

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

Diplomová práce

Určování nahraditelnosti a kompatibility webových služeba

Místo této strany bude
zadání práce.

Prohlášení

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

V Plzni dne 19. dubna 2020

Zdeněk Valeš

Abstract

The text of the abstract (in English). It contains the English translation of the thesis title and a short description of the thesis.

Abstrakt

Text abstraktu (česky). Obsahuje krátkou anotaci (cca 10 řádek) v češtině. Budete ji potřebovat i při vyplňování údajů o bakalářské práci ve STAGu. Český i anglický abstrakt by měly být na stejné stránce a měly by si obsahem co možná nejvíce odpovídat (samozřejmě není možný doslovný překlad!).

Obsah

| | | |
|----------|---|-----------|
| 1 | Úvod | 7 |
| 2 | Principy webových služeb, techniky | 8 |
| 2.1 | Webové služby | 8 |
| 2.2 | REST | 8 |
| 2.2.1 | Technologie použité k realizaci webových služeb . . . | 9 |
| 2.3 | Protokoly | 9 |
| 2.4 | Formální popis webových služeb | 9 |
| 3 | Datové typy a porovnávání | 10 |
| 3.0.1 | Porovnávání datových typů | 10 |
| 4 | Popis ukládání metadat v CRCE, popis indexování API | 11 |
| 4.1 | CRCE | 11 |
| 4.1.1 | Metadata komponent | 12 |
| 4.1.2 | Komponenta v CRCE a její životní cyklus | 13 |
| 4.2 | Indexování API | 13 |
| 4.2.1 | Obecná indexace komponenty | 13 |
| 4.2.2 | Indexace API | 13 |
| 4.2.3 | Struktura metadat popisujících API | 14 |
| 4.2.4 | Indexování REST služeb | 15 |
| 4.2.5 | Indexování webových služeb podle popisu | 15 |
| 4.2.6 | Limity indexování | 16 |
| 5 | Popis funkce porovnávače (co se jak porovnává pro jaké typy API) | 18 |
| 5.1 | Popis porovnávacího algoritmu | 18 |
| 5.1.1 | Složitost algoritmu | 18 |
| 5.1.2 | Obecný algoritmus porovnání | 18 |
| 5.1.3 | Problémy při porovnávání | 19 |
| 5.1.4 | MOV flag | 21 |
| 5.2 | Výsledek porovnání - Diff | 21 |
| 5.2.1 | Vyhodnocení výsledku | 22 |
| 6 | Implementační detaily (jen stručně) | 24 |

| | | |
|----------|---|-----------|
| 7 | Testování | 25 |
| 7.1 | Integrační + akceptační testy | 25 |
| 7.1.1 | Syntetický server v Jave | 25 |
| 7.1.2 | Příklad JSON-WSP z wiki | 25 |
| 7.1.3 | FuelEconomy API | 26 |
| 7.1.4 | Stag WS | 26 |
| 8 | Závěr | 27 |
| | Literatura | 28 |
| | Seznam zkratek | 29 |

1 Úvod

- k čemu je práce dobrá - co text práce obsahuje - use case

2 Principy webových služeb, techniky

V této kapitole jsou definovány webové služby a související pojmy. Jsou zde představeny strojově čitelné způsoby popisu rozhraní služeb a protokoly sloužící ke komunikaci s webovými službami. Tato kapitola také obsahuje popis REST.

2.1 Webové služby

Pojem 'webová služba' má různé významy pro různé lidi, ale dá se najít několik společných bodů [2]:

- Použití HTML, XML a dalších standardů webové architektury jako stavebních kamenů
- Zaměření na podnikové a vnitropodnikové operace

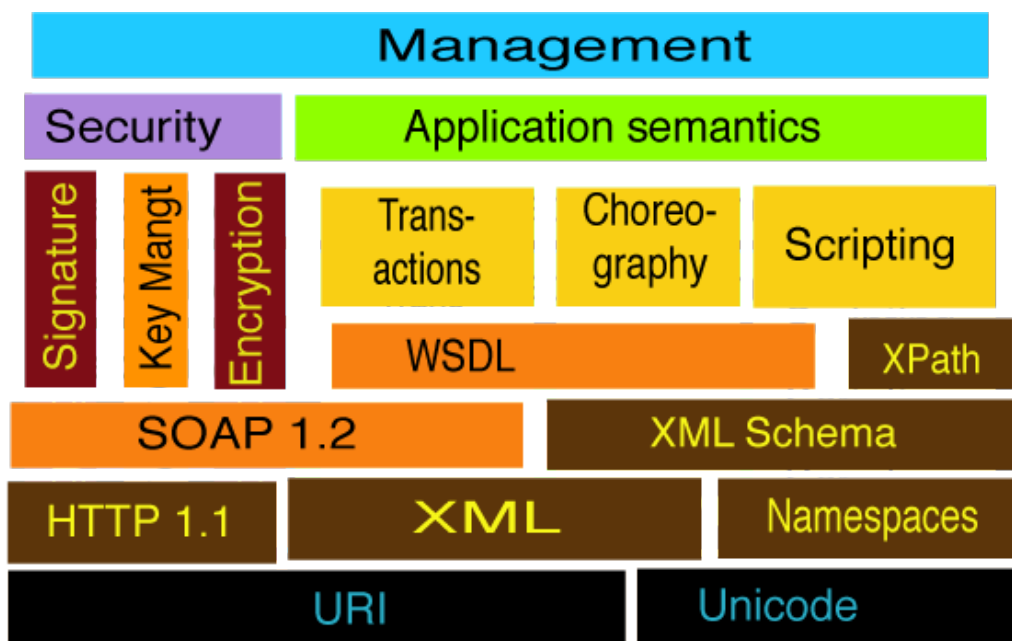
- pro účely této práce je použita následující definice od W3C [7]: "Webová služba je softwarový systém navržený pro podporu mezistrojové komunikace po síti. Webová služba má rozhraní, které je popsáno ve strojově čitelném formátu (konkrétně WSDL). Ostatní systémy interagují s webovými službami předepsaným způsobem za použití zpráv protokolu SOAP, které jsou typicky zprostředkované protokolem HTTP s využitím serializace XML a dalších webových standardů."

- poskytovatel služby = server
- konzument služby = klient (typicky nějaká aplikace)

2.2 REST

- mohlo by se hodit: <https://www.w3.org/TR/2004/NOTE-ws-arch-20040211/#relwwwrest>
- popisuje vztah REST a WWW - REST: založeno na manipulaci s XML reprezentací webových resources skrze stateless operace

- popis architektonického stylu REST: <https://www.ics.uci.edu/~fielding/pubs/dissertation/rest>
- popis elementů: - data, konektory, komponenty - popis view (na modelování)
 - RFC na HTTP: <https://tools.ietf.org/html/rfc7231#section-4>
 - odkaz konkrétně na request methods, mohlo by se hodit, protože ty jsou indexovány, tak alespoň na citaci



Obrázek 2.1: Technologie zahrnuté ve webových službách

2.2.1 Technologie použité k realizaci webových služeb

- zmínit: RPC, SOAP

2.3 Protokoly

- relevantní protokoly: RPC, SOAP, HTTP
 - HTTP (na REST a WS obecně)
 - protokol aplikační vrstvy SOAP (web service)
 - XML pro popis datového modelu
 - specifikace SOAP: <https://www.w3.org/TR/soap12-part1/#intro>

2.4 Formální popis webových služeb

- existují různé, strojově čitelné, formáty pro popis API
 - WSDL 1.1, 2.0, WADL (REST), JSON-WSP
 - Swagger, Raml, OpenApi
 - v případě REST bohužel není nic formálně nutné (oproti třeba SOAP), takže specifikace API nemusí být kompatibilní, nemusí být úplné, nebo můžou být ad-hoc (např. slovní popis ve Word dokumentu) a tím pádem nemusí existovat univerzální způsob strojového čtení těchto specifikací
 - v mé práci se věnuji především formátům WSDL, WADL a JSON-WSP

3 Datové typy a porovnávání

- přednášky z FJP - jak jazyky řeší datové typy - rekurzivní vs. nerekurzivní
- primitivní typy (v xsd) - built-in typy (v Java) - tady budu citovat [1] - subtyping: $A <: B \iff A$ může být použito v kdekoliv kde je očekáváno B
- kontravariance: $F'(A) <: F(B) \iff B <: A$

3.0.1 Porovnávání datových typů

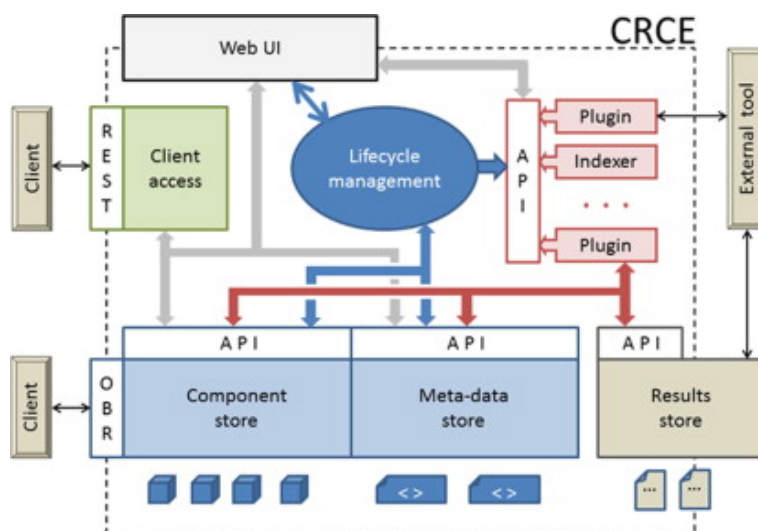
- jak to funguje - problémy při porovnání - subtyping vs. matching ([1])

4 Popis ukládání metadat v CRCE, popis indexování API

Cílem této práce je vytvořit rozšíření pro úložiště CRCE¹, které bude schopno vyhodnocovat kompatibilitu indexovaných webových služeb. Tato kapitola popisuje samotné úložiště, formát a obecný způsob získávání metadat a princip fungování konkrétních rozšíření, která indexují webové služby.

4.1 CRCE

CRCE je komponentové úložiště dlouhodobě vyvíjené a spravované výzkumnou skupinou ReliSA na Katedře Informatiky ZČU, jehož primárním účelem je indexace a následná kontrola vzájemné kompatibility komponent. Úložiště je postaveno na modulární architektuře (viz obrázek 4.1) a je tedy možné přidat rozšíření pro indexaci a zpracování vlastních dat.



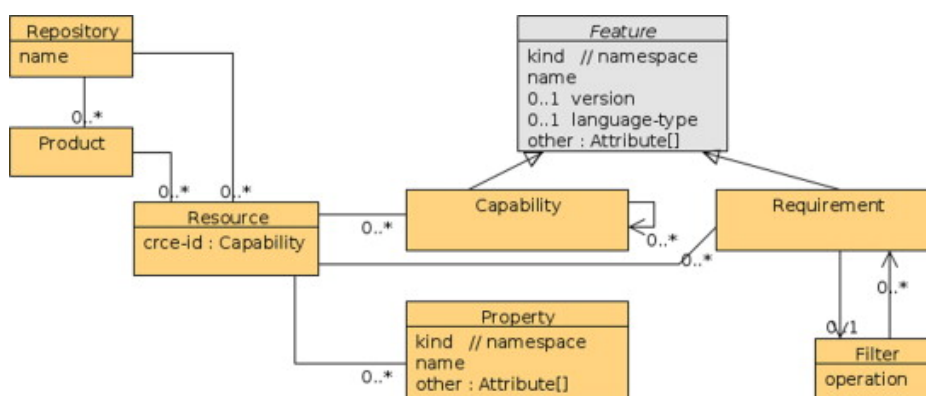
Obrázek 4.1: Architektura CRCE

¹Component Repository supporting Compatibility Evaluation

4.1.1 Metadata komponent

Data, která vzniknou indexací komponenty a případným dalším zpracováním (např. porovnáním) jsou uložena do souboru metadat a představují klíčový element systému CRCE. Návrh struktury těchto metadat, který je naznačen na obrázku 4.2, vychází z konceptu OBR² jehož základními entitami jsou mimo jiné *Resource*, *requirements* a *capabilities*[3].

Entita *Resource* reprezentuje uloženou komponentu, *requirements* a *capabilities* jsou množiny vlastností popisující co komponenta ke své správné funkci vyžaduje, respektive co naopak poskytuje. Model také umožňuje přidání key-value atributů k vlastnostem.



Obrázek 4.2: Reprezentace metadat v CRCE

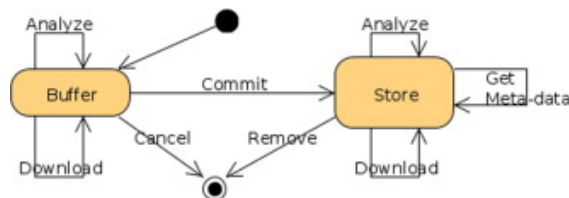
V mé práci jsem pracoval především s poskytovanými vlastnostmi (množina *capabilities*) a proto zde popíši hlavně jejich strukturu. Každá konkrétní vlastnost je reprezentována elementem *Capability* a od ostatních je odlišena identifikátorem *namespace*. Detaily konkrétní vlastnosti jsou popsány elementy *Property* a *Attribute*, kde *Property* reprezentuje logický celek několika atributů. Atributy pak představují páry klíč-hodnota, které nesou konkrétní informace jako např. jméno endpointu, nebo datový typ parametru.

Z obrázku 4.2 je vidět rekurzivní povaha elementu *Capability*, čehož je využito ke skládání jednodušších vlastností do složitějších celků. Vznikne tím stromová struktura, která je vhodná k modelování dat hierarchické povahy mezi něž patří například popisy webových služeb. V případě takto komplexních vlastností je ke komponentě (*Resource*) přiřazena pouze jedna, tzv. kořenová, *capability*, která reprezentuje celou vlastnost.

²OSGi bundle repository

4.1.2 Komponenta v CRCE a její životní cyklus

Úložiště bylo primárně navrženo pro ukládání OSGi, nicméně indexovat lze jakýkoliv soubor. Komponenta je v CRCE popsána již zmíněnými metadaty a prochází vlastním životním cyklem naznačeným na obrázku 4.3.



Obrázek 4.3: Životní cyklus komponenty v CRCE

Dvě hlavní části: buffer a store. V obou probíhá analýza, buffer: sběr informací o komponentě, store: operace a analýza sebraných dat. Pro mou práci jsou relevantní obě části, protože v bufferu dochází k převodu popisu API na metadata, ve store pak plugin, který je výsledkem mé práce počítá vzájemnou kompatibilitu API. Indexace komponent je popsána v následující sekci.

4.2 Indexování API

V této podkapitole je krátce popsán obecný způsob indexace komponent v CRCE. Následně je podrobněji rozebráno indexování API konkrétními moduly a reprezentace popisu API metadaty v CRCE. Na závěr jsou také uvedeny limity indexování.

4.2.1 Obecná indexace komponenty

Indexace komponenty a související sběr metadat je proveden ve fázi *Buffer* k tomu určenými moduly. Ty jsou vzájemně nezávislé a obecně platí, že každý nich je zaměřen na sběr nějaké logicky ucelené části dat jako například informace o OSGi bundlu, koordináty maven artefaktu, nebo popis webových služeb. Vlastní data komponenty zůstávají během sběru dat nezměněná což v kombinaci s řetězením indexerů zaručuje mimo snadnou rozšiřitelnost také transparentní přístup ke komponentě každému z nich.

4.2.2 Indexace API

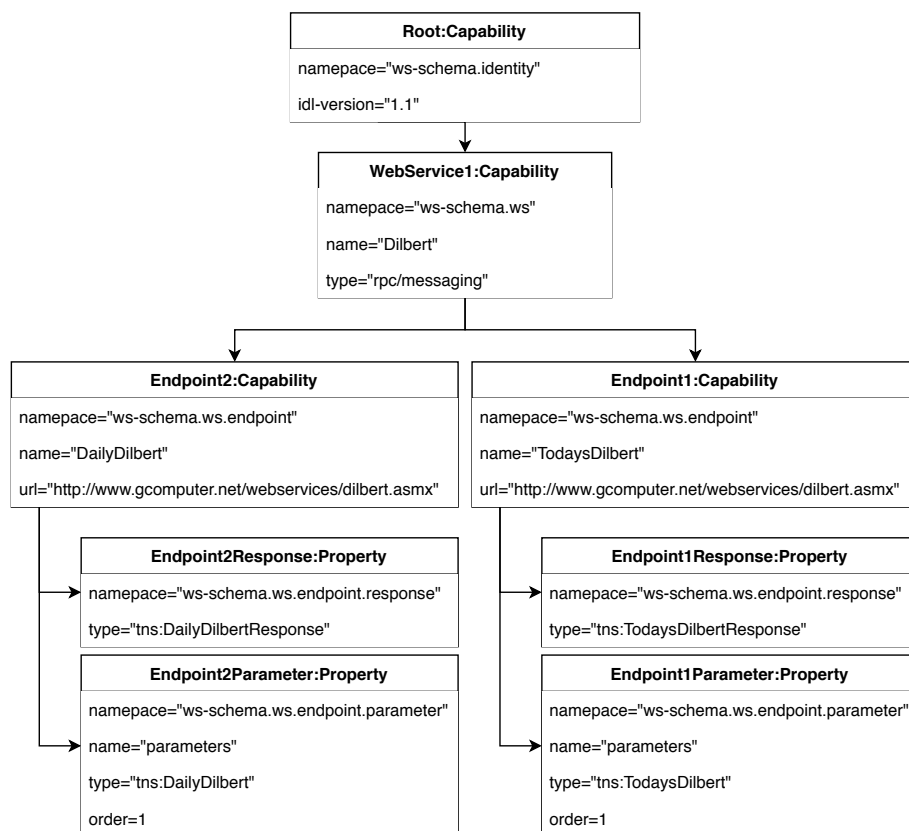
Soubor obsahující implementaci, nebo popis API je v CRCE vnímán jako komponenta a prochází tedy zmíněným životním cyklem včetně výše popsané

indexace. Z důvodů existence mnoha různých způsobů popisu webových služeb je není možné všechny analyzovat jedním indexerem a je nutné zaměřit se pouze na část z nich.

V současné době tedy existují dva moduly podporující několik popisných formátů a implementací. Konkrétně se jedná o modul pro indexaci webových služeb založených na architektonickém stylu REST[5] a o modul pro indexaci webových služeb s popisem ve formátu WSDL, WADL, nebo Json-WSP [6]. Oba dva vznikly v rámci diplomových prací a jsou stručně popsány v následujících sekcích.

4.2.3 Struktura metadat popisujících API

Jak již bylo zmíněno v části 4.1.1, hierarchickou strukturu popisu API lze vhodně vyjádřit metadaty CRCE. Během indexování komponenty reprezentující API jsou shromážděny různé typy popisných vlastností. Jedním z těchto typů je i samotný popis webové služby, který je reprezentován stromem metadat a ke komponentě je přiřazen skrze kořenovou *Capability*.



Obrázek 4.4: Příklad indexované SOAP webové služby pro komix Dilbert

I když jsou různé druhy API indexovány rozdílnými moduly, výsledná metadata mají podobnou strukturu. Příklad metadat API je zobrazen na objektovém diagramu 4.4, jedná se o webovou službu, která vrací strip komixu Dilbert pro dnešní den.

Z uvedeného obrázku je vidět, že klíčové elementy API jako web service, nebo endpoint jsou reprezentovány objektem *Capability*. Detaily těchto elementů jsou popsány objekty *Property*. Jedná se zejména o parametry endpointů, těla requestů a response. Objekt *Attribute* pak představuje konkrétní hodnoty, jež jsou na obrázku naznačeny jen jako "klíč=hodnota". *Attribute* nemusí být vázaný jen na *Property* a lze jej použít i pro popis *Capability*, jak je tomu např. u objektu *WebService1*.

4.2.4 Indexování REST služeb

Modul pro indexování REST API vznikl v rámci diplomové práce Bc. Gabriely Hessové. Princip sběru dat je založen na binární analýze java archivů (JAR) obsahujících implementaci REST služeb pomocí frameworků splňujících specifikaci JAX-RS a frameworku Spring Web MVC. Modul byl testován na frameworkcích Jersey verze 2.26, RESTEasy verze 3.0.16 a Spring Boot verze 1.5.9 [5].

Z implementace REST služby modul rekonstruuje kolekci endpointů s jejich parametry, tělem requestu, response a případnými parametry response. Každý endpoint je reprezentován elementem *Capability*, všechny další jeho vlastnosti pak elementy *Property*. Výčet všech indexovaných elementů rozhraní je uveden v tabulce 4.1.

| Element API | Entita v metadatech | Vstaženo k |
|--------------------|---------------------|------------|
| Endpoint | Capability | - |
| Request body | Property | Endpoint |
| Request parameter | Property | Request |
| Response | Property Endpoint | Endpoint |
| Response parameter | Property | Response |

Tabulka 4.1: Seznam indexovaných elementů REST služby a jejich reprezentací v metadatech

4.2.5 Indexování webových služeb podle popisu

Modul pro indexování webových služeb vznikl v rámci práce Bc. Davida Pejřimovského. Oproti předchozímu modulu pro indexaci REST služeb ne-

pracuje tento s její implementací, ale s pouhým popisem. Podporované formáty popisu služeb jsou WSDL (verze 1.1 i 2.0) pro SOAP webové služby a WADL, Json-WSP pro REST služby[6].

Struktura dat vytvořených pro REST služby (tedy z popisu WADL, nebo Json-WSP) je podobná struktuře dat vytvořenou předchozím indexerem. Endpoint je tedy reprezentován elementem *Capability*, parametry endpointu elementy *Capability*. Z popisu Json-WSP je ještě vytvořena reprezentace response pro daný endpoint (element *Property*). Z popisu WADL se žádné další vlastnosti endpointů nezískávají.

Z WSDL popisu je vytvořena reprezentace služeb jež jsou popsány xml elementy `<wsdl:service>` a vnořených endpointů. Endpoint je ve WSDL popsán elementem `<wsdl:port>` a má definované operace (`<wsdl:operation>`), nicméně modul tyto nevnořuje a vytváří zjednodušenou reprezentaci. Model endpointu tedy obsahuje metadata získaná z `<wsdl:operation>` a url definovanou v elementu `<wsdl:port>`. Služby i endpointy jsou v metadatach reprezentovány elementy *Capability*. Oproti REST službám, které mají jednu úroveň vnoření *Capability*, zde vznikají úrovně dvě. Výčet elementů API a jejich reprezentace v metadatach je uveden v tabulce 4.2.

| Element API | Entita v metadatach | Vstaženo k |
|--------------------|---------------------|------------|
| Service | Capability | - |
| Endpoint | Capability | Service |
| Endpoint parameter | Property | Endpoint |
| Response | Property | Endpoint |
| Response parameter | Property | Endpoint |

Tabulka 4.2: Seznam indexovaných elementů webové služby a jejich reprezentací v metadatach

Logika parsování WSDL souborů (verze 1.1. i 2.0) obsahovala chybu ve čtení adresy endpointu. Ta byla očekávána v atributu `action` elementu `<wsdl:operation>`, který ale není uveden ve specifikaci WSDL 1.1 [8] ani WSDL 2.0 [9]. Tuto chybu jsem v rámci mé práce opravil.

4.2.6 Limity indexování

Současný proces indexování webových služeb naráží na dva známé problémy týkající se datových typů. Jedná se o indexování rekurzivních datových typů a absenci samotných definic typů.

První problém se týká zejména indexování webových služeb podle popisných souborů, protože ty definice typů obsahují. Způsoby rozvoje a ukládání

datových typů jsou popsány v [1], nicméně logika zatím není implementována. Druhý problém se týká binární analýzy REST služeb, protože archiv s implementací služby nemusí nutně obsahovat definice tříd. Ty mohou být například v jiném artefaktu, na který se archiv pouze odkazuje skrze závislost.

Z těchto důvodů je do metadat uložen pouze název datového typu což snižuje možnosti porovnávání služeb.

5 Popis funkce porovnávače (co se jak porovnává pro jaké typy API)

V této kapitole je detailně popsána funkce porovnávacího algoritmu společně s daty, nad kterými je možné porovnávač použít. Zároveň je zde popsán způsob vyhodnocení výsledků porovnání a formát takto získaných dat.

5.1 Popis porovnávacího algoritmu

Porovnávací algoritmus pracuje s výše zmíněnými metadaty, reprezentovanými stromovou strukturou, jejíž uzly tvoří instance tříd `Capability`, `Properties` a `Attributes`. Algoritmus pracuje pouze s daty, která byla vytvořena indexery popsány v části 4.2. Ostatní metadata, jež mohou být případně navěšená na `Resource` reprezentující API zůstanou nedotčena. V současné době je možné porovnat pouze API, jejichž metadata byla vytvořena stejným indexerem. Není tedy možné porovnat například metadata REST API získaná binární analýzou jar s metadaty získanými čtením JSON-WSP dokumentu.

- zmínit taky omezení, která plynou z indexovaných dat

5.1.1 Složitost algoritmu

- v nejhorším případě:
 - WSDL: $O(n^3)$ (ws endpointy ve ws)
 - ostatní $O(n^2)$ (endpointy1 x endpointy 2)

5.1.2 Obecný algoritmus porovnání

TODO: obecný popis, formulovat lépe

Algoritmus pro vše krom WSDL obecně (WSDL má navíc výběr a porovnání webservice):

1. Vstupem jsou endpointy obou API: `endpoints1`, `endpoints2`
2. Vyber endpoint `e1` z `endpoints1`

3. Vyber endpoint **e2** z **endpoints2**, který je vhodný k porovnání s **e1**
4. Pokud neexistuje vhodný **e2**, ulož mezivýsledek reprezentující přebývající endpoint a jdi zpět na 2
5. Postupně porovnej metadata, parametry, tělo request, tělo response obou endpointů
6. Ulož objekt reprezentující mezivýsledek porovnání
7. Pokud není **endpoints1** prázdné, jdi na krok 2, jinak pokračuj dále
8. Pro všechny zbývající endpointy **endpoints2** ulož mezivýsledek reprezentující chybějící endpoint
9. Z mezivýsledků sestav finální objekt reprezentující výsledek porovnání dvou API

5.1.3 Problémy při porovnávání

TODO: rozepsat v patřičných subsections

1. jak vybrat který endpoint/ws porovnat s kterým
2. MOV - pick the best
3. datové typy (java built-in, xsd, custom)
4. verze v URL u REST API (taký vede na MOV)

Výběr endpointu/ws vhodného k porovnání

Metadata:

- kvůli MOV se teoreticky nelze spoléhat na jméno endpointu
- kvůli verzi v cestě k endpointu (popsáno dále v 5.1.3) se nelze spoléhat na cestu k endpointu

Počet parametrů

- některé mohou být nepovinné

Porovnávání datových typů

Custom datové typy: nejsou indexované a lze tedy porovnávat pouze podle jména (např. fully qualified name v případě Java tříd).

Built-in datové typy (java, xsd)

- Java: podpora typů z `java.lang` + dědičnost
- xsd: podpora built-in typů xsd (musí být správná předpona) + dědičnost ve smyslu 'vejde se do'

Verze REST API v cestě k endpointu

Proč:

- klient může chtít volat novou verzi API a je tedy žádané zjistit, jak moc je API kompatibilní (např. se mohla změnit jen implementace, takže signatura je stále stejná)
- Normálně by algoritmus skončil MUT, protože by kvůli rozdílným cestám k endpointům vyhodnotil endpointy z API 1 jako DEL a endpointy z API 2 jako INS
- to není žádané, takže algoritmus je schopný detekovat verzi v cestě k endpointu a při výběru endpointů k porovnání ji ignorovat

Podporovaný formát:

- `v<major>[.minor[.micro]]`
- lowercase i uppercase
- oddělovač může být '.', nebo '-'
- regex: `\[/[vV][0-9]+(?:[.-][0-9]+){0,2}\/`

Detekci verze je možno vypnout. Pokud je detekce zapnutá, algoritmus se před porovnáním cest pokusí najít verzi a pokud ji najde, z cesty ji vyřadí a porovná cesty bez verze

5.1.4 MOV flag

Popis MOV

Co: Příznak označující, že API/endpoint má (částečně) shodnou implementaci, ale nachází se na jiné adrese

Proč: endpointy v API mohou mít jiné url/jména, ale implementačně mohou být shodné -> potřeba detekovat

Jak: na základě ostatních metadat (počet parametrů, počet endpointů ve WS)

Mov se také nastaví pokud je zaplé ignorování verzi REST API v endpoint path a pokud:

- cesty s verzí nejsou stejné
- cesty bez verze jsou stejné

Algoritmy (nemusí vždy fungovat správně):

- obecná detekce před samotným porovnáním -> MovDetectionResult
- 3x diff: host, path to endpoint, operace
- MovDetectionResult se pak použije při výběru endpointu k porovnání a při samotném porovnání (pickBest)

- kombinace které vedou na mov:

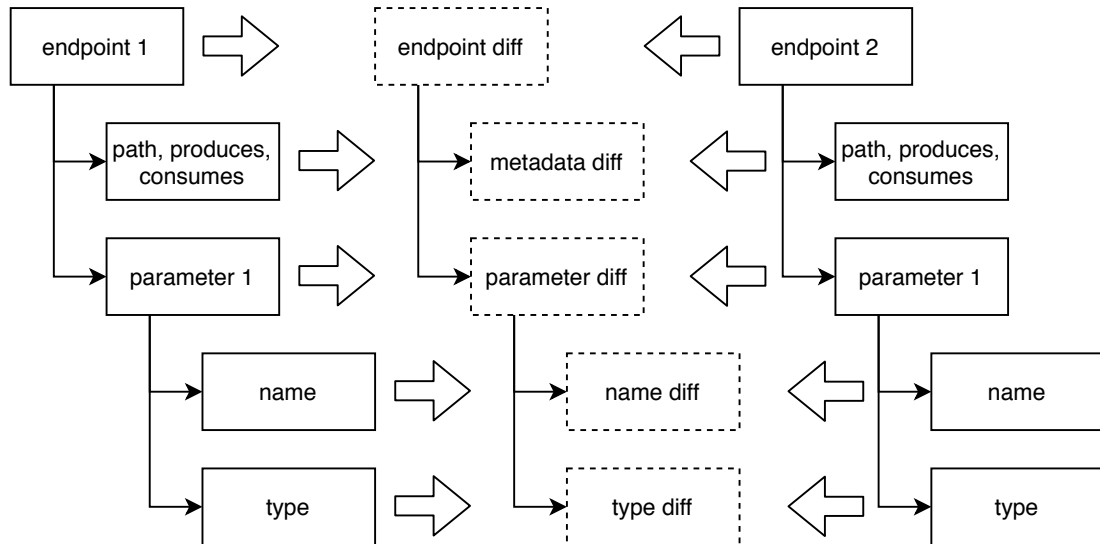
- $h \wedge !pe \wedge !o$
- $h \wedge pe \wedge !o$
- TODO

5.2 Výsledek porovnání - Diff

Popis výsledné datové struktury

- Diff, Compatibility
- vychází z [4]
- stromová struktura rozdílů mezi jednotlivými uzly stromu metadat
- obrázek 5.1 hezky popisuje jak to vznikne
- výsledné hodnoty diffu a jejich významy pro klienta v tabulce 5.1

- SPE/GEN může vzniknout jen z datových typů parametrů/response
-> lze spolehlivě použít kontravarianci a výsledek obrátit
- pokud tedy vyjde SPE, znamená to např. generalizovaný parametr a tedy je to pro klienta bezpečné



Obrázek 5.1: Vytvoření diffů

5.2.1 Vyhodnocení výsledku

- jak probíhá vyhodnocení (nejdříve se určí hodnoty listů, z nich se pak počítá dál nahoru)
 - kontravariance
 - není to úplně problém, ale při vyhodnocování finálního Diffu pro endpoint je potřeba brát v potaz
 - GEN/SPE může vzniknout jen z datových typů parametrů/response endpointu, takže je to poměrně přímočaré

| Difference type | Impact on client |
|----------------------|-----------------------|
| None (NON) | safe |
| Specialization (SPE) | safe |
| Insertion (INS) | safe |
| Deletion (DEL) | potentially dangerous |
| Generalization (GEN) | potentially dangerous |
| Mutation (MUT) | dangerous |
| Unkown (UNK) | dangerous |

Tabulka 5.1: Types of differences between two nodes

6 Implementační detaily (jen stručně)

- zmínit, proč třídy pro porovnávání REST API a WS nemají společného předka (krom rozhraní) - důvod: chtěl jsem nechat implementaci obou porovnávačů oddělenou pro případ, že by se změnila funkce indexerů

7 Testování

- nějaká reálná data - STAG (WSDL) - Fuel Economy - i syntetická data - algoritmus testován pomocí unit testů

7.1 Integrační + akceptační testy

- testování skrze REST API - pomocí Postman (Collection Runner) - několik verzí jednoho API -> testování křížem - todo: příklad - popsat verze API (čím se liší), případně zdůvodnit očekávaný výsledek - tabulka vzájemného porovnání s výsledky

7.1.1 Syntetický server v Java

- postaveno na Jersey
- REST API
- alespoň obrázek/raml api?
- 2 verze otestované křížem

Popis verzí API (rozdíl je vždy popsáný oproti verzi 1):

- V1: základní verze
- V2: typ parametru změněn z Long na Number

Co se testuje:

- GEN/SPEC a kontravariance u parametru endpointu

7.1.2 Příklad JSON-WSP z wiki

- popsáno JSON-WSP souborem
- <https://en.wikipedia.org/wiki/JSON-WSP>
- 4 verze otestované křížem

Popis verzí API (rozdíl je vždy popsáný oproti verzi 1):

- V1: základní verze

- V2: typ User je jiný
- V3: přidána metoda deleteUser
- v4: metoda listGroups změněna na getUsersInGroup

Co se testuje:

- ne-indexování custom datových typů (takže v2 = v1)
- mutace endpointu
- INS, DEL

7.1.3 FuelEconomy API

- popsáno WADL souborem
- <https://www.fueleconomy.gov/ws/rest/application.wadl>
- 3 verze otestované křížem

Popis verzí API (rozdíl je vždy popsáný oproti verzi 1):

- V1: základní verze
- V2: odebrán (poslední) resource /labelvehicle
- V3: odebrán (poslední) resource /labelvehicle, přidán resource /somethingDifferent

Co se testuje:

- INS, DEL a MUT

7.1.4 Stag WS

TODO

8 Závěr

Literatura

- [1] ABADI, M. – CARDELLI, L. On Subtyping and Matching. In *European Conference on Object-Oriented Programming (ECOOP), Lecture Notes in Computer Science*, 952, s. 145–167. ACM Press, January 1995. Dostupné z: <https://www.microsoft.com/en-us/research/publication/on-subtyping-and-matching/>.
- [2] BERNERS-LEE, T. *Web Services - Program Integration across Application and Organization boundaries* [online]. aug 2009. Dostupné z: <https://www.w3.org/DesignIssues/WebServices.html>.
- [3] BRADA, P. – JEZEK, K. Repository and Meta-Data Design for Efficient Component Consistency Verification. *Science of Computer Programming*. 2015, 97, part 3, s. 349–365. ISSN 0167-6423. doi: 10.1016/j.scico.2014.06.013. Dostupné z: <http://www.sciencedirect.com/science/article/pii/S0167642314002925>.
- [4] BRADA, P. – VALENTA, L. Practical Verification of Component Substitutability Using Subtype Relation. s. 38 – 45, 10 2006. doi: 10.1109/EUROMICRO.2006.50.
- [5] HESSOVÁ, G. Automatické získání historických údajů z webových zdrojů [online]. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2015 [cit. 2020-02-22]. Dostupné z: <https://theses.cz/id/pzbgj7/>.
- [6] PEJŘIMOVSKÝ, D. Vytváření a ukládání popisu webových služeb v úložišti CRCE [online]. Diplomová práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2015 [cit. 2020-02-22]. Dostupné z: <https://theses.cz/id/bb74eq/>.
- [7] W3C. *Web Services Architecture* [online]. feb 2004. Dostupné z: <http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/>.
- [8] W3C. *Web Services Description Language (WSDL) 1.1* [online]. mar 2001. Dostupné z: <https://www.w3.org/TR/2001/NOTE-wsdl-20010315>.
- [9] W3C. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language* [online]. jun 2007. Dostupné z: <https://www.w3.org/TR/wsdl/>.

Seznam zkratek

| | |
|------|--|
| CRCE | Component Repository supporting Compatibility Evaluation |
| API | Application Programming Interface |
| HTTP | Hypertext Transfer Protocol |
| JAR | Java Archive |
| JSON | JavaScript Object Notation |
| JSON | JSON Web-Service Protocol |
| OSGi | Open Services Gateway initiative |
| REST | Representational State Transfer |
| SOAP | Simple Object Access Protocol |
| UML | Unified Modeling Language |
| URI | Uniform Resource Identifier |
| URL | Uniform Resource Locator |
| WADL | Web Application Description Language |
| WSDL | Web Services Description Language |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |