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

Diplomová práce

Vytváření a ukládání popisu webových služeb v úložišti CRCE

Plzeň 2015 David Pejřimovský

Prohlášení

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

V Plzni dne 24. června 2015

Bc. David Pejřimovský

Abstract

This diploma thesis deals with creation and storing of web services descriptions. The descriptions are created in unified representation and stored in form of meta-data into CRCE repository. This thesis consists of two parts. Theoretical part deals with analysis of service oriented architecture and web service general principles. Practical part contains breakdown of known web service technologies from which a set of representative web service technologies is selected. An extension module is created for CRCE repository which allows creation and storing of web service descriptions from mentioned set of selected technologies.

Keywords: service oriented architecture, web services, web service description, web service representation, CRCE

Abstrakt

Tato diplomová práce se zabývá vytvářením a ukládáním popisu webových služeb. Tento popis je vytvářen v jednotné reprezentaci a je ukládán v podobě metadat do úložiště CRCE. Práce je rozdělena na dvě části. Teoretická část obsahuje analýzu servisně orientované architektury a obecných principů webových služeb. V praktické části je proveden rozbor známých technologií webových služeb a pro reprezentativní množinu těchto technologií je vytvořen rozšiřující modul do úložiště CRCE, který umožňuje popisy webových služeb, spadajících pod vybrané technologie, vytvářet a ukládat.

Klíčová slova: servisně orientovaná architektura, webové služby, popis webové služby, reprezentace webové služby, CRCE

Obsah

1 Úvod

V rámci výzkumu vznikl na Katedře informatiky a výpočetní techniky ZČU v Plzni model pro reprezentaci modulárních softwarových aplikací, který mimo jiné tvoří datový základ úložiště CRCE (Component Repository supporting Compatibility Evaluation). Cílem této práce je případná úprava tohoto datového modelu tak, aby byl schopný reprezentovat obecný popis webových služeb a následná úprava úložiště CRCE způsobem, který umožní tyto obecné popisy webových služeb získávat a ukládat. Po této úpravě bude možné v úložišti ukládat mnoho popisů reprezentujících různé technologie webových služeb a vznikne tak možnost upravit úložiště tak, aby se v něm dali generovat klienti schopní komunikace s popsanými webovými službami.

Prvotním cílem této práce je tedy analýza technologií webových služeb, včetně souvisejících standardů, definice obecných zákonitostí v této oblasti, výběr vhodné reprezentativní množiny vybraných technologií webových služeb a následná případná úprava zmíněného datového modelu tak, byl schopný tyto vybrané technologie reprezentovat.

Pro výběr konkrétní vhodné množiny technologií webových služeb je třeba tyto vhodně definovat, kategorizovat a porovnat. Při porovnání bude hlavním kritériem pro výběr konkrétní technologie webové služby existence a kvalita tzv. popisného schématu (popisné schéma dané webové služby definuje konkrétní formát komunikace podle protokolu této webové služby a lze z něj tedy pro konkrétní webovou službu vytvořit její popis).

Následným cílem je implementace rozšiřujícího modulu v úložišti CRCE pro získávání a ukládáni popisu webových služeb z vybrané množiny technologií webových služeb, včetně možnosti ovládat tuto funkcionalitu přes existující webové rozhraní úložiště. Po implementaci modulu bude ověřena jeho funkčnost vytvořením a uložením popisu několika netriviálních konkrétních webových služeb pro každou technologii webové služby z vybrané množiny technologií webových služeb.

Teoretická část práce se zabývá analýzou servisně orientované architektury a webových služeb, ze které jsou vyvozeny obecné zákonitosti platné pro webové služby (kapitola 2). Následně jsou s ohledem na tyto zákonitosti popsány technologie známých webových služeb, ze kterých je vybrána vhodná reprezentativní množina (kapitola 3).

V praktické části práce je jednotná reprezentace popisů webových služeb z množiny vybraných technologií webových služeb namapována na entity stávajícího datového modelu (kapitola 4) a je popsán současný stav projektu úložiště CRCE (kapitola 5). Do úložiště je navržen rozšiřující modul (kapitola 6) a je otestována jeho funkčnost (kapitola 7).

2 Servisně orientované architektury

V této kapitole se postupným upřesňováním a skládáním obecných pojmů dojde k definici termínu *webová služba*. Také budou stručně popsány některé trendy a vývoj v oblasti webových služeb k dnešnímu datu.

2.1 Služba

Abychom mohli uspokojivě popsat architekturu orientovanou na služby / servisně orientovanou architekturu (SOA), musíme nejprve zadefinovat pojem *služba* tak, jak jej z architektonického hlediska chápe informatika.

Organizace pro vývoj strukturovaných informačních standardů *OASIS* (Organization for the Advancement of Structured Information Standards) v roce 2006 službu v rámci svého dokumentu *OASIS SOA Reference Model*[1] definovala jako:

A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.

tzn. *mechanismus k zpřístupnění jedné nebo více funkcionalit, ke kterým je přístup zprostředkován podle předepsaného rozhraní a je prováděn konzistentně s omezeními a zásadami specifikovanými v popisu služby.*

Takto definovaný pojem byl částí odborné veřejnosti přijat, ale jak se postupně k SOA referenčnímu modelu od OASIS objevovaly konkurenční dokumenty, objevily se i nové definice pojmu služba. Za největšího konkurenta lze v tomto směru považovat průmyslové konsorcium *The Open Group*, které definuje pojem služba takto[2]:

A service is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit; provide weather data, consolidate drilling reports, etc.) and:

- *Is self-contained,*

[1] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm
[2] http://www.opengroup.org/soa/source-book/togaf/soadef.htm

- *May be composed of other services,*
- *Is a "black box" to consumers of the service.*

tzn. *Služba je logická reprezentace opakující se podnikové aktivity, která má specifikovaný výsledek (např. zjistit zákazníkův kredit, poskytnout data o počasí, konsolidovat těžební hlášení) a:*

- *je samostatná,*
- *může se skládat z jiných služeb,*
- *vůči konzumentům služby se jeví jako "black box".*

Podnikovou (business) aktivitou se zde myslí jakákoliv aktivita, ve které jsou zainteresováni lidé. Nikoliv pouze aktivity spjaté s komerční praxí.

At' již je služba zadefinována jakkoliv, je třeba rozlišovat entitu, která službu využívá (*konzument služby / klient*) a entitu, která ji poskytuje (*poskytovatel služby / služba*). I v případě, že je z jednoho umístění poskytováno vícero funkčně spjatých služeb (například: součet dvou čísel, průměr N čísel, řešení kvadratických rovnic, apod.), je zvykem všechny tyto dílčí služby souhrnně stále označovat jednotným číslem (například: poskytovatel služby matematických úkonů, služba pro matematické výpočty).

Činnost některých služeb může být závislá na jiných službách. Je tedy třeba rozlišovat mezi *atomickou službou*, která ke své činnosti žádnou jinou externí službu nepotřebuje, a *složenou službou*, která během své činnosti volá alespoň jednu jinou službu.

Hierarchický popis vazeb mezi složenými službami sahající až k jednotlivým atomickým službám, který aktivně kontroluje jejich interakci, se nazývá *orchestrace* (nebo také *proces / podnikový proces*). Pokud je však dvě a více služeb (složených či atomických) implementováno tak, aby mezi sebou interagovaly bez centralizované kontroly, mluvíme o *choreografii*. Dá se také říci, že choerografii mezi službami lze realizovat vytvořením orchestrace pro každého účastníka dané choreografie (orchestrace daného účastníka v dané choreofrafii bude tedy zahrnovat i služby, se kterými v rámci této choreografie tento účastník přímo komunikuje). Více se tématu orchestrace a choreografie věnuje článek [Dijkman – Dumas(2004)].

2.2 Servisně orientovaná architektura

Pokud tedy chceme v informatice něco označit jako *servisně orientované*, hovoříme o snaze vytvářet software ve formě služeb. K dnešnímu datu neexistuje žádný průmyslový standard, který by definoval *servisně orientovanou architekturu* (SOA), ale existuje mnoho nezávislých definic, které se tuto architekturu snaží popsat. Ty hlavní, významnější definice nyní uvedeme.

Organizace OASIS definuje servisně orientovanou architekturu jako:

A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations.

tzn. *Paradigma pro organizaci a využití distribuovaných funkcionalit, které mohou být pod správou různých vlastnických domén. Poskytuje uniformní prostředky k nabídce, objevení, interakci s a používání funkcionalit k dosažení efektu konzistentního s měřitelnými počátečními podmínkami a očekáváním.*

Průmyslové konsorcium The Open Group definuje servisně orientovanou architekturu jako:

Service-Oriented Architecture (SOA) is an architectural style that supports service-orientation. Service-orientation is a way of thinking in terms of services and service-based development and the outcomes of services. A service:

- *Is a logical representation of a repeatable business activity that has a specified outcome (e.g., check customer credit, provide weather data, consolidate drilling reports)*
- *Is self-contained*
- *May be composed of other services*
- *Is a "black box" to consumers of the service*

tzn. *Servisně orientovaná architektura (SOA) je architektonický styl, který podporuje servisní orientaci. Servisní orientace je způsob myšlení ve formě služeb a služebně orientovaném vývoji a výsledků služeb. Služba:*

- *je logická reprezentace opakující se podnikové aktivity, která má specifikovaný výsledek (např. zjistit zákazníkův kredit, poskytnout data o počasí, konsolidovat těžební hlášení)*
- *je samostatná,*
- *může se skládat z jiných služeb,*
- *vůči konzumentům služby se jeví jako "black box".*

Řada dalších definic popisuje SOA v podobném duchu. Lze tedy vyjmenovat tyto hlavní, většině definic společné vlastnosti servisně orientované architektury:

- **Standardizovaný kontrakt služby** - Tato vlastnost říká, že služby budou s okolím komunikovat jasně vymezeným způsobem a ve vymezeném datovém formátu. Některé jiné zdroje (např. [Weiss – Rychlý(2007)]) uvádějí jako vlastnost i **Nezávislost na platformě**, ale ta je již automaticky vymezena touto obecnější vlastností. V terminologii této práce lze tuto vlastnost také označit jako *existenci popisného protokolu* (viz sekce 2.5).
- **Slabé vazby mezi službami** - Služby mezi sebou udržují minimální potřebnou vazbu. To umožňuje jejich snadnou obměnitelnost bez nutnosti měnit zároveň i okolní příbuzné služby.
- **Princip abstrakce** - Cokoliv nad rámec kontraktu služby (popisného protokolu) je ve službě zapouzdřeno a skryto před okolním světem.
- **Znovupoužití** - Služby by měly být vytvářeny obecně s ohledem na budoucí použití v jiném kontextu. Stejnými principy be se měl řídit i návrh rozkladu složitějšího procesu na jednotlivé služby.
- **Nezávislost** - Služby mají kontrolu nad logikou, kterou obalují. Kontrola nad touto logikou je jak návrhářského charakteru, tak i kontrolou během vykonávání služby.
- **Bezstavovost** - Služby udržují minimální míru stavovosti vůči svým konzumentům. To snižuje potřebné zdroje a podporuje slabou vazbu, a služba tak spolehlivě vyhoví více požadavkům.
- **Princip objevitelnosti** - Služby jsou popsány přenosu-schopnými metadaty, díky kterým mohou být snadno objevovány a interpretovány.

- **Princip skládání** - Jednotlivé služby v servisně orientované architektuře mohou v rámci své funkcionality využívat jiné, jednodušší služby. Více viz sekce 2.1.

Výše zmíněné vlastnosti davájí SOA především tyto výhody:

- snadná udržovatelnost
- znovupoužitelnost
- snadná a rychlejší rozšiřitelnost
- přehlednost

Mnoho aspektů aplikovaných v SOA není nových, ale používalo se již v technologiích a architekturách, které existovaly před jejím vznikem (především technologie *CORBA*, *DCOM*, ale i *objektově orientovaná architektura* (OOA) a *objektově orientovaný design* (OOD)).

2.3 Komunikační vzory

Poskytovatel služby a *konzument služby* spolu mohou komunikovat různým způsobem. Tyto různé způsoby komunikace nám umožní služby klasifikovat do skupin, v jejichž rámci lze služby dále rozlišovat podle dodatečných kritérií. Použitý komunikační vzor odráží aplikační požadavky služeb a jednotlivé vzory se od sebe liší formátem přenášených dat, jejich adresací i způsobem vyvolání služby u poskytovatele služby.

Různé zdroje definují vždy (byť s malými obměnami v chování a pojmenování) 3 základní komunikační vzory:

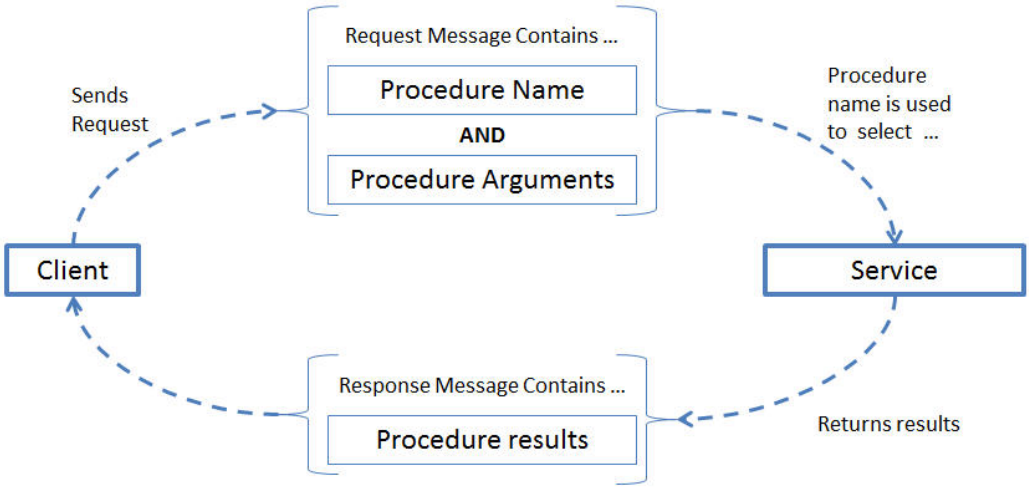
- *volání vzdálené procedury* (*RPC*),
- *zasílání zpráv* (*Messaging*),
- *zdrojově orientovaná komunikace* (*Resource API*).

[pro(2012)] uvádí navíc ještě jeden komunikační vzor:

- *streamování* (*Streaming*).

2.3.1 Volání vzdálené procedury

Při používání tohoto komunikačního vzoru posílá klient směrem ke službě požadavek, který obsahuje název procedury (viz `Procedure Name` na Obr. 2.1), která se má vykonat a parametry (viz `Procedure Arguments` na Obr. 2.1), které se proceduře při jejím spuštění předají. Po vykonání procedury pošle služba klientovi zpátky odpověd s výsledkem (viz `Procedure results` na Obr. 2.1). Termín *procedura* v našem kontextu reprezentuje nějakou konkrétní službu poskytovanou poskytovatelem služby.



Obrázek 2.1: Struktura zasílaných dat v případě vzdáleného volání metody (zdroj: [Daigneau(2012)])

Vzdálenou proceduru / službu lze volat jak synchronně tak asynchronně. Při synchronním volání klient pošle službě svůj požadavek, služba ho zpracuje a pošle klientovi zpátky odpověd s výsledkem (zda je při čekání na odpověd s výsledkem klient blokován, nebo může provádět jinou činnost s touto problematikou nijak nesouvisí). Při asynchronním volání klient opět pošle službě svůj požadavek, ale služba ho pouze vezme na vědomí, uloží si jeho definici do seznamu požadavků ke zpracování a typicky pošle klientovi odpověď, která ovšem neobsahuje výsledek, ale pouze potvrzení o přijetí požadavku (a ve většině implementací i nějaký identifikátor, který byl přijatému požadavku ze strany služby přiřazen). Poté je službou ve volném (nebo jinak stanoveném) čase požadavek zpracován a klient si jej může (typicky za použití jiné, k tomu určené procedury / služby) u poskytovatele služby vyzvednout (přičemž

je ve většině implementací za pomoci přiřazeného identifikátoru požadavku na straně poskytovatele služby vyzvednut výsledek ze seznamu zpracovaných požadavků (tj. seznamu výsledků) a je poslán jako odpověď této následně volané procedury / služby).

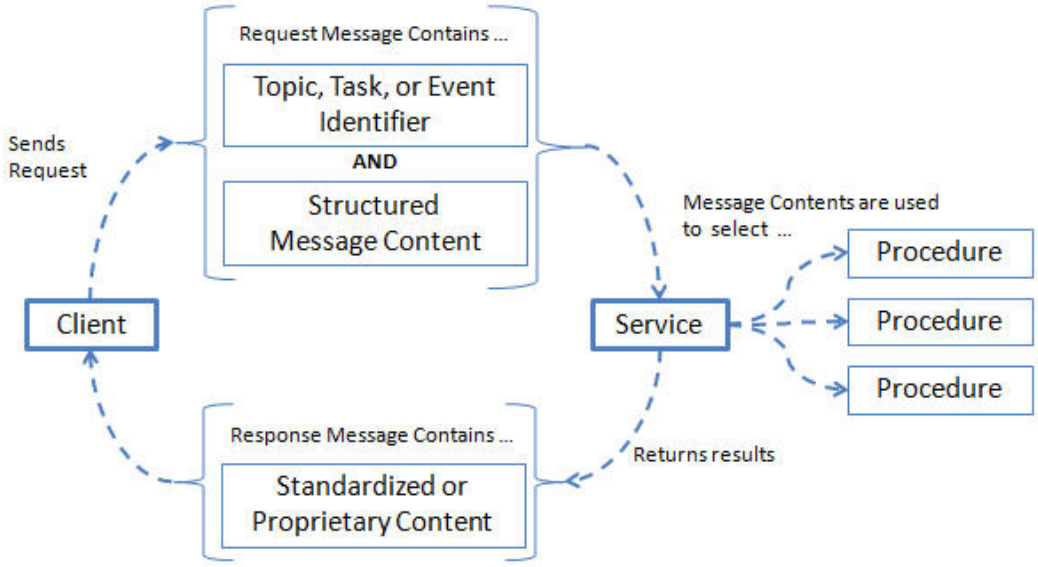
Synchronní volání se hodí zvláště v případech, kdy je výpočetní čas pro zpracování požadavků dostatečně malý a implementace asynchonicity by přinesla pouze zbytečně složitější systém. Asynchronní volání použijeme v případě, že je výpočetní čas pro zpracování požadavků příliš velký nebo je na poskytovatele služby vyvíjen příliš velký tlak (ať již průměrný nebo nárazový) z hlediska počtu obsluhovaných klientů (ti by se tak v synchronním případě zbytečně blokovali).

Typickou technologií z oblasti webových služeb, která komunikuje na principu volání vzdálené metody je např. *XML-RPC*.

2.3.2 Zasílání zpráv

Při používání tohoto komunikačního vzoru posílá klient směrem ke službě požadavek (v tomto případě ho nazýváme zprávou), který obsahuje pouze data (viz `Structured Message Content` na Obr. 2.2) a na základě struktury nebo obsahu dat (viz `Topic, Task, or Event Identifier` na Obr. 2.2) služba sama vyhodnotí jakou proceduru (tzn. konkrétní službu) vyvolat. Po vykonání metody pošle služba klientovi zpátky odpověd (zprávu) s výsledkem (viz `Standartized or Proprietary Content` na Obr. 2.2).

V tomto případě je tedy klient odstíněn od nutnosti volit název konkrétní služby. Jednou z výhod tohoto přístupu je, že poskytovatel služby může své konkrétní poskytované služby vytvářet a rušit a tyto změny si nevyžádají změny na straně klientů (v předchozím případě vzdáleného volání procedur by se klienti museli dozvědět aktuální jména volatelných procedur / poskytovaných služeb).



Obrázek 2.2: Struktura zasílaných dat v případě zasílání zpráv (zdroj: [Daigneau(2012)])

Zasílat zprávy lze, obdobně jako při volání vzdálené procedury, synchronně i asynchronně. V praxi však u tohoto komunikačního vzoru převažuje asynchronní způsob komunikace, jelikož koncept zasílání zpráv (u kterých není potřeba spolu logicky párovat strukturu požadavků a odpovědí pro jednotlivé procedury / služby) se k tomuto způsobu komunikace hodí více.

Typickou technologií z oblasti webových služeb, která komunikuje na principu zasílání zpráv je např. *SOAP*.

2.3.3 Zdrojově orientovaná komunikace

Zasílání zpráv v předchozím komunikačním vzoru zbavuje klienta povinnosti znát v daný okamžik jména všech aktuálních služeb, čímž dochází k tzv. *volnějšímu vázání*[3] mezi klientem a službou. V případě zdrojově orientované komunikace je toto vázání rozvolněno ještě více: mezi klientem a službou vzniká nová abstrakce v podobě zdrojů. Zdroje může tvořit cokoliv: soubory, řádky z databáze, kolekce obecných dat, dokonce např. i programy nebo

[3]http://en.wikipedia.org/wiki/Loose_coupling

podnikové procesy. Hlavní je, že zdroje jsou principielně voleny tak, aby jejich typ nebylo třeba měnit a jsou službou reprezentovány skrze *URI*[4] (viz `URI` na Obr. 2.3). Díky tomu mohou klienti komunikovat s těmito zdroji a na straně poskytovale služeb jsou vazby mezi službami a zdroji nezávislé vůči klientům a změny těchto vazeb se nebudou muset na klienty promítat.

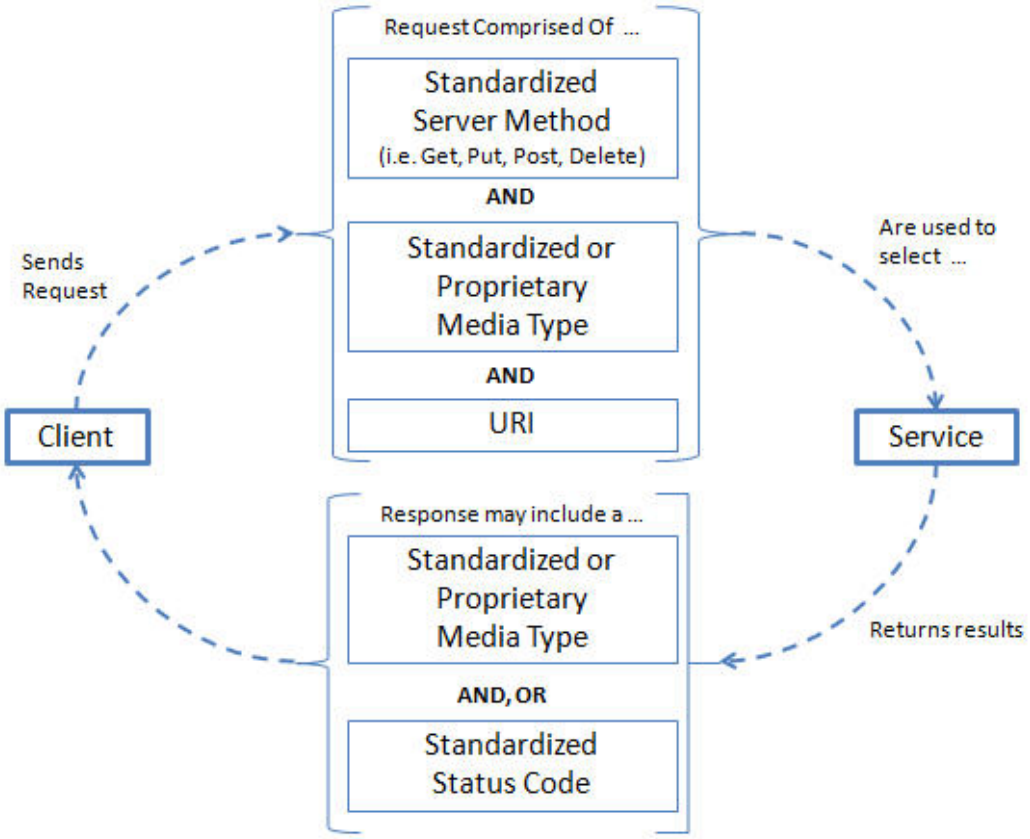
Nyní je třeba popsat, jak jsou v případě tohoto komunikačního vzoru služby navázány na vytvořené zdroje. U všech zdrojů se po jejich vytvoření objevuje opakující se motiv: je třeba je nějakým způsobem skrze služby vytvářet, číst, upravovat a mazat (tzv. *CRUD*[5]). Proto bylo v návrzích tohoto komunikačního vzoru využito HTTP protokolu, resp. jeho metod, které byly přímo namapovány na tyto operace se zdroji a klient je ve svých požadavcích využívá (viz `Standartized Server Method` na Obr. 2.3). Pro vytvoření a úpravu zdroje byla zvolena metoda *PUT*, pro čtení metoda *GET*, pro smazání metoda *DELETE* a pro speciální operace mimo tyto elementární operace nad zdroji byla domluvou vyhrazena metoda *POST*. Více o namapování HTTP metod na manipulaci se zdroji lze najít v [Allamaraju(2010), str. 13-27].

Dále je často vhodné mít pro manipulaci se zdroji na výběr z vícero formátů dat (např. řádek z databáze reprezentovaný jako zdroj lze reprezentovat ve formátu *JSON*, *XML*, binárně, apod.). Opět bylo využito HTTP protokolu, konkrétně toho, že umí v hlavičce rozlišovat různé formáty dat pomocí vlastností *Accept* a *Content-Type* (viz `Standartized Proprietary Media Type` na Obr. 2.3). Zvolený formát je pak použit pro reprezentaci dat v požadavcích a v odpovědích. K získání konkrétně podporovaných formátů dat lze využít HTTP metodu *HEAD* a k získání konkrétně podporovaných operací nad daným zdrojem lze využít HTTP metodu *OPTION*.

Stejně tak je z HTTP protokolu převzata funkcionalita statových kódů (viz `Standartized Status Code` na Obr. 2.3), díky kterým lze z odpovědi rozlišit výsledek požadované operace.

[4] http://www.w3.org/TR/uri-clarification/

[5] http://en.wikipedia.org/wiki/Create,_read,_update_and_delete



Obrázek 2.3: Struktura zasílaných dat v případě zdrojově orientované komunikace (zdroj: [Daigneau(2012)])

V případě předchozích komunikačních vzorů: vzdáleného volání procedur (resp. zasílání zpráv), by při snaze o podobný zdrojově založený přístup mezi klientem a službou vznikalo zbytečně velké a nepřehledné množství procedur / služeb (resp. typů zasílaných zpráv), ve kterých by se navíc zpravidla opakovala velká část logiky.

Představme si například množinu služeb vytvořenou v nějaké firmě pro správu dvou entit (firemních faktur a zákazníků) podle zde uvedených komunikačních vzorů. Budeme chtít tyto entity (ve shodě s CRUD funkcionalitou) vytvářet, číst, upravovat a mazat.

V případě vzdáleného volání procedur dopadne situace nejhůře: Pro každou CRUD funkci každé entity vznikne samostatná procedura / služba obslu-

hující tuto eventualitu. V tomto případě by tedy vzniklo celkem osm procedur / služeb (*createInvoice*, *updateCustomer*, apod.), které bude při nějaké změně potřeba všechny postupně upravovat a navíc o všech těchto změnách i nějakým způsobem notifikovat klienty (aby znali nové názvy procedur / služeb a pro každou proceduru / službu znali správný počet, pořadí a typ parametrů včetně návratové hodnoty).

V případě zasílání zpráv je situace o něco lepší, ale pouze v tom smyslu, že lze typy zasílaných zpráv vytvořit tak, že budou nezávislé na definici skutečně volané služby, kterou podle typu zprávy zavolá poskytovatel služby sám (klient tedy v tomto případě nemusí znát přesnou definici procedury / služby, ale stále musí znát definici jím zasílaných zpráv). Počet vytvořených typů zpráv bude stále minimálně osm - pro každou CRUD funkci každé spravované entity (navíc pro každý tento typ zprávy bude potřeba vytvořit další typ zprávy, který bude poskytovatel služby zasílat klientovi jako odpověď).

V případě zdrojově orientované komunikace je ale situace výrazně lepší, protože jsou zadefinovány dva zdroje (faktury a zákazníci), kterým je přiřazena pevná URI (např. *http://adresa-serveru/rest/invoice* a *http://adresa-serveru/rest/customer*) a pomocí HTTP metod je s nimi operováno. Např. pro zobrazení faktury s identifikačním číslem *a37c428* je vyvolána HTTP metoda GET nad URI *http://adresa-serveru/rest/invoice/a37c428*, vytvoření záznamu o novém zákazníkovi je docíleno vyvoláním HTTP metody PUT na URI *http://adresa-serveru/rest/customer*, která ve svém těle nese požadované parametry apod. Jediné, co musí klient v tomto případě bezpodmíněčně znát, jsou URI zadefinovaných zdrojů (a ty jsou, spolu s dalšími informacemi, často zjistitelné z nějakého primárního URI). Rozdíly mezi těmito komunikačními vzory vyniknou lépe, pokud je spravovaných entit o mnoho více než pouze dvě, jako v tomto příkladu. Při velkém množství spravovaných entit jsou vzdálené volání procedur a zasílání zpráv velice náročné na údržbu, změny v systému a ze strany klienta se hůře používají.

Některé zdrojově orientované komunikace mezi klientem a poskytovatelem služby lze označit jako tzv. *RESTfull*. K tomu, abychom mohli mluvit o *Representational state transfer* (REST) komunikaci, musí tato komunikace splňovat některé požadavky navíc (viz [Daigneau(2012), str. 45-46]):

- **Existence vztahu klient / server** - Tuto vlastnost splňují všechny zdrojově orientované kumunikace díky existenci konzumenta služby a poskytovatele služby.

- **Bezstavovost** - Každý požadavek musí být zpracovatelný bez ohledu na požadavky předchozí. Veškerá potřebná informace o stavu se drží na straně klienta.
- **Cachovatelné odpovědi** - Musí existovat principielní možnost cachovat odpovědi na některé typy požadavků.
- **Uniformní rozhraní** - Mezi všemi účastníky komunikace (konzument služby, poskytovatel služby, případně nějací prostředníci) musí existovat uniformní rozhraní pro komunikaci. Tuto vlastnost splňují všechny zdrojově orientované komunikace díky využití HTTP protokolu, resp. jeho metod.
- **Existence vrstev** - Tuto vlastnost splňují všechny zdrojově orientované kumunikace díky využití HTTP jakožto transportního protokolu (tento protokol je součástí vrstveného modelu pro síťovou komunikaci[6])
- **Code on demand** - Tato vlastnost říká, že funkcionalita klienta může být rozšířena spuštěním skriptů nebo pluginů získaných ze serveru. Tento požadavek je tedy kladen spíše na klienta, než-li na zdrojově orientovaný komunikační vzor.

Komunikovat s orientací na zdroje lze opět, obdobně jako v předchozích případech, synchronně i asynchronně. V praxi však u tohoto komunikačního vzoru převažuje synchronní způsob komunikace, jelikož HTTP metody volané nad konkrétním URI a navrácená odpověď s daty a se stavovým kódem přímo reprezentují zpracování daného požadavku, což potlačuje ideu asynchronního přístupu.

Typickou technologií z oblasti webových služeb, která komunikuje na zdrojově orientovaném principu je např. *Atom*.

2.3.4 Streamování

Tento komunikační vzor je složitější, než vzory dosud představené, a lze jej pomocí dosud představených elementárních komunikačních vzorů realizovat. Tento vzor spočívá v průběžném zásílání zpráv od poskytovatele služby směrem ke klientům v tzv. *datovém kanálu*. Obsah těchto zpráv (např. formát,

[6] http://tools.ietf.org/html/rfc1122

velikost, jejich počet za jednotku času, ...) lze upravovat pomocí tzv. *řídícího kanálu*, kterým klient směrem ke službě posílá požadavky na změnu komunikace (tyto požadavky mohou být potvrzované, nebo nepotvrzované). Komunikace v řídícím kanálu se zpravidla realizuje pomocí zasílání zpráv, ale není to podmínkou.

2.4 Webové služby

Webové služby jsou jedním z prostředků, jak implementovat servisně orientovanou architekturu. Definice termínu *webová služba* se různí a jsou více či méně přesné. Například W3C konsorcium webovou službu definuje takto[7]:

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

tzn. *Webová služba je systém navržený pro výměnu informací dvou strojů skrze síť. Má rozhraní definované ve strojově zpracovatelném formátu (konkrétně WSDL). Ostatní systémy komunikují s webovou službou způsobem popsaným v tomto rozhraní pomocí SOAP zpráv, typicky přenášenými za použití HTTP s XML serializací ve spojení s ostatními webovými standardy.*

Tato definice je pouze reprezentantem řady chybných definic, protože svazuje implementaci webové služby s použitím konkrétních protokolů (v tomto případě WSDL/SOAP) a opomíjí fakt, že webová služba v obecném pojetí je při implementaci nezávislá na použitých protokolech. Navíc se snaží doporučit transport dat v konkrétním formátu (v tomto případě XML), ale opět platí, že při vytváření webové služby můžeme zvolit libovolný vhodný formát, v jakém budou data přenášena.

Pokusme se tedy webovou službu definovat obecněji, bez závislosti na použitých protokolech. Jediným omezujícím kritériem by měla být schopnost služby využívat k přenosu dat HTTP protokol (už kvůli výše popsanému komunikačnímu vzoru *REST*, který HTTP protokol aktivně využívá). [Daigneau(2012)] definuje webové služby jako:

[7] http://www.w3.org/TR/ws-arch/#whatis

Web services provide the means to integrate disparate systems and expose reusable business functions over HTTP. They either leverage HTTP as a simple transport over which data is carried (e.g., SOAP/WSDL services) or use it as a complete application protocol that defines the semantics for service behavior (e.g., RESTful services).

tzn. *Webové služby poskytují prostředky, jak integrovat nesourodé systémy a vystavit opakovaně použitelné podnikové funkce skrze HTTP. HTTP je využíván čistě jako transportní protokol (SOAP/WSDL) nebo definuje sémantiku pro chování služby (RESTful services).*

2.5 Klasifikace webových služeb

V této sekci poukážeme na některé společné prvky a vlastnosti technologií webových služeb, abychom na základě těchto a dříve vymezených vlastností byli schopni jednotlivé technologie webových služeb mezi sebou porovnat.

Základem každé technologie webové služby je její protokol a většinou i schéma, které umožňuje popsat vlastnosti konkrétní implementace této technologie webové služby - nazveme je *protokol webové služby* a *popisné schéma*.

Protokol webové služby je těžištěm celé architektury a definuje, jakým způsobem se budou přenášet data mezi klientem a poskytovatelem služby. Jeho definice nepřímo definuje i konkrétní komunikační vzor potřebný pro přenášení těchto dat.

Z celkové množiny dat, které lze posílat mezi klientem a službou, definované protokolem webové služby, *popisné schéma* definuje podmnožinu dat, kterou lze mezi klientem a službou přenášet a také dává těmto přenášeným datům ontologii. Je třeba zmínit, že existence protokolu webové služby nevyžaduje existenci popisného schématu a pro některé technologie webových služeb (např. *XML-RPC*) popisné schéma neexistuje. V takových případech lze ovšem často pozorovat snahu komunity nebo jednotlivců o vytvoření jednoho nebo více vlastních popisných schémat (např. *Barrister RPC* pro technologii *JSON-RPC*).

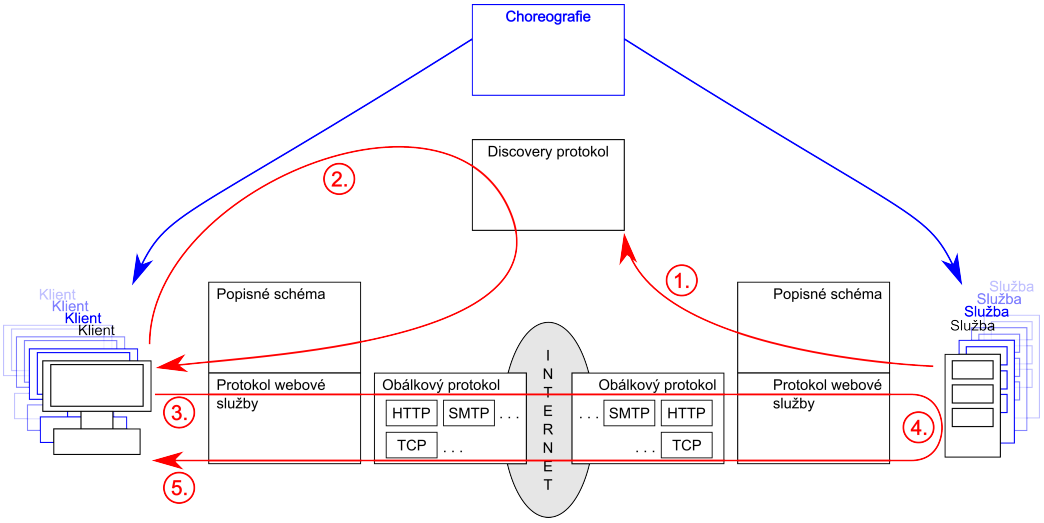
Pro některé technologie webových služeb jsou protokol webové služby s popisným schématem zadefinovány samostatně (např. *SOAP* se svým popisným schématem *WSDL*) a pro jiné jsou definovány v rámci jednoho souhrn-

ného protokolu (např. *JSON-WSP*). I v případě druhých zmiňovaných lze ale vždy na nějaké úrovni rozlišit, co patří pod doménu protokolu webové služby a co pod doménu popisného schématu.

Konkrétní protokol použitý pro přenos dat po síti mezi klientem a službou budeme nazývat *obálkový protokol*. Obálkový protokol je použitý čistě pro transport dat vytvořených protokolem webové služby v souladu s pravidly popisného schématu a oproti protokolu webové služby nebo popisného schématu v něm není definována žádná informace o tom, jaká data přenášet a kdy je přenášet. V sekci 2.4 jsme technologii webové služby definovali tak, že musí být schopná pro transport dat (tedy jako obálkový protokol) využít protokol HTTP. Mnoho webových služeb však umí využívat i jiné síťové protokoly z různých vrstev síťového zásobníku (např. SMTP nebo TCP).

V případě existence popisného schématu pro danou technologii webové služby je výhodné mít možnost shraňovat na jednom místě více popisů generovaných popisným schématem pro jednotlivé konkrétní implementace webových služeb dané technologie. To umožní poskytovatelům služeb na toto místo popisy své služby ukládat a klienti se následně mohou na toto místo dotazovat a hledat vhodné služby, které by naplnily jejich potřeby. Pravidla pro ukládání těchto popisů, jejich uchovávání, dotazování se klientů (poskytování odpovědí na tyto dotazy) a poskytování hledaných popisů klientům definuje tzv. *discovery protokol*. V praxi však u drtivé většiny technologií webových služeb tento protokol neexistuje a u technologií, pro které existuje (např. *UDDI* pro správu *WSDL* popisů protokolu *SOAP*), se příliš neosvědčil (mimo jiné také z toho důvodu, že ne všechny služby je žádoucí poskytovat kompletně veřejně celému světu, a tak vznikaly spíše lokální firemní registry na úkor toho globálního a pro lokální firemní prostředí zase naopak nevznikala kvůli velikosti domény taková potřeba pro vyhledávání webových služeb).

Nyní si po zadefinování všech komponent potřebných pro komunikaci mezi klientem a službou obecný princip této komunikace popíšeme. Toto chování společné všem technologiím webových službeb zachycuje Obr. 2.4.



Obrázek 2.4: Workflow webových služeb

V první řadě se musí klient dozvědět, kde se služba, kterou by rád využil, nachází. Toho lze docílit dvojím způsobem:

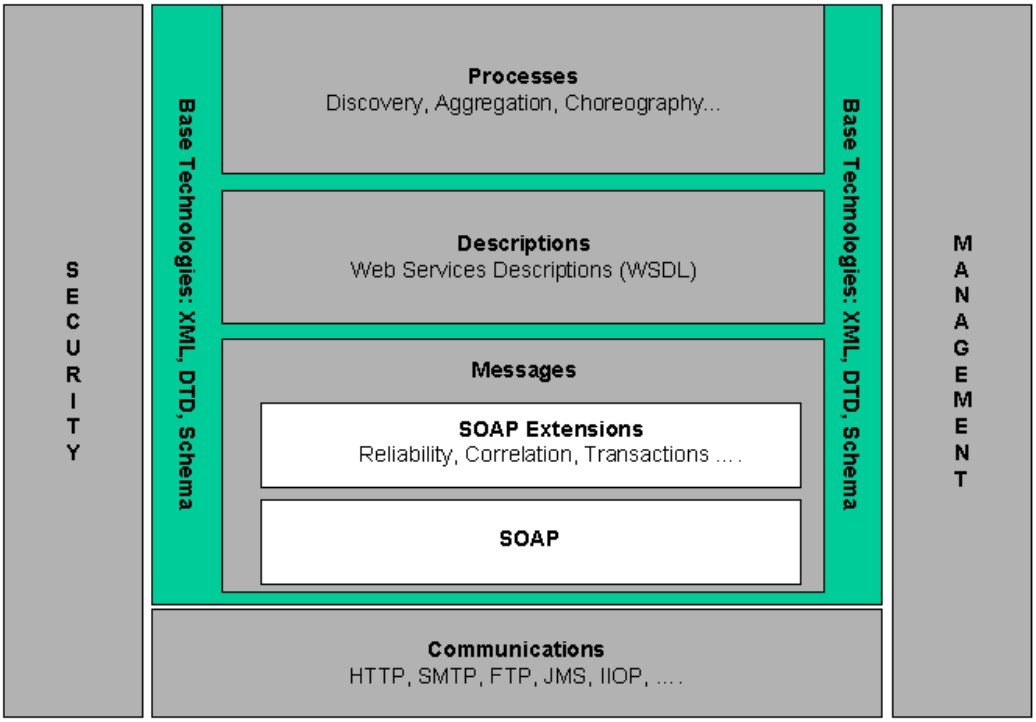
- Klientovi je manuálně dodán popis webové služby dle popisného schématu (v případě neexistence popisného schématu pro danou technologii webové služby je klientovi manuálně dodána rovnou adresa poskytované služby).
- Poskytovatel služby za použití discovery protokolu uloží popis poskytované webové služby dle popisného schématu do veřejného registru (viz krok číslo 1 na Obr. 2.4) a klient se poté za použití discovery protokolu na tento registr dotáže s požadavkem o tuto službu a popis této webové služby dle popisného protokolu je mu registrem vrácen (viz krok číslo 2 na Obr. 2.4).

V případě, kdy klient již ví, kde se služba nachází, k ní může vyslat požadavek (viz krok číslo 3 na Obr. 2.4). Ten je za pomoci obálkového protokolu přes síť doručen poskytovateli služby, který ho zpracuje (viz krok číslo 4 na Obr. 2.4) a odpověď pošle analogickým způsobem zpět klientovi (viz krok číslo 5 na Obr. 2.4).

Nad rámec technologie webové služby a k ní spjatých protokolů lze ještě definovat pojem *choreografie webové služby*, která popisuje nějaká konkrétní

pravidla interakce jednoho nebo více klientů s jednou nebe vícero službami (na Obr. 2.4 je princip choreografie vyznačen modře).

Některé zdroje [8] zavádí pojem *Web Services Architecture Stack* (tzn. *Architektonický zásobník webových služeb*) (viz Obr. 2.5). U tohoto zásobníku ale opět (podobně jako u definic webových služeb) dochází k silné vazbě na konkrétní technologie webových služeb a postrádá tak obecnost potřebnou pro použití u všech technologií webových služeb.



Obrázek 2.5: Architektonický zásobník webových služeb podle W3C (zdroj: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest)

Například architektonický zásobník webových služeb definovaný konsorciem W3C z Obr. 2.5 předpokládá u protokolu webové služby komunikaci pomocí zasílání zpráv (viz sekce 2.3.2) a jako samotný protokol webové služby předpokládá *SOAP*. Entita *Descriptions* (kterou W3C předpokládá jako *WSDL*) odpovídá v terminologii této práce popisnému schématu, entita *Communications* odpovídá obálkovému protokolu a entita *Processes* zahrnuje

[8] http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest

vyšší funkcionalitu v podobě discovery protokolu a choreografie.

Protokoly webových služeb lze rozdělit na *protokoly syntaktické* a *protokoly typové* [pro(2012)]. Protokol označíme jako typový, pokud jím přenášená data podléhají rozdělení do samotným protokolem předem definované množiny datových typů. Naproti tomu protokol syntaktický přenáší data podle nějaké protokolem definované gramatiky a typovým systém se na této úrovni vůbec neřeší. Popisné schéma typového protokolu vytváří mezi komunikujícími stroji jednotné API. U syntaktických protokolů žádné API v rámci popisného schématu neexistuje a oba koncové stroje mohou data zpracovávat různým způsobem (musí pouze dodržet pravidla gramatiky stanovené popisným schématem).

Například *JSON-RPC* je typový protokol, protože v požadavcích a odpovědích přenáší data ve formátu JSON, který svou gramatikou přímo vymezuje hierarchii předem známých datových typů. Pokud se v požadavku objeví například:

```
{
  "param1": "value",
  "param2": 42,
  "param3": null
}
```

cílová strana bez cizí pomoci dokáže rozpoznat, že přijala objekt s třemi atributy: atributem `param1` obsahujícím řetězec `value`, parametrem `param2` s celočíselnou hodnotou `42` a parametrem `param3` bez přiřazené hodnoty.

Naproti tomu *SOAP* je typickým zástupcem syntaktického protokolu, jelikož v těle jeho požadavků a odpovědí se data přenáší ve formátu XML, jehož gramatika není dostatečně restriktivní na to, aby bylo z dat okamžitě jasné, o jaké datové typy jde. Pokud se v požadavku objeví například:

```
<objekt>
  <param1>value</param1>
  <param2>42</param2>
  <param3 xsi:nil=”true” />
</objekt>
```

cílová strana nedokáže bez popisu poskytnutého popisným schématem rozpoznat, o jaké datové typy se jedná, ani jak je souhrnně reprezentovat.

Typové protokoly lze dále rozdělit na *staticky typové* a *dynamicky typové*.

Data jsou staticky typovaným protokolem přenášena bez jakékoliv ontologie - jejich popis je vytvořen podle popisného schématu a tento popis musí mít oba komunikující stroje k dispozici, aby mohly sestavovat a posílat popř. přijímat a parsovat data. Dynamicky typový protokol přenáší popis dat přímo s daty samotnými, takže pokud se struktura dat náhle změní, změní se i jejich popis, kdežto v případě staticky typového protokolu je potřeba rozdistribuovat novou verzi popisu dat.

Příkladem staticky typovaného protokolu je např. *CORBA*, která přenáší data v binární formě podle formátu specifikovaného v příslušném IDL (Interface Definition Language). Pokud se na jedné straně změní IDL a začnou se vysílat data v jiném formátu, je třeba změnit IDL i na straně druhé, jinak bude docházet k chybám. Zástupcem dynamicky typovaného protokolu je například již zmiňovaný *JSON-RPC*. Pokud se v požadavku posílaným tímto protokolem změní atribut `"param2": 42` na `"param2": "42"` nebo `"param2": 42.0`, protistrana se ihned dozví, že začala v atributu `param2` přijímat místo celého čísla řetězec resp. reálné číslo.

Z takto uvedených vlastností vyplývá, že staticky typový protokol je výhodný pro minimalizaci přenášených dat a tam, kde se struktura přenášených dat často nemění. Dynamicky typové protokoly mají výhodu ve snadnější verzovatelnosti a snažší adaptaci na změny.

2.5.1 Soběstačnost zdrojově orientované komunikace

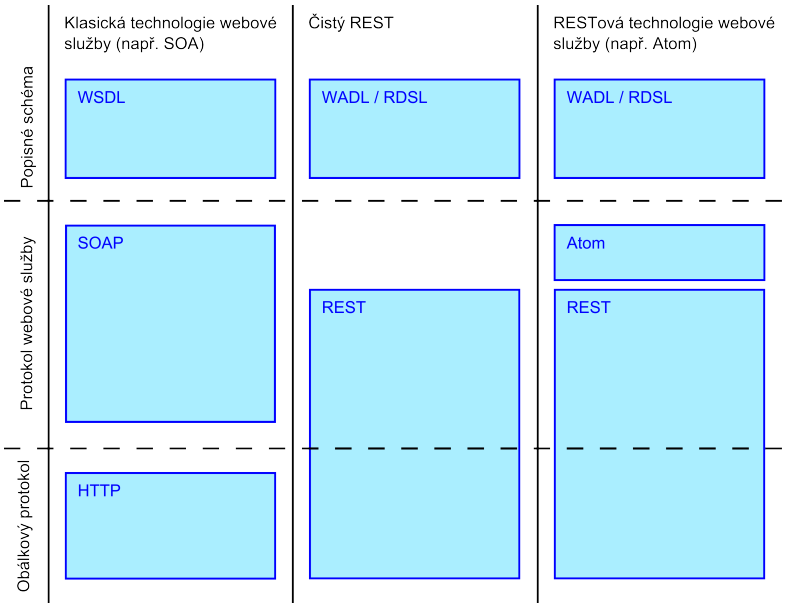
U klasických technologií webových služeb, podléhajících ostatním komunikačním vzorům (vzdálené volání procedury a zasílání zpráv), lze vždy jednoduše rozlišit mezi protokoly použitými pro transport dat (obálkový protokol), vytváření a zpracování dat (protokol webové služby) a ontologii dat (popisné schéma) - viz první sloupec na Obr. 2.6.

V případě zdrojově orientované komunikace ale pod jednu entitu (HTTP protokol) spadá nejen role transportu dat, ale díky aktivnímu využití HTTP metod, HTTP stavových kódů a vlastností HTTP hlaviček je stejný protokol využíván i pro vytváření a zpracování dat. Dochází tak k tomu, že daný protokol plní funkci jak obálkového protokolu, tak protokolu webové služby - viz druhý sloupec na Obr. 2.6. V drtivé většině případů navíc komunikace splňuje požadavky na REST komunikaci definované v sekci 2.3.3, a hovoříme tak rovnou o *RESTové webové službě*.

Na takovouto zdrojově orientovanou komunikaci však lze klást další dodatečná pravidla, která zadefinují konkrétní podobu struktur použitých zdrojů nebo jejich umístění. V tom případě je oblast protokolu webové služby vymezena navíc ještě nadřazeným protokolem, který toto upravuje - viz třetí sloupec na Obr. 2.6. Když se pak hovoří o technologii webové služby jako o celku, používá se název tohoto nadřazeného protokolu (např. Atom).

Tyto RESTové webové služby musíme být pro účely této práce také schopni popsat. Vznikly popisná schémata, které dokážou čistě RESTové webové služby popsat. V případě neexistence nebo nevhodnosti jiného specifického popisného schématu lze tyto popisná schémata použít i pro popis RESTových webových služeb s nadřazenou technologií webové služby (např. Atom žádné své vlastní specifické popisné schéma nemá, ale teoreticky ho mít může).

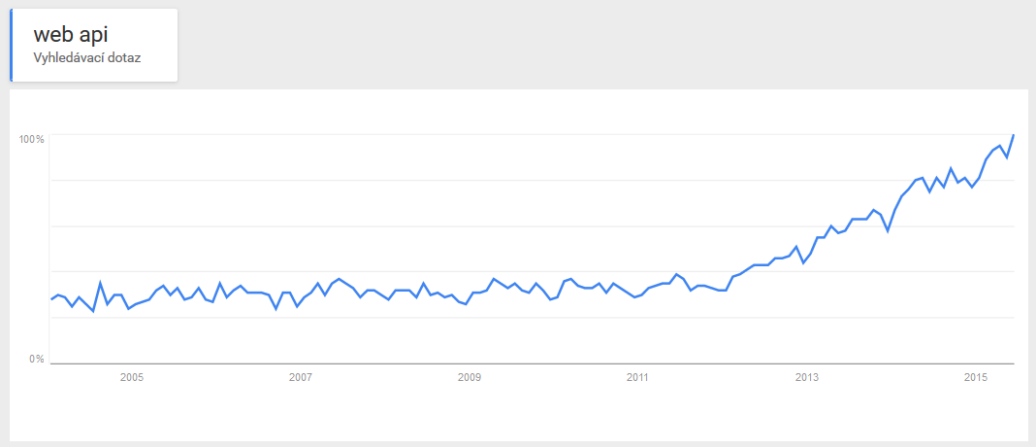
Mezi nejznámější popisná schémata pro RESTové webové služby patří *WADL* (*Web Application Description Language*) a *RSDL* (*RESTful Service Description Language*). RESTové webové služby popsané těmito schématy budou tedy v následující kapitole probrány jako jeden ze zástupců známých technologií webových služeb.



Obrázek 2.6: Porovnání klasických a RESTových technologií webové služby

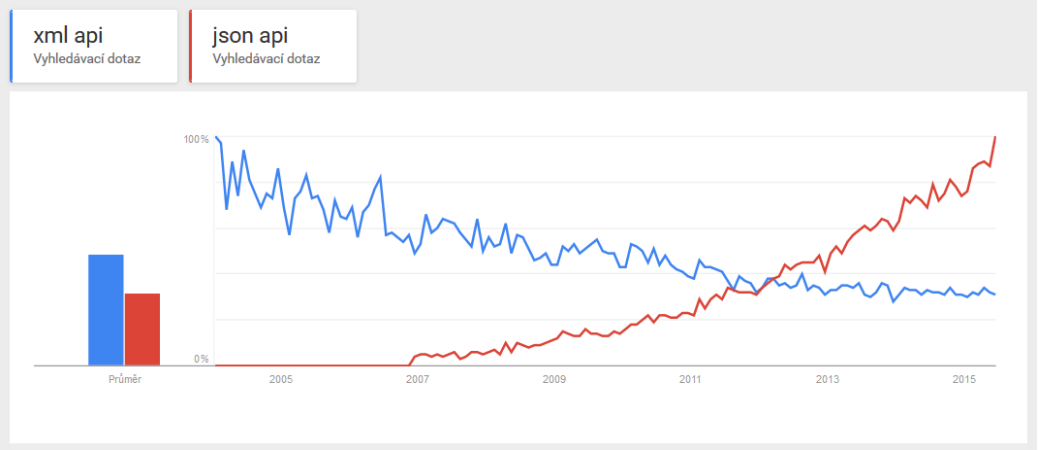
2.6 Současné trendy mezi webovými službami

Oblast webových služeb je v informatice poměrně nový koncept, který se, stejně jako internet, překotně vyvíjí. Neustále se objevují nové trendy a zavedené technologie a způsoby přestávají rychle z různých důvodů vyhovovat. Situace je o to horší, že často se pro nějakou obecnou technologii nebo definici pojmu snaží standard zavést najednou vícero organizací nebo společností a nepřehlednost je tím pouze umocněna. Jedinou konstantou v této oblasti je fakt, že webové služby (a stále různorodější operace nad nimi) jsou rok od roku stále populárnější (viz Obr. 2.7).



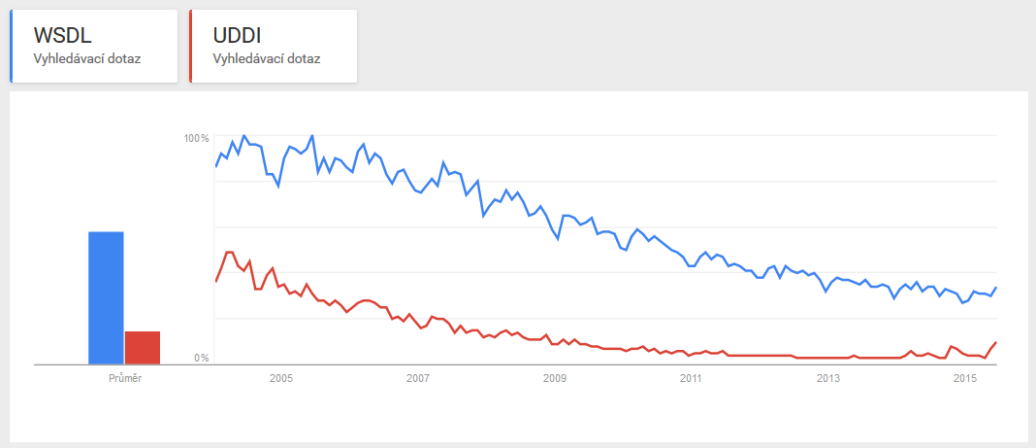
Obrázek 2.7: Zájem o webová API v čase (zdroj: http://www.google.com/trends)

Nejmarkantnějším trendem posledních let je postupný ústup od přenášení dat ve formátu XML ve prospěch formátu JSON. Stále více technologií webových služeb (ale například i databázových a jiných technologií) upřednostňuje transport / reprezentaci dat ve formátu JSON hlavně kvůli tomu, že jsou data kompaktnější, snadněji se čtou a rychleji se zpracovávají [Nurseitov et al.(2009)]. Graf na Obr. 2.8 ukazuje jaký byl o tyto formáty dat zájem v čase.



Obrázek 2.8: Porovnání zájmu o API ve formátech XML a JSON v čase (zdroj: http://www.google.com/trends)

Z definice termínu *webová služba* vytvořené W3C konsorciem (viz sekce 2.4) lze pozorovat, že v době, kdy definice vznikala (rok 2004), panoval názor, že v oblasti webových služeb bude jako protokol výhradně používán SOAP s daty přenášenými ve formátu XML, jako popisné schéma bude používán výhradně WSDL a objevování a kategorizace webových služeb bude probíhat za pomoci discovery protokolu UDDI. Nicméně v tak rapidně měnícím se prostředí, jakým jsou webové služby, žádná sada technologií nebude vyhovovat navždy a jak je vidět z Obr. 2.9, zájem o technologie WSDL a UDDI (o kterých se myslelo, že vytlačí technologie jim konkurující) neustále klesá ve prospěch jiných, nově vytvářených technologií, které odráží současné potřeby v této oblasti.



Obrázek 2.9: Porovnání zájmu o popisný protokol WSDL a discovery protokol UDDI v čase (zdroj: http://www.google.com/trends)

Novodobým trendem v oblasti webových služeb je také vznik mnoha projektů[9], které shraňují informace o veřejně přístupných API a přes webové stránky umožňují API podle různých kritérií vyhledávat nebo přidat vlastní. Jde vlastně o jakousi náhradu discovery protokolů, které se příliš neujaly.

Z informací, které tyto typy projektů o veřejně dostupných webových službách nejčastěji uchovávají, jde především o umístění služby, její kategorie (například `Sport`, `Ekonomika`, apod.), popis a informace o poskytovateli služby. Všechny tyto informace jsou ale vyplněny právě poskytovatelem služby při procesu přidávání vlastní veřejné webové služby. Nedochází tedy k žádnému vystavení a zpracování popisu služby podle popisného schématu, ze kterého by se daly zjistit například detailní informace o poskytovaných metodách.

[9]např. https://www.publicapis.com/ , http://www.programmableweb.com/ a mnoho dalších

3 Technologie webových služeb

Jedním z hlavních cílů této práce je analýza známých technologií webových služeb, z nichž bude poté podle zadefinovaných kritérií vybrána reprezentativní množina. Po získání této množiny bude následně případně upraven datový model úložiště CRCE tak, aby dokázal uchovat popisy technologií webových služeb z této množiny a implementován modul, který bude umět tyto popisy z konkrétních webových služeb vytvářet a do úložiště ukládat.

V této sekci tedy popíšeme a zkategorizujeme nejznámější technologie webových služeb na základě pojmů definovaných v této práci, posoudíme jejich vhodnost zařazení do množiny a výsledek této selekce zhodnotíme.

3.1 Kritéria hodnocení technologií

Každá technologie webové služby je v následujících sekcích teoreticky rozebrána a je posouzena její vhodnost zařazení do vybrané množiny technologií webových služeb. Účelem práce je implementovat uchovávání popisů konkrétních webových služeb spadajících pod technologie z této množiny, a proto budou jako hlavní kritéria pro zařazení do množiny definována především používanost dané technologie webové služby a kvalita jejího popisného schématu (pokud daná technologie webové služby žádný popisné schéma nemá, nelze ji pro účely této práce použít, jelikož nelze získat popis jednotlivých konkrétních webových služeb spadajících pod tuto technologii).

Pro každou následující technologii webové služby je nejprve uvedena přehledová tabulka s následujícími údaji:

- **Typ protokolu** - Typ protokolu webové služby podle definice v sekci 2.5 (syntaktický / dynamicky typový / staticky typový).
- **Kom. vzor** - Komunikační vzor používaný webovou službou podle definice v sekci 2.3.
- **Předchůdce** - Protokoly, které z vývojového nebo ideového hlediska předcházely tomuto protokolu webové služby.

- **Následovník** - Protokoly, které z vývojového nebo ideového hlediska následovaly po tomto protokolu webové služby.
- **Popisné schéma** - Popisné schéma této webové služby podle definice v sekci 2.5.
- **Formát dat** - Formát, ve kterém jsou data přenášena mezi poskytovatelem a konzumentem webové služby.
- **Formát PS** - Formát dat, ve kterém je realizován popis webové služby podle příslušného popisného schématu.
- **Správce** - Firma nebo organizace, která v současné době spravuje tento protokol webové služby.

Po přehledové tabulce následuje stručná historie protokolu, jeho základní popis a dále je prodiskutována míra využitelnosti popisného schématu pro potencionální zahrnutí tohoto protokolu webové služby do vybrané množiny technologií webových služeb. Za závěr je rozhodnuto o zařazení nebo nezařazení daného protokolu do vybrané množiny technologií webových služeb.

3.2 Atom

Typ protokolu	syntaktický	**Kom. vzor**	REST
Předchůdce	RSS	**Následovník**	
Popisné schéma	není		
Formát dat	XML	**Formát PS**	
Správce	Internet Engineering Task Force (RFC 4287)		

Tabulka 3.1: Souhrn vlastností protokolu Atom

Historie

Počátky protokolu sahají do roku 2003, kdy se skupina v oboru významných lidí rozhodla podpořit tvorbu nového protokolu, který by měl širší možnosti než RSS, a který by odstranil některé jeho nejednoznačnosti.[1] Verze Atom 0.3

[1] http://web.archive.org/web/20071211104213/http://www.intertwingly.net/wiki/pie/RoadMap

vydaná v prosinci 2003 už byla adoptována mnoha syndikačními nástroji a hlavně ji implementovala společnost Google v mnoha svých službách (například Blogger, Google News a Gmail). V roce 2004 přešel projekt pod správu IETF[2] a o rok později se Atom ve verzi 1.0 stal standardem v RFC 4287.

Popis

Atom vznikl jako snaha nahradit technologii RSS a stejně jako předchůdce slouží k syndikaci webového obsahu. Protokol samotný je definován v RFC 4287[3]. Pravidla pro komunikaci nad zdroji ve formátu protokolu Atom jsou definovaná v RFC 5023[4]. Přesto, že měl protokol Atom plně nahradit technologii RSS, spolu dnes obě technologie soupeří hlavně kvůli relativní zažitosti RSS.

RFC 4287 v sekci 4.1 zavádí dva hlavní zdroje: *Atom feed* představující informační kanál, který může obsahovat vícero záznamů *Atom entry*, představujících publikované příspěvky daného informačního kanálu.

Využitelnost popisného schématu

Bohužel Atom nemá žádnou standardizovanou podobu pro to, jak pojmenovávat URI adresy, nad kterými poskytuje zdroje. V RFC 5023 v sekci 4.1 přímo stojí:

While the Atom Protocol specifies the formats of the representations that are exchanged and the actions that can be performed on the IRIs embedded in those representations, ***it does not constrain the form of the URIs that are used****.*

tzn. *Přestože protokol Atom specifikuje formáty reprezentací, které jsou vyměňovány a akce, které jsou vykonány na daných IRI navázaných na tyto reprezentace,* ***nijak neomezuje formu URIs, která jsou použita****.*

Některé zdroje sice doporučují používat pro publikované příspěvky URI ve tvaru `adresa-atom-feedu/identifikator-prispevku`, ale toto neoficialní

[2]https://tools.ietf.org/wg/atompub/
[3]http://tools.ietf.org/html/rfc4287
[4]http://tools.ietf.org/html/rfc5023

doporučení často nedopovídá realitě. Navíc bohužel protokol nevymezuje, jak lze při znalosti URI Atom feedu získat URI jednotlivých Atom entries.

Verdikt

Tento protokol nebude zařazen do vybrané množiny technologií webových služeb kvůli nízké kvalitě svého popisného schématu.

3.3 XML-RPC

Typ protokolu	dynamicky typový	**Kom. vzor**	RPC
Předchůdce		**Následovník**	SOAP
Popisné schéma	není (popř. XRDL)		
Formát dat	XML	**Formát PS**	XML (XRDL)
Správce	webMethods		

Tabulka 3.2: Souhrn vlastností protokolu XML-RPC

Historie

Protokol vznikl v roce 1998 pod záštitou firem UserLand Software a Microsoft. Se získáním nové funkcionality se standard postupně přeměnil v protokol SOAP s mírně odlišným chováním.[5] Správa původního protokolu, používajícího komunikační vzor RPC a formát dat XML, přešla v roce 2006 pod firmu webMethods [Merrick et al.(2006)].

Popis

Tento protokol definuje strukturu požadavků a odpovědí ve formátu XML.[6] Ty jsou mezi klientem a službou posílány podle komunikačního vzoru vzdálené volání procedury. Jako obálkový protokol je používán HTTP protokol

[5] http://www.xml.com/pub/a/ws/2001/04/04/soap.html
[6] http://xmlrpc.scripting.com/spec

(je na něj kladen požadavek na správné vyplnění těchto vlastností v HTTP hlavičce: User-Agent, Host, Content-Length a Content-Type (nastavený na `text/xml`)).

Požadavek musí být celý obsažen v tagu `<methodCall>`, který musí obsahovat tag `<methodName>` (ten obsahuje název vyvolávané služby / metody) a pokud má vyvolávaná služba / metoda nějaké vstupní parametry, musí obsahovat i tag `<params>` s těmito parametry. Odpověď je celá vrácena v tagu `<methodResponse>`, který obsahuje buď tag `<params>` s vrácenou hodnotou, nebo, v případě chyby, tag `<fault>` s kódem a popisem chyby. Podporuje běžné primitivní i složené datové typy.

Využitelnost popisného schématu

Tento protokol nemá žádné oficiální popisné schéma, ale právě kvůli jeho absenci vzniklo neoficiální popisné schéma XRDL[7], které přineslo alespoň základní možnost popisu služeb fungujících na technologii XML-RPC. I přesto, že je XRDL validním popisným schématem pro XML-RPC (viz [Troanca – BoianRemco(2013)]) nebyla tato technologie doposud standardizována ani šiřoce podpořena. Popisy jsou vytvářeny podle XSD definice[8].

Verdikt

Tento protokol nebude zařazen do vybrané množiny technologií webových služeb kvůli neexistenci oficiálního popisného schématu (a nízké míře rozšíření neoficiálního popisného schématu XRDL) a také kvůli faktu, že i přes udržovanou existenci této technologie byla v některých řešeních kvůli relativní jednoduchosti technologie XML-RPC z velké části vytlačena technologií SOAP.

[7] https://code.google.com/p/xrdl/

[8] https://code.google.com/p/xrdl/source/browse/documentation/xrdl.xsd

3.4 SOAP

Typ protokolu	syntaktický	**Kom. vzor**	Messaging
Předchůdce	XML-RPC	**Následovník**	
Popisné schéma	WSDL		
Formát dat	XML	**Formát PS**	XML
Správce	World Wide Web Consortium		

Tabulka 3.3: Souhrn vlastností protokolu SOAP

Historie

Počátky protokolu SOAP lze pozorovat v roce 1998, kdy část jeho prvotní specifikace vyšla jako protokol XML-RPC. SOAP byl tudíž v té době zamýšlen čistě jako protokol podléhající komunikačnímu vzoru vzdáleného volání procedur (RPC). Obrat nastal v roce 1999, kdy byl protokol přepracován tak, že podléhal více komunikačnímu vzoru zasílání zpráv (messaging).[Graham(2002)] Ten samý rok vyšel s pár změnami protokol SOAP verze 1.0.

V roce 2000 byl SOAP verze 1.1 předložen jako poznámka World Wide Web konsorciu[9], na jejímž základě byla o pár měsíců později založena pracovní skupina pro správu protokolů na bázi XML. Tato skupina začala pracovat na verzi SOAP 1.2 a v roce 2001 vydala jeho první pracovní návrh [Graham(2002)]. V roce 2007 tato skupina vydala zatím nejnovější specifikaci SOAP protokolu již jako oficiální W3C doporučení[10].

Popis

Jde o jeden z nejrozšířenějších a nejpoužívanějších protokolů webových služeb. Na základě tohoto protokolu jsou mezi klientem a poskytovatelem služby přenášeny zprávy ve formátu XML s jasně definovanou strukturou. Protokol je spravován World Wide Web konsorciem, které poskytuje i jeho kompletní specifikaci.[11]

[9] http://www.w3.org/TR/2000/NOTE-SOAP-20000508/
[10] http://www.w3.org/TR/soap12/
[11] http://www.w3.org/TR/soap/

Jako obálkový protokol umí využít vícero síťových protokolů (SMTP, TCP, UDP, ...), ale nejčastěji je pro transport dat s výhodou využíván protokol HTTP. Spolu s protokolem SOAP se vyvíjelo i jeho popisné schéma *WSDL (Web Services Description Language)* a discovery protokol *UDDI (Universal Description Discovery and Integration)*[12]. Zatímco UDDI (a dá se říci, že globální discovery protokoly obecně) se nedočkal širokého rozšíření, WSDL se kvůli potřebě popsat a provozovat komunikaci mezi klientem a poskytovalem služby od roku 2000 postupně vyvíjelo a dnes se lze již setkat s WSDL ve verzi 1.1 a 1.2 (přejmenované na verzi 2.0).

Využitelnost popisného schématu

Popisné schéma WSDL také definuje data ve formátu XML a také je spravováno World Wide Web konsorciem, které poskytuje i jeho kompletní specifikaci.[13]. V závislosti na použité verzi je WSDL dokument různě strukturovaný, ale vždy obsahuje informace o použitých složených datových typech, formátech posílaných zpráv a URL adres, na které lze tyto zprávy zasílat.

Verdikt

Tento protokol bude zařazen do vybrané množiny technologií webových služeb kvůli dostatečné kvalitě svého popisného schématu a kvůli faktu, že jeho přítomnost v množině je vyžadována zadáním této práce.

[12] http://www.oasis-open.org/committees/uddi-spec
[13] http://www.w3.org/TR/wsdl

3.5 JSON-RPC

Typ protokolu	dynamicky typový	**Kom. vzor**	RPC
Předchůdce		**Následovník**	JSON-WSP
Popisné schéma	není (popř. Barrister RPC)		
Formát dat	JSON	**Formát PS**	IDL (Barrister RPC)
Správce	JSON-RPC Working Group		

Tabulka 3.4: Souhrn vlastností protokolu JSON-RPC

Historie

V roce 2004 vznikla iniciativa definovat protokol pro vzdálené volání procedur, který by jako formát dat používal JSON[14]. Od té doby postupně vycházely nové verze protokolu až po nejnovější verzi 2.0 vydanou v roce 2010 (a revidovanou v roce 2013).

Popis

Specifikace ve verzi 2.0[15] definuje tři základní objekty: požadavky posílané od klienta směrem k poskytovateli služby a odpovědi, popř. chybové odpovědi posílané opačným směrem. Pokud klient nezadefinuje v požadavku jeho identifikátor (pole `id`), jedná se tzv. notifikaci a poskytovatel služby nebude po vyvolání příslušné metody klientu posílat žádnou odpověď.

Využitelnost popisného schématu

Pro JSON-RPC neexistuje žádné oficiální popisné schéma. Jedinou komunitní snahou o nápravu je *Barrister RPC*[16], který popisuje JSON-RPC pomocí formátu IDL (*IDL - Interface Description Language* se zde myslí v původním

[14]http://json-rpc.org/wiki/about
[15]http://www.jsonrpc.org/specification
[16]http://barrister.bitmechanic.com/

konkrétním významu [Lamb(1983)]; později se tímto termínem mimo jiné začaly označovat obecně všechna popisná schémata).

Kvůli neexistenci oficiálního popisného schématu a nejednotnému formátu dat v případě Barrister RPC (JSON vs. IDL) později vznikl protokol webové služby JSON-WSP jakožto nástupce protokolu JSON-RPC, který přímo jako svoji součást obsahoval i popisné schéma ve stejném formátu dat.

Verdikt

Tento protokol nebude zařazen do vybrané množiny technologií webových služeb kvůli neexistenci oficiálního popisného schématu (a nízké míře rozšíření a nevhodnosti neoficiálního popisného schématu Barrister RPC) a také kvůli faktu, že je v současné době tento protokol nahrazován protokolem JSON-WSP.

3.6 JSON-WSP

Typ protokolu	dynamicky typový	**Kom. vzor**	RPC
Předchůdce	JSON-RPC	**Následovník**	
Popisné schéma	JSON-WSP		
Formát dat	JSON	**Formát PS**	JSON
Správce			

Tabulka 3.5: Souhrn vlastností protokolu JSON-WSP

Historie

Protokol JSON-WSP (Java Script Object Notation Web-Service Protocol) začal vznikat v roce 2011 z protokolu JSON-RPC kvůli potřebě kvalitního popisného schématu. Díky tomu by měl být protokol JSON-RPC postupně tímto protokolem nahrazován. JSON-WSP zatím není zaštítěn žádnou organizací nebo společností, ale v současné době prochází standardizačním procesem organizace Internet Engineering Task Force[Hellbruck et al.(2013)] a

k protokolu již existují implementace pro použití v různých programovacích jazycích.

Popis

Podobně jako protokol JSON-RPC tento protokol definuje tři základní objekty: požadavky posílané od klienta směrem k poskytovateli služby a odpovědi, popř. chybové odpovědi posílané opačným směrem. Ale oproti svému předchůdci navíc definuje ještě jeden druh objektu navíc: popis poskytované webové služby podle specifikace popisného schématu, ze kterého se klient dozví, jakým způsobem může s poskytovatelem služby komunikovat.

Kompletní specifikace tohoto protokolu (včetně specifikace popisného schématu) je v současné době udržována na Wikipedii[17]. Oproti JSON-RPC navíc tento protokol zavádí datový typ *attachment*, díky kterému lze pomocí transportního protokolu (HTTP) přenášet spolu s požadavkem a odpovědí i surová data (například soubor, který je potřeba nahrát).

Využitelnost popisného schématu

Specifikace protokolu JSON-WSP definuje zároveň jak protokol webové služby, tak i popisné schéma. Webová služba JSON-WSP je popsána jako JSON objekt s pevně danou strukturou, která vždy obsahuje základní informace o popisované webové službě (hlavně URL, na kterém lze se službou komunikovat), použitých složených datových typech, metodách a jejich parametrech.

Verdikt

Tento protokol bude zařazen do vybrané množiny technologií webových služeb kvůli dostatečné kvalitě popisného schématu a také kvůli faktu, že vychází z velmi používaného protokolu JSON-RPC, jemuž popisné schéma chybí.

[17] https://en.wikipedia.org/wiki/JSON-WSP

3.7 BPEL

Typ protokolu	syntaktický	**Kom. vzor**	Messaging
Předchůdce	WSFL, Xlang	**Následovník**	
Popisné schéma			
Formát dat	XML	**Formát PS**	
Správce	OASIS		

Tabulka 3.6: Souhrn vlastností protokolu BPEL

Historie

V roce 2001 vznikly nezávisle na sobě dvě podobné technologie pro orchestraci webových služeb. Jednu vytvořila firma IBM a nazvala ji *WSFL*[18] (Web Services Flow Language) a druhou vytvořil Microsoft a nazval ji *Xlang*[19]. Tyto firmy (spolu s dalšími) se později rozhodly tyto technologie zkombinovat do podoby, kterou nazvaly právě *BPEL* (Business Process Execution Language) a v roce 2003 předaly BPEL ve verzi 1.1 ke standardizaci orzanizaci OASIS[20]. OASIS vydala o rok později specifikaci nazvanou WS-BPEL 2.0.

Hlavním konkurentem technologie BPEL byla dlouho podobná technologie *BPML* (Business Process Modeling Language) od organizace BPMI (Business Process Management Initiative), která mimo jiné vyvinula i grafickou obdobu těchto technologií nazvanou *BPMN* (Business Process Model and Notation). Ale když byla v roce 2005 BPMI zaštítěna organizací OMG[21] (Object Management Group) od BPML bylo ustoupeno ve prospěch BPEL, který se tak stal de facto standardem pro orchestraci webových služeb.

Popis

Technologie, jejíž plný název zní WS-BPEL (Web Services Business Process Execution Language) 2.0, má specifikaci[22] udržovanou organizací OASIS.

[18] http://xml.coverpages.org/wsfl.html
[19] http://xml.coverpages.org/XLANG-C-200106.html
[20] https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
[21] http://www.omg.org/news/releases/pr2005/06-29-05.htm
[22] http://docs.oasis-open.org/wsbpel/2.0/

BPEL není klasickým protokolem webové služby, jelikož je používán jako nástroj pro orchestraci jiných klasických webových služeb. Definuje tzv. *podnikové procesy* [Desel et al.(2004)], které obsahují aktivní ovládací prvky (sekvence, cykly, větvení na základě podmínky apod.), a jejichž vstupy a výstupy jsou navázany na okolní webové služby.

Využitelnost popisného schématu

Pro BPEL neexistuje žádné popisné schéma.

Verdikt

Tato technologie nebude zařazena do vybrané množiny technologií webových služeb kvůli neexistenci popisného schématu a kvůli přílišným odlišnostem od klasických technologií webových služeb (nástroj pro orchestraci).

3.8 CORBA

Typ protokolu	staticky typový	**Kom. vzor**	RPC
Předchůdce		**Následovník**	
Popisné schéma	OMG IDL		
Formát dat	binární	**Formát PS**	binární
Správce	Object Management Group		

Tabulka 3.7: Souhrn vlastností protokolu CORBA

Historie

Technologie CORBA (Common Object Request Broker Architecture) byla představena v roce 1991 spolu se svým popisným schématem IDL (Interface Definition Language). V roce 1996 byla vydána verze 2.0, která přinesla řadu vylepšení. Vývoj postupně pokračoval až do roku 2002, kdy byla vydána verze 3.0.2. Od té doby se žádný další vývoj neuskutečnil[23].

[23] http://www.omg.org/gettingstarted/history_of_corba.htm

Popis

Tento standard je spravován consorciem OMG[24]. Pro komunikaci klienta a poskytovatele služby je používáno jednotné zkompilované rozhraní vytvořené podle specifikace OMG IDL. Dnes se už tento protokol oproti ostatním novějším technologiím webových služeb nepoužívá /[25] i kvůli faktu, že přenášení, pro člověka nečitelných, binárních dat je zbytečně složité.

Využitelnost popisného schématu

Popisy služby jsou vytvořeny v definované textové podobě podle specifikace OMG IDL[26], a poté přeloženy do binární podoby pro cílovou architekturu. Dříve se toto popisné schéma označovalo pouze jako IDL (Interface Definition Language), ale tento termín zobecněl a dnes se tak souhrnně označuje jakékoliv popisné schéma.

Verdikt

Tato technologie nebude zařazena do vybrané množiny technologií webových služeb kvůli faktu, že v dnešní době téměř není využívána.

3.9 Hessian

Typ protokolu	dynamicky typový	**Kom. vzor**	RPC/Messaging
Předchůdce		**Následovník**	
Popisné schéma	Hessian		
Formát dat	binární	**Formát PS**	binární
Správce	Caucho Technology		

Tabulka 3.8: Souhrn vlastností protokolu Hessian

[24]http://www.omg.org/spec/
[25]https://www.google.com/trends/explore#q=CORBA%2C%20Hessian&cmpt=q
[26]http://www.omg.org/orbrev/drafts/3_idlsyn.pdf

Historie

Protokol Hessian byl vydán ve verzi 1.0 v roce 2004[27]. V roce 2007 byla vytvořena specifikace ve verzi 2.0[28].

Popis

Protokol se nikdy, i přes své zjevné výhody oproti protokolu CORBA [pro(2012)], nedočkal na poli (už tak řídce používaných) binárních webových protokolů širšího rozšíření[29].

Má velice podobné vlastnosti jako technologie CORBA, ale oproti ní přidává jasně definovaným způsobem[30] do každého vysílaného požadavku a odpovědi i popis přenášených dat (datové typy a délku). Tudíž není třeba při změně formátu komunikace distribuovat mezi klienta a poskytovatele služby novou verzi IDL jako v případě technologie CORBA.

Využitelnost popisného schématu

Na rozdíl od protokolu CORBA, protokol Hessian nevyžaduje žádné externí IDL, jelikož informace o přenášené struktuře dat přenáší přímo s daty.

Verdikt

Tento protokol nebude zařazen do vybrané množiny technologií webových služeb kvůli neexistenci popisného schématu a kvůli faktu, že v dnešní době téměř není využíván.

[27] http://caucho.com/about
[28] http://hessian.caucho.com/doc/
[29] https://www.google.com/trends/explore#q=CORBA%2C%20Hessian&cmpt=q
[30] http://hessian.caucho.com/doc/hessian-serialization.html

3.10 RESTová webová služba

Typ protokolu	syntaktický	**Kom. vzor**	REST
Předchůdce		**Následovník**	
Popisné schéma	WADL, RSDL		
Formát dat	libovolný	**Formát PS**	XML
Správce	World Wide Web Consortium		

Tabulka 3.9: Souhrn vlastností RESTové webové služby

Historie

Termín REST poprvé zavedl Roy T. Fielding ve své disertační práci [Fielding(2000)]. RESTové webové služby se začaly objevovat počátkem roku 2006 a jejich oblíbenost od té doby jenom sílí. Jejich využívanost již daleko předčila i kdysi velmi rozšířenou technologii webové služby SOAP[31].

Popis

Jelikož REST vychází ze zdrojově orientovaného komunikačního vzoru, mluvíme spíše o softwarovém architektonickém vzoru, než o technologii webové služby. Podle pravidel tohoto vzoru však lze vytvořit provozuschopnou webovou službu. Tato služba může podléhat dodatečným restrikcím v podobě nějakého protokolu využívajícho REST jako základ (např. protokol Atom), ale může fungovat i zcela samostatně, což je případ probíraný v této sekci. Detailněji se tímto tématem zabývá sekce 2.5.1.

Využitelnost popisného schématu

S masivním rozšířením RESTových webových služeb vznikla přirozeně snaha tyto služby nějak popsat. Dnes mezi nejpoužívanější popisná schémata pro RESTové webové služby patří *WADL* (*Web Application Description Language*) a *RSDL* (*RESTful Service Description Language*). WADL specifikace je spravována W3C konsorciem[32]. RSDL je specifikováno v [Robie et al.(2013)].

[31] https://www.google.com/trends/explore#q=rest%20api%2C%20soap%20api&cmpt=q
[32] http://www.w3.org/Submission/wadl/

RSDL se však nikdy nedostalo širšího použití a dnes nelze najít jedinou webovou službu, kterou by RSDL popisovalo. Oproti tomu WADL se hned v době kdy vznikaly první RESTové webové služby velmi rozšířil a je dodnes používán[33].

Verdikt

Tento protokol bude zařazen do vybrané množiny technologií webových služeb kvůli jeho používanosti a kvůli dostatečné kvalitě popisného schématu (WADL).

3.11 Shrnutí

Do množiny vybraných technologií webových služeb byly vybrány tyto technologie (je uveden vybraný protokol webové služby následovaný vybraným popisným schématem):

1. JSON-WSP / JSON-WSP description object
2. SOAP / WSDL
3. RESTová webová služba / WADL

Popisy vybraných protokolů webových služeb bude třeba získávat z definic vytvořených podle vybraných popisných schémat (tzn. z konkrétních popisných dokumentů většinou dostupných na nějaké URL adrese). Všechna vybraná popisná schémata definují obecné informace o popisované webové službě / službách, včetně tzv. *endpointů* (vstupních bodů do dané webové služby).

Když klient posílá požadavek na endpoint, je třeba, aby měl nějakou strukturu, jinak jej poskytovatel služby odmítne jako neplatný (a většinou pošle zpět chybovou zprávu). V případě vzdáleného volání procedury (RPC) se jedná o počet, pořadí a datové typy přenášených parametrů, v případě zasílání zpráv (messaging) jde o gramatiku, kterou musí posílaná zpráva splňovat

[33]https://www.google.com/trends/explore#q=wadl%2C%20rsdl&cmpt=q

a v případě zdrojově orientované komunikace (resource API) není obecně tvar zasílaných dat definován, ale popisná schémata kombinují oba předchozí způsoby (umožňují definovat jak parametry volání, tak gramatiky požadavků).

Všechna vybraná popisná schémata popisují kromě výše zmíněného právě i tyto požadavky na strukturu posílaných dat mezi klientem a poskytovatelem webové služby (tzn. požadavky a odpovědi), a proto půjdou konkrétní dokumenty těchto vybraných popisných schémat automaticky analyzovat a jejich reprezentace (popis webových služeb z vybrané množiny webových služeb) půjde uložit v jednotném formátu.

Výhodou je také to, že byl do množiny vybraných technologií webových služeb vybrán jeden zástupce každého ze tří elementárních komunikačních vzorů definovaných v sekci 2.3 (JSON-WSP jako zástupce vzdáleného volání procedur, SOAP jako zástupce zasílání zpráv a RESTová webová služba jako zástupce zdrojově orientované komunikace).

4 Reprezentace vybraných technologií v datovém modelu

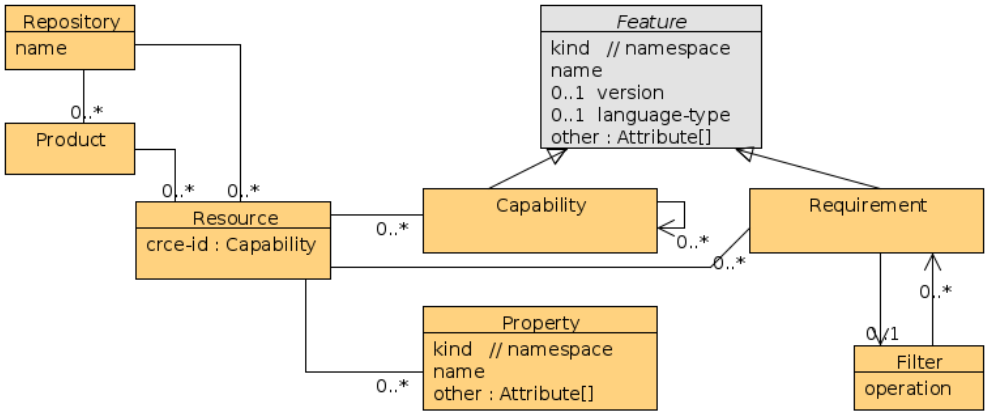
V této kapitole bude popsán datový model, který byl na Katedře informatiky a výpočetní techniky ZČU v Plzni vyvinut pro reprezentaci modulárních softwarových aplikací a tvoří mimo jiné i základ úložiště CRCE. Budou probrány popisná schémata z množiny vybraných technologií webových služeb (viz sekce 3), vyhodnocena případná potřeba změnit datový model pro reprezentaci informací, které poskytují a popsána nově zavedená metadata ukládaná do tohoto datového modelu.

Datový model uchovává metadata o softwarových modulech (komponentách) a vychází z obecného konceptu použitého v technologii OBR (OSGi Bundle Repository), který pro ukládané *resources* (zdroje / artefakty) definuje tzv. *requirements* (požadavky) a *capabilities* (schopnosti) [Nicholson(2014)]. Základem modelu[1] (viz Obr. 4.1) je úložiště (Repository), které uchovává množinu zdrojů (Resource). Zdroj představuje množinu metadat pro nějaký konkrétní softwarový modul (komponentu) a může obsahovat libovolné množství požadavků (Requirement), schopností (Capability) a vlastností (Property). Tyto entity definované v metadatech pro zdroj mají následující význam:

- Requirement - Představuje požadavky komponenty, které musí být splněny pro její bezchybnou funkčnost. Jsou vyjádřeny jako množina schopností komponenty.
- Capability - Představuje schopnosti komponenty, které nabízí vůči svému okolí. Používají se pro vyhodnocování naplnění požadavků jiných komponent.
- Property - Představuje dodatečné vlastnosti komponenty v podobě metadat.

Všechny tři výše zmíněné entity jsou definovány se svým namespace, který je vůči sobě hierarchicky uspořádává.

[1] https://www.assembla.com/spaces/crce/wiki/Metadata_structure



Obrázek 4.1: Struktura datového modelu pro reprezentaci modulárních softwarových aplikací (zdroj: https://www.assembla.com/spaces/crce)

4.1 Informace poskytované vybranými popisnými protokoly

Každý z vybraných popisných schémat poskytuje o typu webových služeb, který popisuje, trochu jiné informace. Nicméně ve všech lze najít společný průnik základních poskytovaných informací vhodných pro jednotnou reprezentaci webové služby v tomto datovém modelu. Tyto vybrané základní poskytované informace jsou:

- Souhrnné informace o webové službě (jméno webové služby, URL ze které je přístupná, její komunikační vzor, typ a verze popisného schématu popisujícího tuto službu, apod.)
- Informace o endpointech webové služby (jméno endpointu, URL ze které je přístupný, vstupní a výstupní parametry)
- Informace o parametrech (jméno parametru, jeho datový typ popř. odkaz na použitou gramatiku, pořadí parametru, apod.)

Vybrané popisné schéma nemusí nutně poskytovat informace o všech jednotlivostech uvedených výše (například nemusí poskytovat informaci o své

verzi, nebo všechny endpointy sdílí společné URL definované v souhrnných informacích o webové službě). V takovém případě zůstane vlastnost v reprezentaci popisu webové služby jednoduše nevyplněna.

4.1.1 JSON-WSP description object

Tento dokument má pevnou strukturu dat (viz sekce 3.6) ve formátu JSON a popisuje právě jednu webovou službu typu JSON-WSP.

V kořenovém JSON objektu je definována verze popisného schématu (v současné době pouze `1.0`), jméno popisované webové služby, množina službou používaných složených datových typů a množina definovaných endpointů (metod).

Složený datový typ se zde skládá ze svého jména a libovolného množství položek, skládajících se ze jména a reference na jiný složený datový typ nebo názvu primitivního datového typu.

Endpoint se skádá ze svého jména, libovolného počtu parametrů a vrácené odpovědi. K endpointu, parametru i odpovědi lze definovat tzv. *dokumentační řetězec* popisující podrobně účel daného prvku. Každý parametr i odpověď musí mít definovaný svůj datový typ (název primitivního datového typu nebo přes referenci odkaz na složený datový typ) a parametr musí navíc obsahovat informaci o tom, jaké je jeho pořadí v rámci endpointu, a zda je volitelný či nikoliv.

Příklad popisného dokumentu JSON-WSP description object

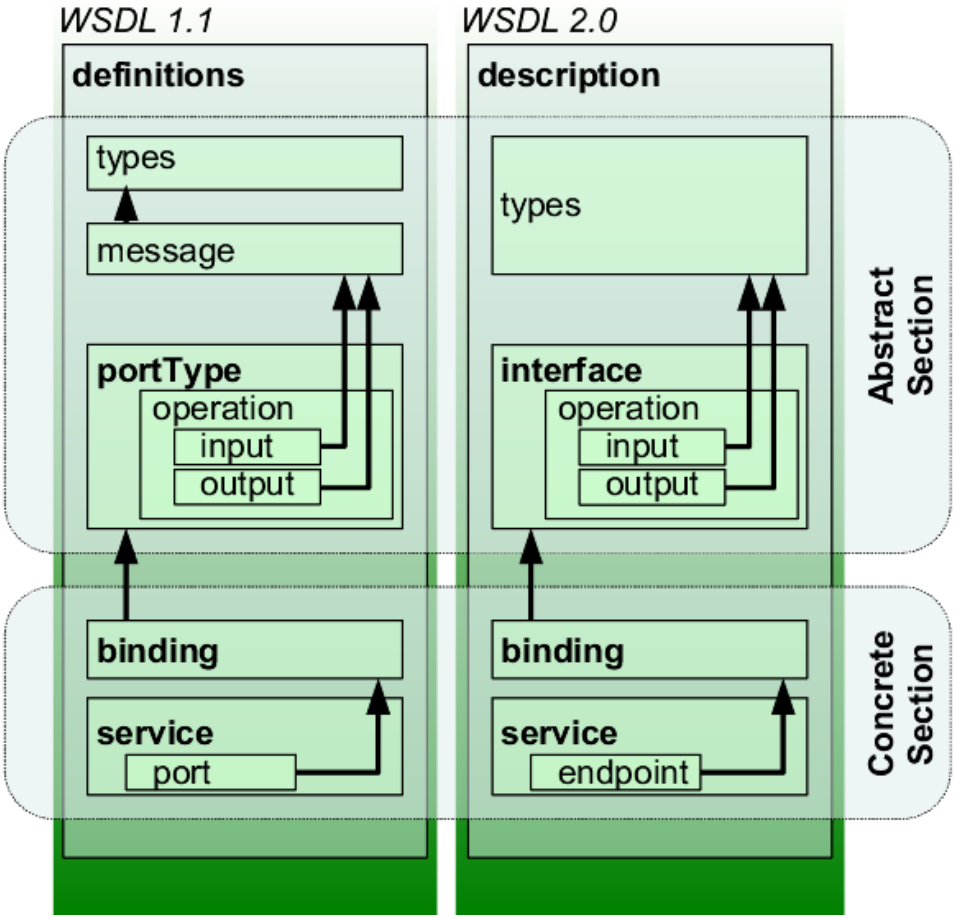
```
{
 "version": "1.0",
 "url": "http://ladonize.org/python-demos/AlbumService/jsonwsp",
 "servicename": "AlbumService",
 "type": "jsonwsp/description",
 "types": {
  "Album": {
   "band": "Band",
   "title": "string",
   "songs": ["string"]
  },
  "Band": {
   "album_titles": ["string"],
```

```
    "name": "string"
   }
  },
  "methods": {
   "listBands": {
    "ret_info": {
     "doc_lines": [],
     "type": ["Band"]
    },
    "doc_lines": ["Fetch a list of albums matching search_frase"],
    "params": {
     "search_frase": {
      "def_order": 1,
      "doc_lines": [],
      "type": "string",
      "optional": true
     }
    }
   },
   "listAlbums": {
    "ret_info": {
     "doc_lines": [],
     "type": ["Album"]
    },
    "doc_lines": ["Fetch a list of albums matching search_frase"],
    "params": {
     "search_frase": {
      "def_order": 1,
      "doc_lines": [],
      "type": "string",
      "optional": true
     }
    }
   }
  }
}
```

4.1.2 WSDL

Dokument WSDL se v současné době používá ve dvou mírně odlišných verzích: verzi `1.1` a `2.0` (dříve označované jako `1.2`). Obě verze mají, kromě dalších věcí, společné to, že se vždy skládají z tzv. *abstraktní části* a *konkrétní části* (viz Obr. 4.2). Jeden WSDL dokument může popisovat libovolný počet webových služeb a je ve formátu XML. Dokument popisuje právě tolik webových služeb, kolik jeho kořenový element obsahuje elementů `service`.

Všechny typy elementů v dokumentu mají definované své jméno a způsobem, který je vyznačen na Obr. 4.2, na sebe odkazují. Podle toho, jak jsou některé elementy nazvané, lze snadno detekovat, o kterou verzi WSDL se jedná.



Obrázek 4.2: Struktura WSDL dokumentu (zdroj: https://en.wikipedia.org/wiki/WSDL)

Element `service` obsahuje libovolný počet elementů `port` / `endpoint` (reprezentující skupinu endpointů dané webové služby), které se odkazují na nějaký konkrétní element `binding`. Element `binding` se odkazuje do konkrétní sekce na element `portType` / `interface` do abstrakní sekce a obsahuje libovolné množství elementů `operation`, jejichž jména odkazují na elementy `operation` definované v abstraktní sekci daného elementu `portType` / `in-`

`terface`. Každý element `operation` definovaný v konkrétní sekci pod elementem `binding` představuje jeden endpoint, tak jak byl definován v sekci 3.11.

Obecně vzato tedy lze v abstraktní sekci definovat element `portType` / `interface`, na který nebude v konkrétní sekci odkazovat žádný element `binding`. Tato skupina endpointů je pak pouze definována, ale webová služba ji neposkytuje. Analogicky ne všechny elementy `operation` v elementu `portType` / `interface` musí mít odpovídající element `operation` v elementu `binding` (endpointy, které jsou pouze definované, ale služba je neposkytuje).

Element `operation` (z abstraktní části dokumentu) obsahuje informace o vstupu a výstupu endpointu. Definice vstupu a výstupu se v závislosti na verzi WSDL dokumentu liší, ale vždy je zadefinována informace o gramatice, kterou musí vstup či výstup splňovat (formou XSD schématu[2]).

Příklad popisného dokumentu WSDL 2.0

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.w3.org/ns/wsdl"
             xmlns:tns="http://www.tmsws.com/wsdl20sample"
             xmlns:whttp="http://schemas.xmlsoap.org/wsdl/http/"
             xmlns:wsoap="http://schemas.xmlsoap.org/wsdl/soap/"
             targetNamespace="http://www.tmsws.com/wsdl20sample">

<!-- Abstract type -->
  <types>
    <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
               xmlns="http://www.tmsws.com/wsdl20sample"
               targetNamespace="http://www.example.com/wsdl20sample">

      <xs:element name="request"> ... </xs:element>
      <xs:element name="response"> ... </xs:element>
    </xs:schema>
  </types>

<!-- Abstract interfaces -->
  <interface name="Interface1">
    <fault name="Error1" element="tns:response"/>
    <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
      <input messageLabel="In" element="tns:request"/>
      <output messageLabel="Out" element="tns:response"/>
    </operation>
```

[2] http://www.w3.org/TR/xmlschema11-1/

```
26    </interface>
27
28  <!-- Concrete Binding Over HTTP -->
29    <binding name="HttpBinding" interface="tns:Interface1"
30          type="http://www.w3.org/ns/wsdl/http">
31      <operation ref="tns:Get" whttp:method="GET"/>
32    </binding>
33
34  <!-- Concrete Binding with SOAP-->
35    <binding name="SoapBinding" interface="tns:Interface1"
36          type="http://www.w3.org/ns/wsdl/soap"
37          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
38          wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response"
        >
39      <operation ref="tns:Get" />
40    </binding>
41
42  <!-- Web Service offering endpoints for both bindings-->
43    <service name="Service1" interface="tns:Interface1">
44      <endpoint name="HttpEndpoint"
45              binding="tns:HttpBinding"
46              address="http://www.example.com/rest/"/>
47      <endpoint name="SoapEndpoint"
48              binding="tns:SoapBinding"
49              address="http://www.example.com/soap/"/>
50    </service>
51  </definitions>
```

4.1.3 WADL

Dokument WADL má pevnou strukturu dat[3] ve formátu XML a popisuje právě jednu RESTovou webovou službu.

Kořenový element dokumentu WADL může obsahovat element `grammars` s definicí použitých gramatik a libovolné množství elementů `resources`. Element `resources` může obsahovat libovolné množství celkem tří různých elementů: `resource`, `method` a `param`. Stejně tak každý element `resource` může obsahovat libovolné množství stejné trojice elementů. Vzniká tak stromová struktura, která má za kořen element `resources`, mající za uzly elementy `resource` a za listy elementy `method` a `param`.

Element `resources` obsahuje informaci o absolutní cestě ke skupině endpointů, které jsou pod ním definovány. Element `resource` obsahuje relativní

[3] http://www.w3.org/Submission/wadl/

část cesty ke skupině endpointů, které jsou pod ním definovány a tato část je při vyhodnocování vložena na konec cesty definované v elementu `resource` o úroveň výše.

Element `method` reprezentuje jeden endpoint, tak jak byl definován v sekci 3.11. Díky stromové struktuře elementů `resource` lze vyhodnocením relativních cest až ke kořeni (elementu `resources`) pro endpoint definovat jeho absolutní cestu. Dále tento element obsahuje informaci o požadovaném formátu požadavku a odpovědi (buď jako počet a datový typ parametrů, nebo odkaz na gramatiky). Konkrétní element `method` se svým nadřazeným elementem `resource` odpovídá v termínech zdrojově orientovaného komunikačního vzoru volání jednoho HTTP slovesa nad zdrojem (viz sekce 2.3.3).

Element `param` existuje buď v kontextu elementu `resource`, kde představuje parametr, který je společný všem endpointům v definované úrovni, nebo v kontextu elementu `method`, kde představuje unikátní parametr pro daný endpoint. Element `param` obsahuje své jméno, datový typ / referenci na gramatiku, informaci o tom, zda je povinný atp.

Příklad popisného dokumentu WADL

```
<?xml version="1.0"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://wadl.dev.java.net/2009/02 wadl.xsd"
 xmlns:tns="urn:yahoo:yn"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:yn="urn:yahoo:yn"
 xmlns:ya="urn:yahoo:api"
 xmlns="http://wadl.dev.java.net/2009/02">
  <grammars>
    <include
      href="NewsSearchResponse.xsd"/>
    <include
      href="Error.xsd"/>
  </grammars>

  <resources base="http://api.search.yahoo.com/NewsSearchService/V1/">
    <resource path="newsSearch">
      <method name="GET" id="search">
        <request>
          <param name="appid" type="xsd:string"
            style="query" required="true"/>
          <param name="query" type="xsd:string"
            style="query" required="true"/>
```

```
24          <param name="type" style="query" default="all">
25            <option value="all"/>
26            <option value="any"/>
27            <option value="phrase"/>
28          </param>
29          <param name="results" style="query" type="xsd:int" default="10"/>
30          <param name="start" style="query" type="xsd:int" default="1"/>
31          <param name="sort" style="query" default="rank">
32            <option value="rank"/>
33            <option value="date"/>
34          </param>
35          <param name="language" style="query" type="xsd:string"/>
36        </request>
37        <response status="200">
38          <representation mediaType="application/xml"
39            element="yn:ResultSet"/>
40        </response>
41        <response status="400">
42          <representation mediaType="application/xml"
43            element="ya:Error"/>
44        </response>
45      </method>
46    </resource>
47  </resources>
48
49 </application>
```

4.2 Úprava datového modelu a nově zavedená metadata

Pro reprezentaci popisu webových služeb ve výše zmíněném datovém modelu je třeba namapovat entity přirozeně popisující webovou službu (viz sekce 4.1) na entity datového modelu.

Webové službě zřejmě odpovídá zdroj (Resource). Obojí je samostatná entita, která poskytuje nějaké schopnosti (Capabilities) svému okolí. Webová služba splňující požadavky na servisně orientovanou architekturu je samostatná (viz sekce 2.2), tudíž nebude mít žádné požadavky (Requirements). Endpointy webové služby můžeme popsat pomocí entity *Capability* a pro popis formátu odpovědi endpointu a parametrů jeho požadavku můžeme využít entitu *Property*, kterých muže mít entita *Capability* libovolné množství a které nastavíme vlastnost namespace tak, abychom mohli rozlišovat mezi vstupním parametrem a odpovědí.

Z výše uvedeného vyplývá, že není potřeba žádným způsobem měnit stávající datový model. Pouze budou zavedeny nové druhy používaných metadat. V metadatech budou zavedeny nové prvky Capabilities a Properties, každý se svým unikátním jmenným prostorem (namespace) a pro každý tento prvek budou definovány nové atributy tak, aby bylo možné v datovém modelu uchovávat popisy webových služeb z vybrané množiny technologií webových služeb.

4.2.1 Capability webservice.identity

Reprezentuje souhrnné vlastnosti webové služby (jméno webové služby je ukládáno do Capability crce.identity). Souhrn všech atributů této Capability je následující:

- **idl-version** (typ String) - Verze popisného schématu popisujícího tuto webovou službu.
- **type** (typ String) - Komunikační vzor používaný touto webovou službou.
- **uri** (typ String) - URL adresa, na které je webová služba dostupná.
- **idl-uri** (typ String) - URL adresa, na které je dostupný popisný dokument popisující tuto webovou službu.
- **parsedAtTimestamp** (typ Double) - Čas, kdy byl popisný dokument popisující tuto webovou službu zpracován a popis webové služby byl uložen v podobě metadat do úložiště.

4.2.2 Capability webservice.endpoint

Reprezentuje konkrétní endpoint webové služby. Souhrn všech atributů této Capability je následující:

- **name** (typ String) - Jméno endpointu.
- **url** (typ String) - URL adresa, na které je endpoint dostupný.

4.2.3 Property webservice.endpoint.parameter

Reprezentuje konkrétní vstupní parametr endpointu webové služby. Souhrn všech atributů této Property je následující:

- **name** (typ String) - Jméno parametru.
- **type** (typ String) - Datový typ parametru, popř. odkaz na gramatiku.
- **order** (typ Long) - Pořadí parametru v rámci všech parametrů endpointu.
- **isOptional** (typ Long) - Obsahuje logickou `0` nebo `1`. Značí, zda je parametr dobrovolný.
- **isArray** (typ Long) - Obsahuje logickou `0` nebo `1`. Značí, zda je jako tento parametr očekávána jedna instance daného datového typu nebo pole těchto instancí.

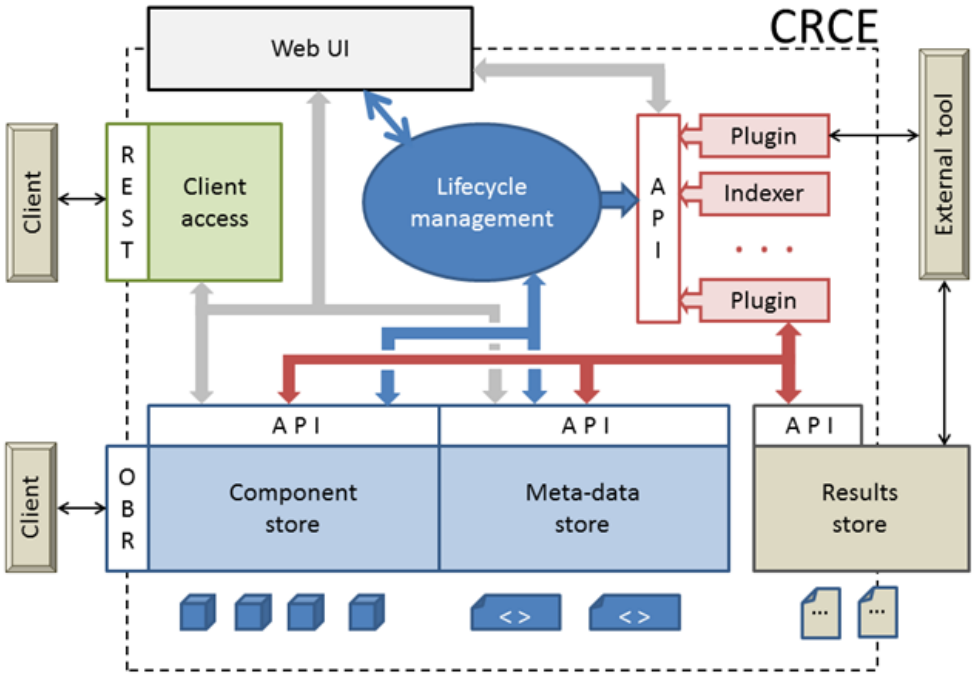
4.2.4 Property webservice.endpoint.response

Reprezentuje konkrétní odpověď endpointu webové služby. Souhrn všech atributů této Property je následující:

- **type** (typ String) - Datový typ vrácené odpovědi, popř. odkaz na gramatiku.
- **isArray** (typ Long) - Obsahuje logickou `0` nebo `1`. Značí, zda je jako tento parametr očekávána jedna instance daného datového typu nebo pole těchto instancí.

5 Úložiště CRCE

Úložiště CRCE je softwarový projekt vytvořený Katedrou informatiky a výpočetní techniky ZČU v Plzni, který umožňuje ukládání komponentových aplikací, vytváření jejich metadat a následnou kontrolu kompatibility použitých komponent. Více o této problematice lze nalézt v [Brada – Ježek(2012)]. Obr. 5.1 zachycuje architekturu úložiště, která bude nyní stručně popsána.



Obrázek 5.1: Architektura úložiště CRCE (zdroj: https://www.assembla.com/spaces/crce)

Základem úložiště jsou tři samostatná podúložiště:

1. Úložiště komponent (Component store), kam jsou ukládány nahrávané komponentové aplikace (artefakty).

2. Úložiště metadat (Meta-data store), kam jsou ukládána metadata popisující vlastnosti a závislosti komponent uložených v úložišti komponent.

3. Úložiště výsledků (Results store), kam jsou ukládány výsledky generované rozšiřujícími moduly v CRCE nebo externími nástroji, které s těmito moduly spolupracují (může se jednat například o výsledky různých testů prováděných nad uloženými komponentami, které mohou sloužit jako doplňující informace k metadatům při vyhodnocování komponent)

Součástí úložiště je webové rozhraní (Web UI), přes které lze do CRCE artefakty nahrávat, stahovat a zobrazovat jejich metadata. Po nahrání artefaktu přes webové rozhraní je tento postupně zpracován řetězcem pluginů / indexerů, které mohou zapisovat a číst data ve všech třech popsaných podúložištích. Tyto pluginy / indexery implementují jednotné rozhraní, díky čemuž lze postupně zadefinovanými pluginy / indexery proiterovat a předat jim referenci na nahrávaný artefakt, který mohou postupně zpracovat.

Součástí uložiště je RESTová webová služba poskytující přístup k úložišti komponent a metadat a součást pro řízení životního cyklu artefaktů (Lifecycle management), která dohlíží na stav artefaktu od jeho nahrání do bufferu přes webové rozhraní až po jeho commit do úložiště komponent a promítá tento stav do metadat.

5.1 Použité technologie

Úložiště je implementováno na technologii *OSGi*[1]. Pro překlad a spuštění projektu se používá *Apache Maven*[2] (pro spuštění se používá plugin Mavenu `pax`[3]). Projekt se skládá z modulů za použití technologie OSGi frameworku *Apache Felix*[4]. Pro úspěšný provoz je také třeba, aby byl na cílovém stroji dostupný *Java Development Kit*[5] a *MongoDB*[6] server naslouchající na standardním portu (27017).

[1] http://www.osgi.org
[2] https://maven.apache.org/
[3] https://ops4j1.jira.com/wiki/display/paxrunner/Pax+Runner
[4] http://felix.apache.org/
[5] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[6] https://www.mongodb.org/

Více informací o nutných prerekvizitách a použitých technologiích lze nalézt na stránkách projektu[7].

5.2 Pluginy a Indexery

Všechny pluginy / indexery jsou v rámci projektu úložiště CRCE třídy, implementující jednotné rozhraní. Toto rozhraní lze programově využít tak, že lze postupně iterovat přes indexery a každému přes rozhraní předat referenci na zpracovávaný CRCE artefakt. Každý indexer artefakt vyhodnotí a do metadat přidá nové informace, podle toho jaký má daný indexer účel. Výsledkem je komplexně a přehledně oindexovaný artefakt.

V současné implementaci úložiště je napevno definován jeden řetězec těchto indexerů určených pro indexaci komponentových aplikací v podobě souborů (jednotlivé indexery v tomto řetězci pevně definovány nejsou a dají se měnit). Rozšiřující modul, který bude vyvinut v rámci této práce bude zpracovávat jiný druh artefaktů: IDL dokumenty umístěné na URL adresách popisující webové služby. Proto bude pro tyto druhy artefaktů zvolena vlastní specifická indexace, která nepoužívá zmíněný řetězec indexerů.

Do budoucna by bylo výhodné vytvořit takovou funkcionalitu, která umožní jakýkoliv artefakt oindexovat libovolnou posloupností řetězců indexerů na základě průběžného vyhodnocování již oindexovaných metadat. Po zpracování libovolným indexerem by měla existovat možnost se na základě informací, které byly o artefaktu zjištěny předchozími indexery, rozhodnout, jaký řetězec indexerů bude artefakt zpracovávat dále.

5.3 Překlad a spuštění

Pro úspěšný překlad a spuštění projektu je třeba následujících kroků:

1. Instalace technologií JDK, Mongo DB and Apache Maven (viz sekce 5.1).

[7] https://www.assembla.com/spaces/crce/wiki

2. Získání zdrojových kódů z repozitáře projektu a nastavení konfiguračních souborů v adresáři `/modules/conf`.

3. Překlad projektu z kořenového adresáře pomocí nástroje Apache Maven.

4. Spuštění projektu z adresáře `/modules` pomocí nástroje Apache Maven a jeho pluginu pax.

Po úspěšném spuštění projektu by mělo být jeho webové rozhraní přístupné na adrese `http://localhost:8080/crce`. Více informací o překladu a spouštění lze nalézt na stránkách projektu[8].

[8]https://www.assembla.com/spaces/crce/wiki

6 Implementace rozšiřujícího modulu CRCE

Ve shodě se zadáním práce byl v projektu úložiště CRCE vytvořen rozšiřující modul umožňující získávání a ukládání popisu webových služeb z množiny vybraných technologií webových služeb. Pro modul byl zvolen název `Webservices Indexer`. Při implementaci byl mimo jiné kladen důraz na budoucí rozšiřitelnost množiny vybraných technologií webových služeb (tedy o možnost snadno doprogramovat získávání a ukládání popisu webových služeb pro dosud neimplementovanou technologii webové služby).

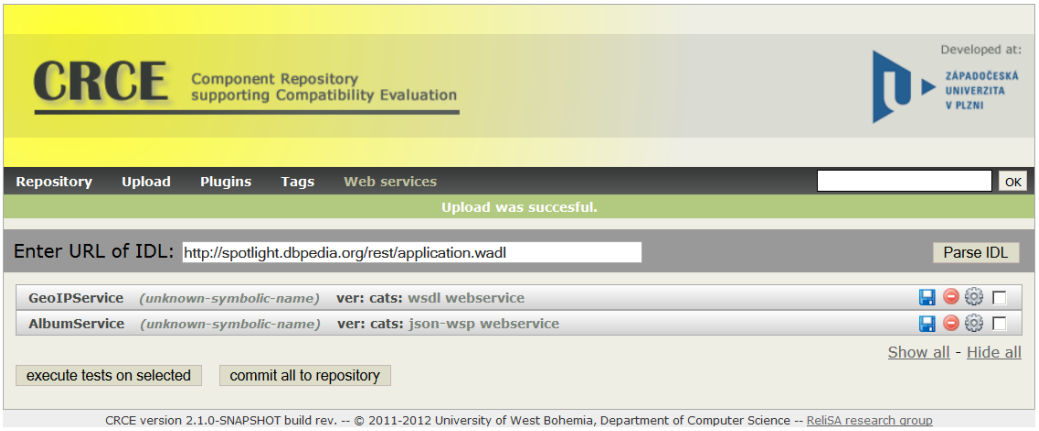
Kód včetně komentářů byl vyhotoven v anglickém jazyce, byly dorženy konvence definované pro psaní kódu v tomto projektu[1] a pro všechny metody ve všech třídách a rozhraních byl vytvořen JavaDoc.

6.1 Integrace modulu v úložišti

Aby bylo možné modul v úložišti používat, bylo třeba upravit kód některých ostatních modulů. Mezi nejvíce upravené cizí moduly patří modul pro webové rozhraní úložiště - `Web UI`. Bylo třeba v něm vytvořit novou sekci, která umožní uživateli získat a uložit popis webové služby. Z toho vyplývá, že modul webového rozhraní je tím modulem, který má na tento modul vytvořenou závislost a volá jeho funkcionalitu.

Nově vytvořená sekce ve webovém rozhraní je co do funkčnosti téměř identická se stávající sekcí pro nahrávání lokálně uložených souborů komponentových aplikací. Také používá buffer, který zdroj zpracuje, a ten je pak možné uložit do úložiště nebo smazat. Nově vytvořená sekce se však liší tím, že zpracovává popisné dokumenty na nějaké URL adrese, proto prvek pro nahrávání souborů nahradilo textové pole, do kterého uživatel může vepsat / zkopírovat URL, na kterém se nachází popisný dokument a stisknutím tlačítka zahájit jeho zpracování a uložení reprezentace webové služby do bufferu (viz Obr. 6.1).

[1] https://www.assembla.com/spaces/crce/wiki/Code_style



Obrázek 6.1: Ukázka sekce webového rozhraní pro získávání a ukládání popisu webových služeb

Ukládání metadat v úložišti je silně svázáno s nějakým artefaktem (souborem), který metadata popisují, a který je také uložen v úložišti. Proto je v případě získávání a ukládání popisu webové služby jako tento artefakt uložen zpracovávaný popisný dokument. Tento postup navíc přináší tu výhodu, že uložením popisného dokumentu z webu do úložiště zachytíme jeho podobu v čase (snapshot), jelikož se do metadat ukládá i informace o tom, kdy byl popisný dokument zpracován. Pokud je popisný dokument schopen definovat více webových služeb najednou, bude pro každý jednotlivý popis webové služby získaný z tohoto dokumentu uložena jedna celá jeho kopie.

6.2 Struktura modulu

Tento modul je, stejně jako všechny ostatní, implementován jako *OSGi bundle*[2] a svůj aktivátor, privátní a veřejné balíčky definuje skrze nástroj *bnd* jako:

```
1 Bundle-Activator: ${bundle.namespace}.internal.Activator
2 Private-Package: ${bundle.namespace}.internal, ${bundle.namespace}.structures.*
3 Export-Package: ${bundle.namespace}
```

Nyní budou popsány balíčky a důležité rozhraní / třídy, které obsahují.

[2] http://spring.io/blog/2008/02/18/creating-osgi-bundles/

6.2.1 Veřejná funkcionalita

Název balíčku: cz.zcu.kiv.crce.webservices.indexer

Veřejný balíček tohoho OSGi bundle definuje jediné rozhraní, které okolním modulům poskytuje funkcionalitu.

Rozhraní WebservicesDescription

Toto rozhraní definuje metodu, které je parametrem předána URL adresa, na které se nachází popisný dokument a metoda vrátí seznam zpracovaných popisů webových služeb obsažených v předaném dokumentu.

6.2.2 Privátní funkcionalita

Název balíčku: cz.zcu.kiv.crce.webservices.indexer.internal

Tento balíček obsahuje veškerou interní funkcionalitu modulu a třídu definovanou jako OSGi bundle aktivátor.

Rozhraní WebserviceType

Toto rozhraní definuje metody, které musí bezpodmínečně implementovat každá třída, která bude rozpoznávat a zpracovávat nějaký konkrétní typ popisného dokumentu vytvořeného podle příslušného popisného schématu. Obsahuje definici metody pro rozpoznání konkrétního implementovaného typu popisného dokumentu, definici metody pro jeho zpracování a definici metody, která vrací jméno popisného schématu, jehož dokumenty jsou touto implementací rozpoznávány (tedy např. JSON-WSP, WSDL, WADL, atp.).

Třída WebserviceTypeBase

Tato třída poskytuje funkcionalitu, která může být užitečná pro každou třídu implementující rozhraní `WebserviceType`, která tuto třídu rozšíří. Obsahuje reference na služby pro práci s metadaty, užitečné funkce a definice všech

jmenných prostorů a atributů, které jsou tímto modulem vytvářeny v metadatech.

Třída WebserviceTypeJsonWsp

Tato třída implementuje rozhraní `WebserviceType` a rozšiřuje třídu `WebserviceTypeBase`. Slouží k rozpoznávání a zpracování JSON-WSP description objektů (výsledkem zpracování je popis webové služby).

Třída WebserviceTypeWadl

Tato třída implementuje rozhraní `WebserviceType` a rozšiřuje třídu `WebserviceTypeBase`. Slouží k rozpoznávání a zpracování WADL dokumentů (výsledkem zpracování je popis webové služby).

Třída WebserviceTypeWsdl

Tato třída implementuje rozhraní `WebserviceType` a rozšiřuje třídu `WebserviceTypeBase`. Slouží k rozpoznávání a zpracování WSDL dokumentů (výsledkem zpracování je seznam popisů webových služeb).

Třída WebservicesDescriptionImpl

Tato třída implementuje rozhraní `WebservicesDescription`. Její jediná implementovaná metoda se stará o rozpoznání typu popisného dokumentu postupným voláním rozpoznávácí metody pro všechny registrované rozpoznávací třídy (implementující rozhraní `WebserviceType`). Při prvním úspěšném rozpoznání je rozpoznávací proces ukončen a pro třídu, která popisný dokument úspěšně rozpoznala, je zavolána její metoda pro zpracování rozpoznaného popisného dokumentu, která vrátí seznam rozpoznaných webových služeb.

6.2.3 Obecné datové struktury

Název balíčku: cz.zcu.kiv.crce.webservices.indexer.structures

Tento balíček obsahuje pomocné datové struktury použité při zpracovávání popisného dokumentu. Při zpracovávání popisného dokumentu jsou vytvořené instance těchto datových struktur naplněny, aby je následně bylo možné použít při vytváření metadat.

Třída Webservice

Tato třída uchovává data o webové službě získaná zpracováním popisného dokumentu. Uchovává atributy definované v sekci 4.2.1 a navíc seznam referencí na instance třídy `WebserviceEndpoint` reprezentující endpointy této webové služby.

Třída WebserviceEndpoint

Tato třída uchovává data o endpointu webové služby získaná zpracováním popisného dokumentu. Uchovává atributy definované v sekci 4.2.2 a navíc seznam referencí na instance třídy `WebserviceEndpointParameter` reprezentující parametry tohoto endpointu a referenci na instanci třídy `WebserviceEndpointResponse` reprezentující jeho odpověď.

Třída WebserviceEndpointParameter

Tato třída uchovává data o parametru endpointu webové služby získaná zpracováním popisného dokumentu. Uchovává atributy definované v sekci 4.2.3.

Třída WebserviceEndpointResponse

Tato třída uchovává data o odpovědi endpointu webové služby získaná zpracováním popisného dokumentu. Uchovává atributy definované v sekci 4.2.4.

6.2.4 Datové struktury specifické pro WSDL

Název balíčku: cz.zcu.kiv.crce.webservices.indexer.structures.wsdl

Tento balíček obsahuje dodatečné pomocné datové struktury použité při zpracovávání WSDL dokumentu. Ten má oproti ostatním zpracovávaným IDL dokumentům složitější strukturu, a tak bylo potřeba k obecným pomocným datovým strukturám použít ještě další specifické.

6.3 Funkcionalita modulu

Konkrétní případ užití tohoto modulu je tedy následující: uživatel přes webové rozhraní zadá URL adresu směřující na popisný dokument. Po odeslání požadavku je URL adresa předána tomuto modulu, který provede postupně následující operace:

1. Zjistí, zda se na zadané URL adrese nachází nějaký dokument (pokud ne, ohlásí chybu) a přečte ho.
2. Popisný dokument je rozpoznán a zpracován způsobem popsaným u třídy `WebservicesDescriptionImpl` v sekci 6.2.2.
3. Pro všechny získané popisy webových služeb jsou provedeny některé společné operace (přiřazena hlavní kategorie, nastaven čas zpracování atp.)

Poté se výsledek dostane zpět do modulu s webovým rozhraním a může být zobrazen výsledek v podobě popisu webové služby.

Při realizaci metod pro rozpoznávání a zpracování popisného dokumentu pro vybraná popisná schémata (JSON-WSP description object, WSDL, WADL) bylo dbáno na to, aby se detekce vždy slepě neřídila do posledního detailu specifikací. To s sebou přináší schopnost rozpoznat i nevalidní popisné dokumenty daného typu, u kterých je i přesto, že nesplňují specifikaci svého popisného protokolu, jasné, že nějakou webovou službu popisují. Tyto popisné dokumenty mohou být následně zpracovány alespoň částečně, ale v případě, že by existoval požadavek na jejich úplnou validitu, by k jejich zpracování nedošlo vůbec, protože by ani nebyly rozpoznány.

7 Ověření funkčnosti

V této kapitole budou popsány metody ověření správné funkčnosti modulu. Tj. faktu, že vytvořený modul správně rozpoznává, vytváří a ukládá popisy webových služeb z vybrané množiny technologií webových služeb.

Pro každou technologii z vybrané množiny technologií webových služeb bylo prohledáváním webu nalezeno větší množství popisných schémat, z nichž byly do užšího výběru vybrány ty, které se mezi sebou v různých aspektech (délce, složitosti popisované služby, rozdílné struktuře apod.) co nejvíce lišily, aby bylo otestováno co nejvíce okrajových případů. Každé z těchto popisných schémat bylo přes webové rozhraní zpracováno, nahráno do bufferu a uloženo do úložiště. Po úspěšném uložení artefaktu v úložišti byla přes webové rozhraní zobrazena jeho metadata a porovnáním bylo ověřeno, že vytvořený popis webové služby odpovídá informacím v popisném schématu.

Všechny popisné dokumenty, na kterých byl modul testován, byly správně rozpoznány, zpracovány a byly z nich vytvořeny popisy webových služeb v jednotné reprezentaci (viz sekce 4.2).

V kořenové složce zdrojových kódů přiložených k této práci na CD se nachází soubor `WEBSERVICES_DESCRIPTION_README.md`, který obsahuje více příkladů pro každou ze zpracovávaných technologií webové služby.

7.1 JSON-WSP description object

7.1.1 AlbumService

Jednoduchý popisný dokument, který záměrně využívá všechna pravidla definované popisným schématem. Obsahuje primitivní i složené datové typy.

URL popisného dokumentu:
http://ladonize.org/python-demos/AlbumService/jsonwsp/description

Výsledek:
Webová služba byla správně rozpoznána jako JSON-WSP a byl vytvořen její popis.

7.1.2 MUSE_services_V3

Popisný dokument z reálného provozu. Nedefinuje sice žádné složené datové typy, ale obsahuje velké množství endpointů.

URL popisného dokumentu:
http://grom.ijs.si:8001/MUSE_services_V3/jsonwsp/description

Výsledek:
Webová služba byla správně rozpoznána jako JSON-WSP a byl vytvořen její popis.

7.2 WSDL

7.2.1 AlbumService

Jednoduchý popisný dokument, který záměrně využívá všechna pravidla definované popisným schématem.

URL popisného dokumentu:
http://ladonize.org/python-demos/AlbumService/soap/description

Výsledek:
Webová služba byla správně rozpoznána jako SOAP a byl vytvořen její popis.

7.2.2 GeoIPService

Popisný dokument z reálného provozu. Pro všechny dokumenty používá XML namespace a obsahuje vetší množství všech elementů definovaných popisným schématem WSDL.

URL popisného dokumentu:
http://www.webservicex.net/geoipservice.asmx?WSDL

Výsledek:
Webová služba byla správně rozpoznána jako SOAP a byl vytvořen její popis.

7.3 WADL

7.3.1 Apache Camel Web Console and API

Popisný dokument z reálného provozu. Je specifický tím, že pro definované elementy používá XML namespace a v hlavičce popisného dokumentu obsahuje dokumentaci.

URL popisného dokumentu:
ftp://ftp.bgbilling.ru/pub/bgbilling/activemq/win/apache-activemq-5.4.2/webapps/camel/WEB-INF/classes/application.wadl

Výsledek:
Webová služba byla správně rozpoznána jako RESTová webová služba a byl vytvořen její popis.

7.3.2 fueleconomy.gov

Popisný dokument z reálného provozu. Je specifický tím, že používá více úrovní elementů `resource` a URL cesta k endpointům tak musí být při jejich zpracování složena z více částí.

URL popisného dokumentu:
https://www.fueleconomy.gov/ws/rest/application.wadl

Výsledek:
Webová služba byla správně rozpoznána jako RESTová webová služba a byl vytvořen její popis.

8 Závěr

V rámci této diplomové práce byla provedena analýza servisně orientované architektury a technologií webových služeb. Z této analýzy byly vyvozeny principy, podle kterých lze konkrétní technologie webových služeb kategorizovat a byly definovány obecné vlastnosti, které jsou všem technologiím webových služeb společné.

Následně byla na základě předchozí analýzy vytvořena reprezentativní množina existujících technologií webových služeb (JSON-WSP, SOAP, RESTová webová služba), pro které byla vytvořena jednotná datová reprezentace popisů webových služeb spadajících do technologií z vybrané množiny. Na základě této reprezentace bylo rozhodnuto, že stávající datový model používaný v úložišti CRCE je dostatečný a není třeba ho upravovat.

Datová reprezentace popisů webových služeb, spadajících do technologií z vybrané množiny, byla vhodně namapována na entity stávajícího datového modelu a do úložiště CRCE byl vytvořen rozšiřující modul, který tyto popisy umí získávat a do úložiště ukládat. Po vytvoření byl modul otestován na několika netriviálních webových službách pro každou technologii z vybrané množiny, aby byla dokázána jeho správná funkčnost.

Další aktivity navazující na tuto práci mohou být zaměřeny především na vylepšení modulu o možnost vytvářet popisy webových služeb spadajících pod dosud neimplementované technologie webových služeb. Rozšiřující modul byl navržen s ohledem na tuto možnost a implementací, pro tento účel definovaného rozhraní, ho lze snadno takto rozšířit. Mezi další možná vylepšení lze zařadit například možnost uchovávat detailní reprezentaci gramatik a složených datových typů, které různé typy popisů webových služeb poskytují (nyní jsou uchovávány pouze reference, které na ně odkazují). Do budoucna lze také zvolit nějakou vhodnou jednotnou reprezentaci datových typů (např. tu, která je použitá v XML schema) a primitivní datové typy použité v popisu webových služeb na ni mapovat (nyní jsou v popisech webových služeb zobrazeny primitivní datové typy, které jsou specifické pro každou technologii webové služby).

Literatura

[pro(2012)] *metaprotocol taxonomy* [online]. 2012. [cit. 2015-03-30]. Dostupné z: `http://hessian.caucho.com/doc/metaprotocol-taxonomy.xtp`.

[Allamaraju(2010)] ALLAMARAJU, S. *RESTful Web Services Cookbook*. 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly Media, Inc., 2010. ISBN 978-0-596-80168-7.

[Brada – Ježek(2012)] BRADA, P. – JEžEK, K. Ensuring Component Application Consistency on Small Devices : A Repository-Based Approach. In *38th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE Computer Society Press, September 2012. doi: 10.1109/SEAA.2012.48. ISBN 9780769547909.

[Daigneau(2012)] DAIGNEAU, R. *Service Design Patterns: fundamental design solution for SOAP/WSDL and restful Web services*. New Jersey : Pearson Education, Inc., 2012. ISBN 978-0-321-54420-9.

[Desel et al.(2004)] DESEL, J. – PERNICI, B. – WESKE, M. *Business Process Management: Second International Conference, BPM 2004, Potsdam, Germany, June 17-18, 2004, Proceedings*. Č. sv. 2 v Lecture Notes in Computer Science. Springer, 2004. Dostupné z: `https://books.google.cz/books?id=F75MqHCmkXsC`. ISBN 9783540222354.

[Dijkman – Dumas(2004)] DIJKMAN, R. – DUMAS, M. Service-oriented Design: A Multi-viewpoint Approach. *International Journal of Cooperative Information Systems*. December 2004, 13, 4, s. 337–378. Dostupné z: `http://eprints.qut.edu.au/622/`.

[Fielding(2000)] FIELDING, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[Graham(2002)] GRAHAM, S. *Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI*. Java (Sams). Sams, 2002. Dostupné z: `https://books.google.cz/books?id=5TPEHhuIum8C`. ISBN 9780672321818.

[Hellbruck et al.(2013)] HELLBRUCK, H. – TEUBLER, T. – FISCHER, S. Name-Centric Service Architecture for Cyber-Physical Systems (Short Paper). *2012 Fifth IEEE International Conference on Service-Oriented Computing and Applications (SOCA)*. 2013, 0, s. 77–82. doi: http://doi.ieeecomputersociety.org/10.1109/SOCA.2013.63.

[Lamb(1983)] LAMB, D. A. *Sharing Intermediate Representations: The Interface Description Language*. PhD thesis, Pittsburgh, PA, USA, 1983. AAI8400068.

[Merrick et al.(2006)] MERRICK, P. – ALLEN, S. – LAPP, J. XML remote procedure call (XML-RPC), April 11 2006. Dostupné z: `http://www.google.com/patents/US7028312`. US Patent 7,028,312.

[Nicholson(2014)] NICHOLSON, R. Agility and Modularity: Two Sides of the Same Coin. Technical report, OSGi Alliance, 05 2014. Dostupné z: `http://www.osgi.org/wiki/uploads/Resources/ AgilityandModularity2014v2.pdf`.

[Nurseitov et al.(2009)] NURSEITOV, N. et al. Comparison of JSON and XML Data Interchange Formats: A Case Study. s. 157–162. CAINE, 2009. Dostupné z: `http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf`.

[Robie et al.(2013)] ROBIE, J. et al. RESTful Service Description Language (RSDL): Describing RESTful Services Without Tight Coupling. In *The Markup Conference 2013. Balisage Series on Markup Technologies*, 2013. doi: 10.4242/BalisageVol10.Robie01.

[Troanca – BoianRemco(2013)] TROANCA, D. – BOIANREMCO, F. M. Xrdl: A Valid Description Language for XML-RPC. *STUDIA UNIVERSITATIS BABEŞ-BOLYAI, INFORMATICA*. 2013, LVIII, 3, s. 90–104. Dostupné z: `http://www.cs.ubbcluj.ro/ studia-i/2013-3/`.

[Weiss – Rychlý(2007)] WEISS, P. – RYCHLý, M. Modelování architektur založených na službách, 2007. grant: FR2233/2007/G1.