*") tato informace se předává do metody



Obrázek 4.2: Uživatelské rozhraní pro hledání procesů

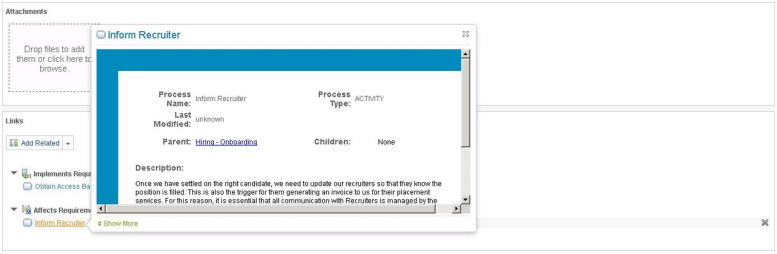
sendFilteredBugsResponse() (resp. sendFilteredProcessesResponse()), kde již probíhá volání API serveru Blueworks Live. Jak v původní, tak i ve finální verze adaptéru se pro účely hledání procesu používá dotaz "Search", který např. pro proces s názvem "Hiring"vypadá následovně:

```
https://www.blueworkslive.com/scr/api/Search?version=20111217&
searchValue=Hiring&searchFieldName=process_name&returnFields=*
```

Server udává odpověď ve formě JSON objektu. Pokud hledaný proces existuje, výstupný soubor bude obsahovat název procesu, jeho ID a také jak název, tak i ID aktivit. Tyto data se pak transformují pomocí metody fromRequirement() (resp. fromBlueworksProcess()) ze třídy OSLCRequirement v souladu s požadavky Requirements Management domény (viz [9]). Avšak problém nastává přesně v tomto kroku. V původní verzi adaptéru se pro každou aktivitu, kterou je potřeba převést na objekt typu OSLCRequirement, serveru posílá dvojice dotazů "Followed Processes"a "Process Data"pro vyhledaní "rodiče", tj. nadřazeného procesu. Tuto informaci potřebujeme zobrazovat v náhledu (UI Preview) (viz podsekce 4.2.2). Tento postup ale není optimální. Pokud si představíme proces, který obsahuje 20 aktivit, ztratíme průměrně 82 sekundy tím, že celkem pošleme 41 dotaz na server. Jednoduchým řešením je předaní objektu OSLCRequirement, reprezentujícího "rodiče", mezi parametry metody fromRequirement() (resp. fromBlueworksProcess()). Výsledkem celého postupu je seznam objektu OSLCRequirement, který tvoří právě tyto transformované procesy a obsahované aktivity.

4.2.2 Optimalizace rychlosti generování náhledové stránky

Náhledová stránka (UI preview) je vnořený HTML soubor, který se zobrazuje uživateli v programu RTC při navedení myši na uložený proces nebo aktivitu. Jedná se o zobrazení nejdůležitějších parametrů, popisujících zvolený objekt bez nutnosti přechodu na server Blueworks Live. Implementace podpory generování náhledové stránky není nutná podle OSLC specifikace, ovšem poskytovatel (provider) OSLC služeb musí se držet standardu, pokud takovou možnost nabízí [7]. Obrázek 4.3 znázorňuje příklad použití náhledových stránek v této práci.



Obrázek 4.3: Příklad náhledové stránky

Proces generování náhledové stránky se začíná s odesláním dotazu na URL:

```
http://HOSTNAME//bwloslc/services/bwl/requirements/{Id}
```

Do HTTP dotazu musí být nutně předána hlavička "Accept: application/x-oslc-compact+xml". Tím se zavolá metoda getCompact() ze třídy Bugzilla-ChangeRequestService (resp. BlueworksRequirementsService), která vrací instanci třídy Compact, obsahující nastavení náhledové stránky, např. délku a šířku. Obsah vnořeného okna (UI preview) se nastavuje pomocí metody smallPreview() ze stejné třídy. Volání API serveru Blueworks Live probíhá v tomto místě.

Vzhledem k tomu, že URL neposkytuje žádnou jinou informaci kromě Id artefaktu, nevíme předem, jestli se jedná o proces nebo aktivitu. Navíc, API serveru Blueworks Live nepodporuje hledání dat podle unikátního identifikátoru. Původní verze adaptéru pro tyto účely používá jíž zmíněnou dvojicí příkazu "Followed Processes"a "Process Data"(viz podsekci 4.2.1) pro zobrazení ID všech procesu a aktivit, ke kterým uživatel má právo přístupu. Tento postup ale není příliš efektivní, protože je závislý na počtu procesů uživatele, a již pro prostředí s třemi procesy v nejhorším případě bude trvat průměrně 14 sekund, pokud se jedná o aktivitě, nebo 12 sekund v případě procesu. Nesmíme také zapomenout, že náhledová stránka pro aktivitu obsahuje také i dokumentaci, kterou se dá získat pomocí příkazu "Activity Documentation".

Po zkoumání API serveru Blueworks Live se podařilo najít efektivní způsob zobrazení všech procesů a aktivit, ke kterým uživatel má právo přístupu. Jedná se o jednoduchý trik za pomocí příkazu "Search", kde za hodnotu parametru "searchValue"stačí dát symbol "*".

```
https://www.blueworkslive.com/scr/api/Search?version=20150930&
  searchValue=*&searchFieldName=process_name&returnFields=*
```

Tento postup je již nezávislý na počtu procesů uživatele, a jsou potřeba 2 sekundy pro obdržení informace o procesu (resp. 4 sekundy pro aktivitu). Nevýhodou tohoto přístupu je ztráta informace o datu modifikace a uživateli, který daný proces naposledy upravoval. Pokud budeme chtít tento nedostatek eliminovat, potřebujeme navíc 2 sekundy pro zaslání příkazu "Followed Processes". Tedy potřebujeme 4 sekundy celkem pro libovolný případ, ať už se jedná o aktivitě nebo procesu. Je zřejmé, že tento postup o mnohém efektivnější, něž původně implementovaný. Vyhodnocením předchozího tvrzení ve formě grafu se zabývá sekce 5.3.

Kapitola 5

Výsledky

V této práci jsme probrali průběh vývoje OSLC adaptéru, integrujícího produkty IBM Blueworks Live a IBM Rational Team Concert. V kapitole 3 jsou uvedené hlavní požadavky, tykající se vytvořené aplikace. Průběh řešení některých z nich je popsán v kapitole 4. V následujících sekcích si probereme výsledky po splnění jednotlivých požadavků.

5.1 Refaktorování a rozšíření oblasti funkčnosti adaptéru na CCM doménu

Proces refaktorování je jeden ze splněných požadavků, které usnadní další změny a rozšíření funkcionality vytvořeného adaptéru. Výsledkem této činnosti je kompletní úprava všech JSP souborů a Java tříd s výhradou těch, které řeší mechanismy autorizaci uživatele na serveru Blueworks Live a práci s bezpečnostními tokeny potřebnými pro komunikaci s programem RTC.

Požadavek na rozšíření oblasti funkčnosti adaptéru na CCM doménu je také zahrnut do této sekci, protože v našem případe úzce souvisí s procesem refaktorování. Jedná se o chybu při předání špatné hodnoty parametru do OSLC anotace, která byla eliminována přesně v etapě prohledávání a detailního rozboru zdrojového kódu.

Podrobnější přehled struktury adaptéru je popsán v sekci 5.2.

5.2 Výsledná struktura adaptéru

Vytvořený adaptér tvoří 4 základní adresáře:

- org.eclipse.lyo.server.oauth.consumerstore
- org.eclipse.lyo.server.oauth.core
- org.eclipse.lyo.server.oauth.webapp
- uk.ibm.com.oslc.blueworkslive

První tři prvky řeší OAuth autorizaci. Tato práce se nezabývá ani rozborem principu fungování, ani implementací tohoto protokolu. Všechny změny včetně

refaktorování se tykají pouze adresáře uk.ibm.com.oslc.blueworkslive, který má následující strukturu:

- uk.ibm.com.oslc.blueworkslive
  - src
    - test
    - main
      - java ... adresář obsahující zdrojové kódy vytvořeného adaptéru
      - resources ..... adresář obsahující nastavení OSLC adaptéru
      - webapp ..... adresář obsahující obrázky, .js a .jsp soubory
  - license
  - target ..... adresář obsahující výsledky kompilace aplikace
  - pom.xml ..... XML soubor popisující konfiguraci Maven projektu
  - ..... .classpath, .gitgnore a další pomocné soubory

Avšak pro nás je důležitější vnitřní uspořádání Java tříd. Jednou ze změn je odstranění třídy RequirementInfo, která byla zbytečným krokem v procesu transformaci dat. Následující diagram popisuje Java balíčky spolu s nejdůležitějšími datovými typy:

- src\main\java\uk\com\ibm\oslc
  - blueworks ..... soubor Java tříd pro práci se službou Blueworks Live
    - BlueworksProcess.java
    - BlueworksProcessActivity.java
    - BlueworksRestApiClient.java
  - exception ..... soubor Java tříd definující vlastní výjimky
    - RestException.java
    - UnauthorizedException.java
  - resources ..... soubor Java tříd definující základní datové typy
    - OSLCRequirement.java
  - rm ..... rozhraní pro implementaci poskytovatele služeb
  - services ..... soubor tříd, poskytující základní funkcionalitu adapteru
    - BlueworksRequirementService.java
  - servlet ..... Java třídy inicializující servlet
    - ApplicationManager.java
  - utils ..... pomocné třídy pro práci s HTTP protokolem

5.3 Dosazená výkonnost adaptéru

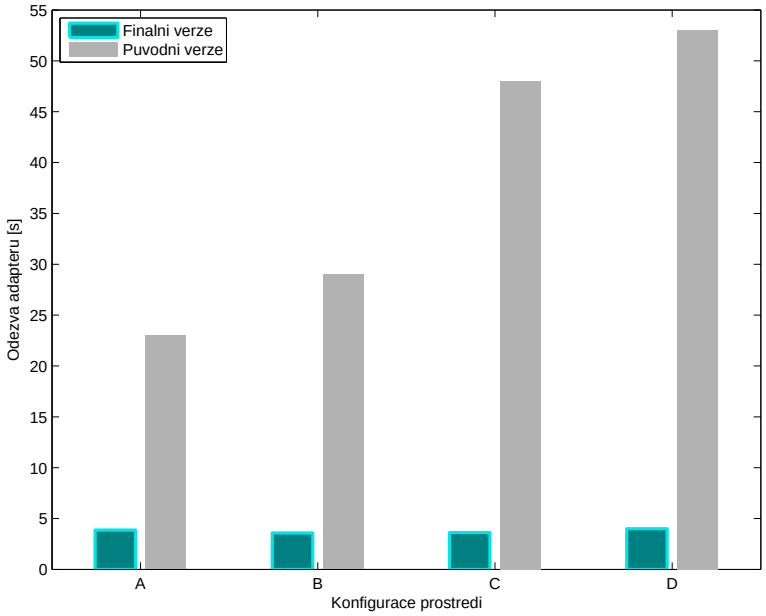
Optimalizace rychlosti adaptéru je jedna z hlavních úloh, která se řeší v této práci. Jedná se o efektivnější využíti API služby Bluework Live. Sekce 4.2 udává podrobnější přehled o průběhu řešení tohoto problému.

Předtím, než si ukážeme výsledné sloupcové grafy, definujeme pojem konfigurace prostředí. Tato formulace odpovídá prostředí, ve kterém byl otestován vytvořený adaptér. Hlavním důvodem pro zavedení zmíněného pojmu je ukázka jak závislosti původní verze adaptéru, tak i nezávislost výsledného na

počtu procesů a aktivit, ke kterým uživatel má právo přístupu. Obě aplikace jsou otestované v následujících testovacích prostředích:

- Konfigurace A - jedná se o prostředí obsahující přesně jeden proces, který tvoří 3 aktivity.
- Konfigurace B - jedná se o prostředí obsahující přesně jeden proces, který tvoří 5 aktivit.
- Konfigurace C - jedná se o prostředí obsahující přesně jeden proces, který tvoří 10 aktivit.
- Konfigurace D - jedná se o prostředí obsahující již dva procesy. Každý proces je tvořen pěti aktivitami.

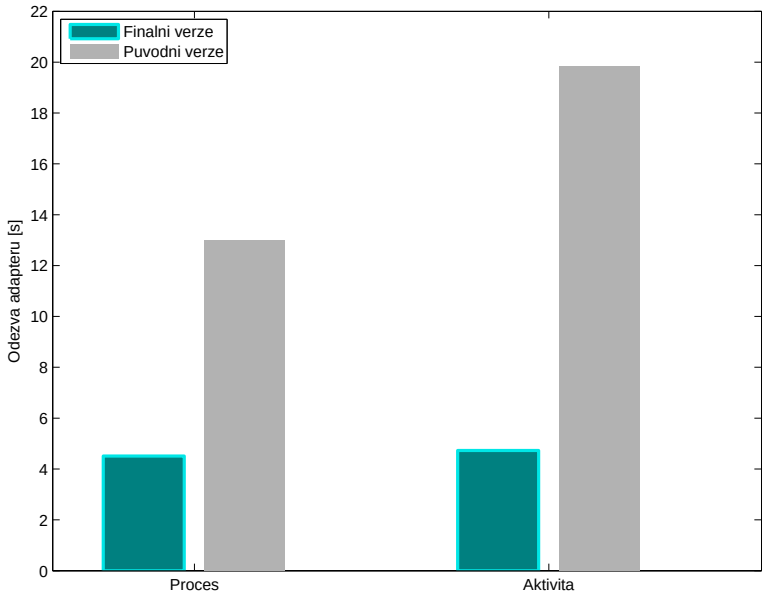
Vzhledem k nízké efektivitě původní verze adaptéru, tyto konfigurace jsou velmi zjednodušenými příklady oproti reálným situacím, kde proces může být tvořen několika desítkami aktivit a uživatelské prostředí obsahuje nejeden pár takových procesů. Graf 5.1 znázorňuje výkonnost (tj. rychlost hledání procesů) obou verzí adaptéru. Z výsledků je zřejmě, že upravený adaptér



Obrázek 5.1: Rychlost odezvy adaptéru v závislosti na testovacím prostředí

nezávisí na uživatelském prostředí. A to odpovídá teoretickým předpokladům, které jsou popsáné v sekci 4.2.1. Co se týče původní verze, z grafu je vidět, jak počet aktivit a zejména procesů ovlivňuje odezvu, můžeme takže konstatovat, že se i při zjednodušených testovacích podmínkách adaptér nedá považovat za použitelný.

Zvýšení rychlosti generování náhledových stránek (UI preview) je dalším optimalizačním úkolem. V sekci 4.2.2 je vysvětlen postup řešení problému.



Obrázek 5.2: Rychlost generování náhledové stránky (UI preview)

Graf 5.2 odpovídá předpokládaným výsledkům. Rychlost odezvy upraveného adaptéru zase nepřekračuje stanovenou hranici 8 sekund.

Z uvedených grafů je zřejmé, že finální verze adaptéru je o mnoho rychlejší než původní analog. Navíc jsou splněné požadavky na optimalizaci aplikace, tzn. že stanovená hranice 8 sekund není překročená. Nicméně, existují i další cesty jak dosáhnout ještě lepších výsledků. Jeden z nápadu je zavedení datové struktury (např. HashMap), která bude dočasně uchovávat informaci o vyhledaných procesech. Tato data se dají použit při opakovaném hledaní téhož procesu. Avšak hlavní problém v tomto případě je zajištění aktuálnosti uchovávané informace. Aplikaci tohoto přístupu se plánuje v budoucnu.

Kapitola 6

Závěr

Cílem práce byla úprava a optimalizace již existujícího řešení pro zajištění integrace mezi cloudovou službou IBM Blueworks Live a on-premise systémem IBM Rational Team Concert. Zvolená integrační metoda je založena na standardech vyvinutých OSLC komunitou.

Nejprve byla provedena analýza požadavků, která se týká základní funkcionality adaptéru. Splněné byly hlavní požadavky na rozšíření oblasti funkčnosti na CCM doménu a optimalizaci rychlosti adaptéru, aby byl použitelný koncovým uživatelem, tedy čas se zkrátil na konstantní dobu 5 sekund nezávislou na počtu procesů, ke kterým má uživatel práva přístupu.

Dalším pokračováním práce bude nasazení vytvořené aplikace v prostředí zákazníka, odladění případných chyb a doimplementování méně důležitých požadavků 3 a 4 (viz kapitola 3), na které nezbyl čas. Za vyzkoušení stojí i nápad, jak se dá zvýšit rychlost adaptéru. A to zavedením datové struktury, kde se bude dočasně ukládat informace o vyhledaných procesech (viz sekce 5.3).

Literatura

[1] Hurwitz J., Kaufman M. and Halper F. 2012. *Cloud for dummies*. New-York: John Wiley & Sons, Inc.

[2] Cheemalapati, et al., 2015. *Hybrid Cloud Data and API Integration*. [e-book] IBM ITSO. Dostupné z: IBM Redbooks `http://www.redbooks.ibm.com/` [Naposledy navštíveno 7.1.2016]

[3] Microsoft, 2011. *Rise of the Cloud Ecosystems*. [online obrázek]. Dostupné z: `https://blogs.msdn.microsoft.com/dachou/2011/03/16/rise-of-the-cloud-ecosystems/` [Naposledy navštíveno 25.5.2016]

[4] Blueworks Live, 2016. *Getting started with process blueprints*. [online obrázek]. Dostupné z: `https://ibm.blueworkslive.com/scr/docs/bwl/topics/gs_blueprints.html` [Naposledy navštíveno 25.5.2016]

[5] King, et al., 2014. *Process Discovery Best Practices*. [e-book] IBM ITSO. Dostupné z: IBM Redbooks `http://www.redbooks.ibm.com/` [Naposledy navštíveno 7.1.2016]

[6] Gothe, et al., 2008. *Collaborative Application Lifecycle Management with IBM Rational Products*. [e-book] IBM ITSO. Dostupné z: IBM Redbooks `http://www.redbooks.ibm.com/` [Naposledy navštíveno 7.1.2016]

[7] OSLC Wiki, 2013. *Open Services for Lifecycle Collaboration Core Specification Version 2.0*. [online] Dostupné z: `http://open-services.net/bin/view/Main/OslcCoreSpecification` [Naposledy navštíveno 22.05.2016]

[8] OSLC Wiki, 2010. *Open Services for Lifecycle Collaboration Change Management Specification Version 2.0*. [online] Dostupné z: `http://open-services.net/bin/view/Main/CmSpecificationV2` [Naposledy navštíveno 26.05.2016]

[9] OSLC Wiki, 2012. *Open Services for Lifecycle Collaboration Requirements Management Specification Version 2.0*. [online] Dostupné z: `http://open-services.net/bin/view/Main/RmSpecificationV2` [Naposledy navštíveno 22.05.2016]

[10] Wikipedia, 2013. *HTTP*. [online] Dostupné z: `https://ru.wikipedia.org/wiki/HTTP` [Naposledy navštíveno 14.5.2016]

[11] Rodriguez A., 2008. *RESTful Web services: The basics*. [online] IBM Corp. Dostupné z: `https://www.ibm.com/developerworks/library/ws-restful/` [Naposledy navštíveno 14.5.2016]

[12] Dr. Westphal M., 2016. *Do more with the IBM Blueworks Live REST API*. [online] IBM Corp. Dostupné z: `http://www.ibm.com/developerworks/bpm/library` [Naposledy navštíveno 14.5.2016]

[13] Amrhein D. and Gallardo N., 2010. *Create RESTful Web services with Java technology*. [online] IBM Corp. Dostupné z: `http://www.ibm.com/developerworks/webservices/library/wa-jaxrs/index.html` [Naposledy navštíveno 15.5.2016]

[14] Eclipse Wiki, 2013. *Lyo OSLC4J*. Dostupné z: `http://wiki.eclipse.org/Lyo/LyoOSLC4J` [Naposledy navštíveno 16.5.2016]

Příloha A

Obrázky

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: Ruslan B a k e y e v

Studijní program: Kybernetika a robotika (bakalářský)

Obor: Robotika

Název tématu: Vývoj integračního adaptéru pro IBM RTC a Blueworks Live

Pokyny pro vypracování:

1. Vývoj adaptéru pro zajištění integrace mezi systémy RTC (Rational Team Concert) a BWL (Blueworks Live).
2. Adaptér bude vyvinut v jazyce Java a bude využívat RTC Server API a BWL REST API.
3. Cílový runtime adaptéru bude JEE aplikační server (Tomcat).
4. Pro vývoj adaptéru bude využit Rational Application Developer (IDE).
5. Adaptér bude vyvinut jako polling adaptér, který bude v daném intervale volat BWL REST API a aktualizovat příslušná data v RTC.

Seznam odborné literatury:

[1] King Joshua et al.: Process Discovery Best Practices, www.ibm.com/redbooks
[2] Gothe Mats et al.: Collaborative Application Lifecycle Management, www.ibm.com/redbooks
[3] Cheemalapati S. et al.: Hybrid Cloud Data and API Integration, www.ibm.com/redbooks

Vedoucí bakalářské práce: Ing. Martin Samek

Platnost zadání: do konce letního semestru 2016/2017

L.S.

prof. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 2. 2. 2016