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

Diplomová práce

Určování nahraditelnosti a kompatibility webových služeba

Plzeň 2020 Zdeněk Valeš

Místo této strany bude zadání práce.

Prohlášení

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

V Plzni dne 3. 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

- k čemu je práce dobrá - co text práce obahuje - use casy

2 Principy webových služeb, techniky

- co je to API - co jsou to webové služby - REST - asi by bylo fajn zmínit i XML - relevantní protokoly: - HTTP (na REST a obecné API) - protokol aplikační vrstvy - SOAP (web service) - typicky používá HTTP - používá XML pro popis datového modelu - specifikace: https://www.w3.org/TR/soap12-part1/#intro - 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 použitých technologií: XML, SOAP, WSDL - formální popis použití WS - popis architektonického stylu REST: https://www.ics.uci.edu/ fielding/pubs/dissertation/rest_arch_style.htm - popis elementů: - data, konektory, komponenty - popis view(na modelování) - RFC na HTTP: https://tools.ietf.org/html/rfc
4 - odkaz konkrétně na request methods - mohlo by se hodit, protože ty jsou indexovány, tak alespoň na citaci - existují různé, strojově čitelné, formáty pro popis API - WSDL, WADL, 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

3 Datové typy a porovnávání

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

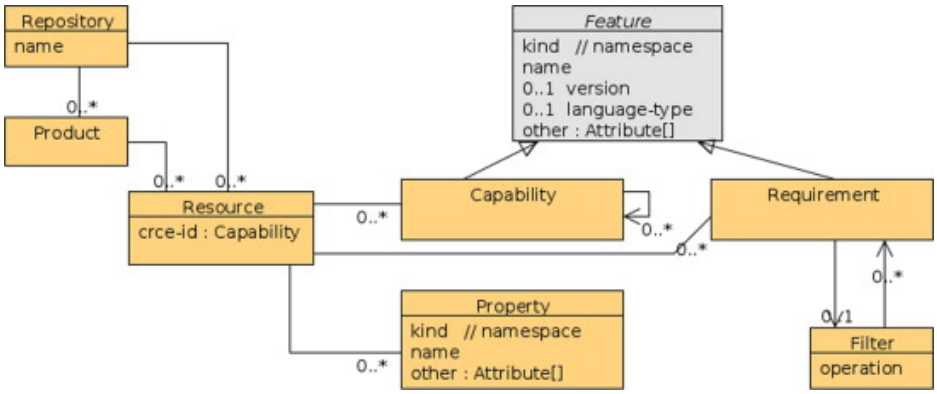
Tato kapitola popisuje formát a způsob ukládání metadat v úložišti CRCE. Dále je zde popsána funkce modulů, která indexují API a získané údaje, nad kterými je později provedeno porovnání, ukládají do CRCE.

4.1 CRCE

- Citovat [2]
- CRCE je systém pro ukládání SW komponent, který je vyvíjený na KIV FAV
- CRCE je modulární a lze přidat pluginy pro indexaci custom dat

4.1.1 Metadata v CRCE

- Metadata v CRCE mají hierachickou strukturu: - Resource + Capability + Properties + Atributy - taky Requirements, ale ty v práci nepoužívám - Resource reprezentuje indexovanou komponentu - jednotlivé featury (indexovaná data) jsou reprezantovány stromem Capabilit - k Resource je vždy přiřazena root Capabilita - každá Capabilita má namespace, podle kterého lze určit co za konkrétní vlastnost popisuje - v případě root Capability by namespace měl být unikátní pro Resource (a resource by tedy neměl mít více root Capabilit s jedním namespace (snad?)) - detaily fetatury jsou pak uloženy v dětských capabilitách, jejich Properties a Attributes - Capability mají Attributy + Properties - Properties mají atributy - Attributes pak nesou konkrétní hodnoty (Capability a Propeties slouží pouze jako jakési kontejnery) - Lze tak modelovat různé vlastnosti indexovaného objektu (viz [2], tam je to dobře popsaný) - hierarchická struktura metadat je vhodná pro reprezentaci webových API, která jsou rovněž hierarchická



Obrázek 4.1: Reprezentace metadat v CRCE

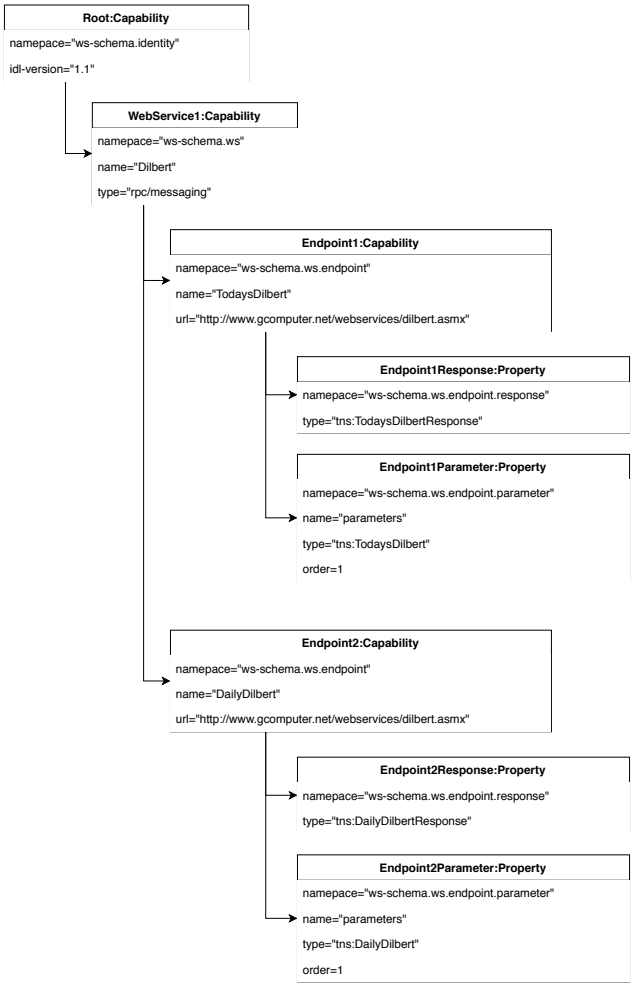
4.2 Indexování API

- různé druhy jsou jinak indexované - každý druh API indexován vlastním modulem - diplomky Pejřimovského [**?**] a Hessové [**?**] - někde by asi bylo fajn ustanovit názvosloví použité v práci: - co je API: interface přístupné skrze síť (internet) - co je web service: service popsaný WSDL, WADL, nebo Json-WSP dokumentem - co je service: Service element in WSDL - co je endpoint - WSDL: port+operation - endpoint: REST, WADL, JSON-WSP - indexované API je v CRCE uloženo jako Resource - popis API je reprezentován jako samostatná feature (1 root Capability) daného Resource - Namespacy kořenových Capabilit si definuje indexer - Service a endpoint jsou reprezentovány jako Capaibility - endpoint parametry, endpoint response, endpoint request body a endpoint request body jako Property - obecná struktura indexovaných dat na obrázku 4.2 - vlastní hodnoty pak jako Attribute - příklad metadat indexovaného API: 4.3

Obrázek 4.2: Obecná struktura indexovaného API

4.2.1 Indexování REST API

- práce: [**?**] - binární analýza JAR s implementací API - funguje na principu hledání patternů v byte kódu - indexer vytváří hierarchii metadat ve formátu root capability -> child endpoint capabilities - podpora formátů: - REST: JAX-RS, Spring Web MVC podporovány



Obrázek 4.3: Příklad indexované SOAP web service Dilbert

4.2.2 Indexování WS

- práce: Pejřimovského [**?**] - nějaký trefný obrázek indexovaných dat - konkrétní formát API detekován z buď z formátu vstupního souboru, nebo z metadata v top elementech - podle typu je pak použit daný parser - podpora formátů: - WSDL: hierarchie root capability -> web service capabilities -> child endpoint capabilities - WADL, Json-WSP: hierarchie root capability -> child endpoint capabilities - parsování souboru s popisem API, CRCE stačí i URL - indexer obsahoval drobné chyby, které jsem v rámci DP opravil - špatná indexace URL v případě WSDL (nebyla podle specifikace)

4.2.3 Limity indexování

- custom datové typy - 2 problémy - rekurzivní typy - jsou způsoby pro jejich rozvoj: [1] a ukládání - nicméně indexovací logika není implementovaná (ani v jednom ze zmíněných indexerů) - chybějící definice custom typů - v případě např REST jsou uloženy v implementaci (nemusí se jednat ani o stejnou knihovnu) a indexer k nim nemusí mít přístup - tím pádem je jméno datového typu (např. fully qualified name v případě Java třídy) jedinou informací, která je o typu dostupná

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 popsanými 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 (taky 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]]`
- lowecase 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

Algorimuts (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:

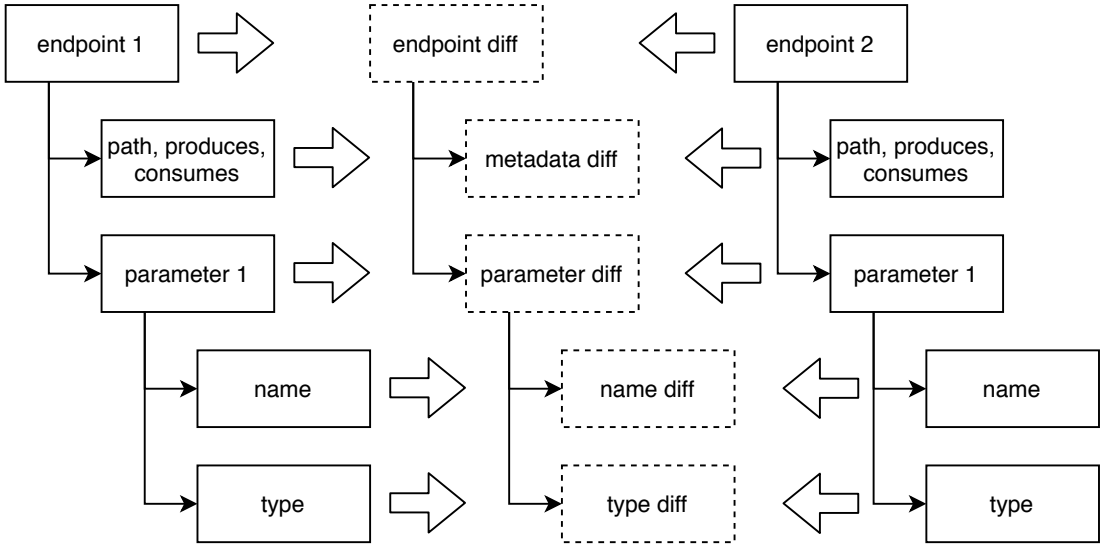
- $h \wedge !pe \wedge !o$
- $h \wedge pe \wedge !o$
- TODO

5.2 Výsledek porovnání - Diff

Popis výsledné datové struktury

- Diff, Compatibility
- vychází z [3]
- 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 daových 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

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] 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.

[3] Brada, P. – Valenta, L. Practical Verification of Component Substitutability Using Subtype Relation. s. 38 – 45, 10 2006. doi: 10.1109/EUROMICRO.2006.50.

[5]]hessova2015rest 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/.

[5]]pejrimovsky2015ws 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/.