Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

Bc. Ivan Martoš

Odporúčanie softvérových služieb s využitím kontextu

Diplomová práca

Študijný program: Softvérové inžinierstvo
Študijný odbor: 9.2.5 Softvérové inžinierstvo
Miesto vypracovania: Ústav informatiky a softvérového inžinierstva, FIIT STU, Bratislava
Vedúci práce: doc. Ing. Viera Rozinajová, PhD.
Máj, 2015

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Odporúčanie softvérových služieb s využitím kontextu

Študijný program: Softvérové inžinierstvo
Autor: Bc. Ivan Martoš
Vedúci diplomovej práce: doc. Ing. Viera Rozinajová, PhD.
Máj, 2015

Aktuálne existuje veľké množstvo rôznych dostupných softvérových služieb. Tieto softvérové služby majú rôzne vlastnosti, pričom výber správnej služby pre riešenie určitého problému je netriviálna a veľmi zložitá úloha.
Táto práca sa zameriava na problém odporúčania softvérových služieb. Odporúčaním chceme dosiahnuť vybratie práve tej služby, ktorá čo najviac spĺňa potreby používateľa. Používateľovi plánujeme odporúčať usporiadaný zoznam softvérových služieb, zoradený podľa viacerých kritérií vhodnosti služby pre splnenie používateľovej požiadavky.
Odporúčanie bude vykonané na základe sémantických opisov softvérových služieb, pričom pri odporúčaní plánujeme využívať moderný a vyvíjajúci sa prístup na základe využitia kontextu. V práci modelujeme kontext softvérovej služby a kontext riešeného problému, pričom sa zameriavame na automatické spresňovanie týchto kontextových informácií.

Slovak University of Technology Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Software services recommendation using context

Degree Course: Software Engineering
Author: Bc. Ivan Martoš
Supervisor: doc. Ing. Viera Rozinajová, PhD.
May, 2015

Nowadays a great number of different software services are available for usage. These services have different attributes and proper selection of software service for solving given problem is very difficult and non-trivial task.
This thesis focuses on recommending software services. By recommendation, we want to achieve selection of software service that fulfills the needs of the user as much as possible. We plan to recommend ordered list of software services, where services are ordered by our estimate of likelihood that software service fulfills user needs.
Recommendation will be based on semantic descriptions of software services. In recommendation we plan to use modern and still-evolving approach based on context. We plan to model context of a problem and context of a software service. We also focus on automatic clarification of contexts.

Poďakovanie

Týmto by som chcel poďakovať vedúcej svojej diplomovej práce doc. Ing. Viere Rozinajovej PhD. za jej vedenie, rady, pomoc pri vypracovaní práce a vynaložený čas.

Taktiež by som rád poďakoval svojej rodine a priateľom za podporu a povzbudzovanie pri riešení práce.

Čestné prehlásenie

Prehlasujem, že diplomovú prácu som vypracoval samostatne s využitím uvedených zdrojov a vlastných vedomostí.

V Bratislave dňa

_____

podpis autora

Obsah

1. Úvod

V súčasnom svete existuje veľké množstvo rôznych softvérových a webových služieb. Tieto služby sú poskytované rôznymi poskytovateľmi pričom rozsah ich funkcionality je veľmi veľký a rôznorodý.

Z dôvodu veľkého množstva dostupných služieb je pre používateľa často veľmi náročné vybratie práve tej služby ktorá je najvhodnejšia pre splnenie jeho požiadaviek. Existujúce systémy pre odporúčanie služieb odporúčajú služby nie priamo pre používateľa, ale skôr pre skupinu podobných používateľov. Taktiež častým problémom je ignorovanie kontextu používateľa a kontextu služby a ich podobnosti. Pokiaľ je kontext braný do úvahy, tak sa pokladá za finálny a jeho hodnota sa nemení v priebehu odporúčaní.

Táto práca sa zameriava práve na vyriešenie problému odporúčania služieb. Zvolený prístup spočíva vo vytváraní odporúčania personalizovaného pre špecifického používateľa. Navrhovaný prístup spočíva taktiež vo využití kontextovej informácie o používateľovi a službe. Keďže kontextová informácia môže byť zadaná nesprávne, prípadne isté atribúty kontextovej informácie nemusia byť špecifikované, navrhované riešenie taktiež počíta s upresňovaním tejto kontextovej informácie.

Týmto riešením plánujeme odbremeniť používateľa od nutnosti vyberania z veľkého množstva služieb pri riešení problému. Taktiež predpokladáme, že za pomoci spresňovania kontextovej informácie sa bude presnosť odporúčania zlepšovať a budú teda odporúčané vhodnejšie služby pre špecifického používateľa.

2. Architektúra Orientovaná na Služby a sémantika

Architektúra Orientovaná na Služby (Service Oriented Architecture – SOA) nie je novou technológiou na poli informatiky [3]. Má dlhú históriu, pričom táto technológia priamo vychádzala z požiadaviek a potrieb veľkých firiem, ktoré museli byť schopné rýchlo a flexibilne reagovať na zmeny trhu [2].

Pokiaľ má byť spoločnosť na poli svojho pôsobiska úspešná a konkurencie schopná, je potrebné aby bolo jej fungovanie efektívne a taktiež musí byť schopná flexibilne reagovať na neustále sa meniace požiadavky zákazníkov a trhu. Existuje viacero príkladov rôznych veľkých firiem, ktoré sa s postupným rozvojom informačných technológií dostali do problému efektívneho využitia rôznych systémov, komponentov, technológií a prostredí [1],[2],[3],[4].

2.1. Vývoj Architektúry Orientovanej na Služby

Častým problémom bolo, že jednotlivé systémy boli príliš špecializované pre danú funkcionalitu, pričom abstraktnosť riešenia nebola implementovaná. Taktiež rozhrania ktoré definovali komunikáciu neboli univerzálne, ale proprietárne a používali zastarané metódy prenosu (napríklad FTP, magnetické pásky), pričom často bola komunikácia iniciovaná operátormi a nebola automatizovaná. Takto vznikali silno previazané systémy ktoré neboli schopné dostatočne rýchlo reagovať na zmeny. Pri zmene jedného systému bolo nutné zmeniť aj ďalšie s ktorými prebiehala komunikácia, pretože z dôvodu pevne definovaných rozhraní cez ktoré by prebiehala komunikácia, zmena jednej strany priamo vyvolala zmenu druhej. Taktiež keďže existujúce systémy boli príliš špecializované na úlohy pre ktoré boli vytvorené, nebolo možné využívať ich funkčnosť pre inú úlohu z dôvodu nedostatku abstrakcie. Tým pádom bolo potrebné pre dodanie novej funkcionality buď existujúcu funkcionalitu modifikovať, prípadne vytvoriť nový komponent ktorý implementoval novú funkcionalitu [2],[3],[4].

Spoločnosti si boli samozrejme vedomé týchto problémov a začali hľadať čo najvhodnejšie spôsoby pre zefektívnenie svojej činnosti. Odpoveďou bola integrácia existujúcich systémov a distribuované počítanie. Prvotné riešenia zahŕňali využitie Remote Procedure Call (RPC) ako napríklad Common Object Request Broker Architecture (CORBA), alebo Distributed Component Object Model (DCOM), ktoré povoľovali využívanie vzdialených zdrojov pomocou komunikácie medzi komponentami ktoré boli uložené na rôznych miestach. Ako príklad tejto technológie možno demonštrovať prípad, kde sa dátová časť systémov spoločnosti nachádzala na jednom mieste a ostatné systémy ju mohli využívať nezávisle na ich vývojovom prostredí. Takto bol

dosiahnutý jednotný dátový model, pričom bola eliminovaná redundantnosť funkcionality slúžiacej pre napájanie sa na dátovú časť [1].
Problémom týchto riešení bola avšak príliš veľká technická a implementačná náročnosť. Tieto technológie boli komplikované a príliš náročné na pochopenie, pričom ich implementácia si vyžadovala značné úsilie čo malo za následok to, že viaceré spoločnosti zlyhali v úspešnej implementácii a adaptácii týchto technológií. Postupným vývojom sa ako vhodné riešenie spomenutých problémov začala javiť orientácia na služby [1].

2.2. Orientácia na služby

Orientácia na služby je dizajnová paradigma, ktorá je zložená zo špecifického súboru dizajnových princípov. Vytváranie distribuovanej logiky vychádza z princípu softvérového inžinierstva zvaného oddelenie záležitostí (anglicky „separation of concerns“). V skratke, táto teória hovorí o tom, že riešenie veľkého problému je efektívnejšie pokiaľ je problém dekomponovaný na menšie samostatne riešiteľné časti zvané záležitosti (anglicky „concerns“) [2].
Základnou výhodou pri riešení problémov týmto spôsobom je, že viacero menších podproblémov je možné riešiť samostatne, pričom riešením týchto podproblémov nenastáva odklonenie od riešenia pôvodného veľkého problému [2],[3].
Keďže jednotlivé riešené podproblémy sú samostatné a nezávislé od celkovej funkcionality, je možné ich taktiež využívať aj pri iných aplikáciách. Takto je vo veľkej miere umožnená znovupoužiteľnosť softvéru, pričom jednotlivé samostatné podproblémy (prípadne časti funkcionality), je možné označiť ako služby (anglicky „services“) (pre viac informácií o službách viď kapitolu 2.4) [2].
Aktuálne existuje viacero rozličných paradigiem pre riešenie distribuovanej logiky. To čo odlišuje paradigmu orientácie na služby je práve jej dôraz na využívanie oddelenia záležitostí a to, ako vytvára samostatné individuálne logické jednotky [2].

2.3. Definícia

V súčasnom svete neexistuje jednotná definícia SOA ktorá by bola štandardizovaná a všeobecne uznávaná, avšak jedna z hlavných osobností na poli SOA, Thomas Erl, ju definuje nasledovne:
Architektúra Orientovaná na služby je forma technologickej architektúry ktorá sa riadi zásadami orientácie na služby [1].
Napriek tomu že SOA nemá všeobecne prijatú a štandardizovanú definíciu, nasledovné princípy SOA sú všeobecne známe a uznávané:

1. **Znovu použiteľnosť** – jednotlivé komponenty SOA by mali byť dostatočne nezávislé a agilné na to, aby dokázali vykonávať svoju funkcionalitu nezávisle na prostredí. Taktiež je potrebné aby mali dostatočnú variabilitu funkcionality aby mohli splniť požiadavky zákazníka [2],[3],
2. **Efektivita vývoja** – vytváranie viacerých funkcionalít za kratší čas, alebo za vynaloženia nižších nákladov. Tento princíp vo veľkej miere závisí od znovu použiteľnosti a schopnosti vytvorenia nových aplikácií za pomoci existujúcich komponentov [3],
3. **Integrácia aplikácií a dát** – SOA ponúka možnosť zjednodušenia integrácie za pomoci univerzálneho prepojenia existujúcich systémov a dát. Taktiež odporúča existenciu referenčnej architektúry ktorá popisuje všeobecné vzory pre integráciu a taktiež aj rozdeľuje medzi biznis a integračnými službami [3],
4. **Agilnosť a Flexibilita** – nové procesy môžu byť vytvorené na základe existujúcich skupín komponentov,
5. **Autonómnosť** – tento princíp vyžaduje aby jednotlivé komponenty SOA boli nezávislé a samo-udržiavacie tak veľmi ako to je len možné, pričom schopnosť kontrolovať logiku ktorá im zodpovedá by bola zachovaná [1],
6. **Založenie na otvorených štandardoch** – výmena dát je riadená otvorenými štandardami. Keď je správa odoslaná z jedného komponentu do druhého, je transportovaná pomocou skupiny protokolov ktoré sú globálne štandardizované a akceptované [1],
7. **Podpora rozdielnosti prostredí** – SOA predstavuje architektúru na vymieňanie správ a abstrahuje od implementačných detailov jednotlivých komponentov, pričom priamo podporuje využívanie rozdielnych technológií na rozdielne úlohy pre ktoré boli tieto technológie vyvinuté. Takto umožňuje organizáciám voľby technológií ktoré sú pre ne najvyhovujúcejšie [1],[2],
8. **Rozdelenie funkcionality do vrstiev** – jednou z charakteristík ktoré sa prirodzene vyvinuli za pomoci orientácie na služby je princíp abstraktnosti. Abstrakcia sa môže zamerať na vytvorenie biznis a aplikačnej logiky, ktoré môžu byť oddelené od seba do jednotlivých vrstiev v ktorých sa nachádzajú príslušné komponenty zodpovedné za danú funkcionalitu [1],[2],
9. **Jemná previazanosť** – za pomoci jemnej previazanosti jednotlivých komponentov je možné vytvorenie nezávislých častí funkcionality ktoré oddeľujú logiku systému. Taktiež je týmto podporovaný princíp znovu použiteľnosti [1],[2],
10. **Neustály vývoj** – SOA nie je pevne a nemenne zadefinovaná architektúra, ale neustále sa mení a vyvíja na základe rôznych a meniacich sa požiadaviek biznis sféry [3].

2.4. Softvérové služby

Softvérové služby (skrátene služby) sú samostatné jednotky logiky ktoré boli vytvorené na základe princípov orientácie na služby. Služba je abstraktný pojem, ktorý sa nezakladá na špecifickej technológií, prípadne vývojovom prostredí [2].

2.4.1. Vlastnosti softvérových služieb

Napriek tomu že služby nie sú orientované na špecifickú platformu, pri vytváraní služieb je potrebné dodržať nasledovné princípy [2]:

1. **Štandardizovaný kontrakt** – služby by mala ponúkať svoju funkcionalitu za pomoci štandardizovaného kontraktu cez ktorý je možné zistiť rozhranie cez ktoré sa dá so službou komunikovať [2],
2. **Jemná previazanosť** – previazanosť hovorí o spojení viacerých vecí. Služba by mala byť čo najmenej prepojená s inými službami. Takto je možné dosiahnuť čo najvyššiu možnú mieru interoperability. Taktiež tento princíp poukazuje na samostatný dizajn, vývoj logiky a implementáciu tak, aby stále dokázala garantovať svoju funkčnosť [2],
3. **Abstrakcia** – tento princíp hovorí o tom, že služba by mala skrývať detaily svojej základnej funkčnosti najviac ako je možné. Tým je umožnené zachovanie jemnej previazanosti, pričom tento princíp taktiež hrá dôležitú rolu pri kompozíciách služieb [2],
4. **Znovupoužiteľnosť** – tento princíp je jedným z najdôležitejších pri orientácii na služby a službách samotných až v takej miere, že je typickou súčasťou analýzového a návrhového procesu. Princíp vychádza z princípov jemnej previazanosti a abstrakcie, pričom hovorí o tom, že služba by nemala byť zameraná priamo na špecifické podmienky, ale má byť použiteľná v rôznych systémoch [2],
5. **Autonómnosť** – logika služby musí mať kontrolu nad jej prostredím a zdrojmi. Služby musia byť taktiež nezávislé od iných služieb, pričom musia byť schopné fungovať správne samostatne [2],[3],
6. **Bezstavovosť** – keďže uchovávanie stavu služby môže ohroziť jej dostupnosť pre iných konzumentov, je dôležité aby boli služby vytvárané ako bezstavové. Avšak pokiaľ by mali mať stav, tak len vtedy pokiaľ to je nevyhnutné a nedá sa tomu predísť [2],
7. **Objaviteľnosť** – keďže služby by mali byť dostupné pre ich konzumentov, predpokladom pre ich využitie je ich objavenie. Tento princíp hovorí o tom, že služba by mala byť objaviteľná pre použitie [2],
8. **Vytváranie kompozícií** – keďže sofistikovanosť systémov orientovanách na služby je čoraz väčšia, častým riešením týchto problémov je spájanie existujúcich služieb do kompozícií. Každá služba by teda mala byť schopná zúčastňovať sa v kompozícií [2].

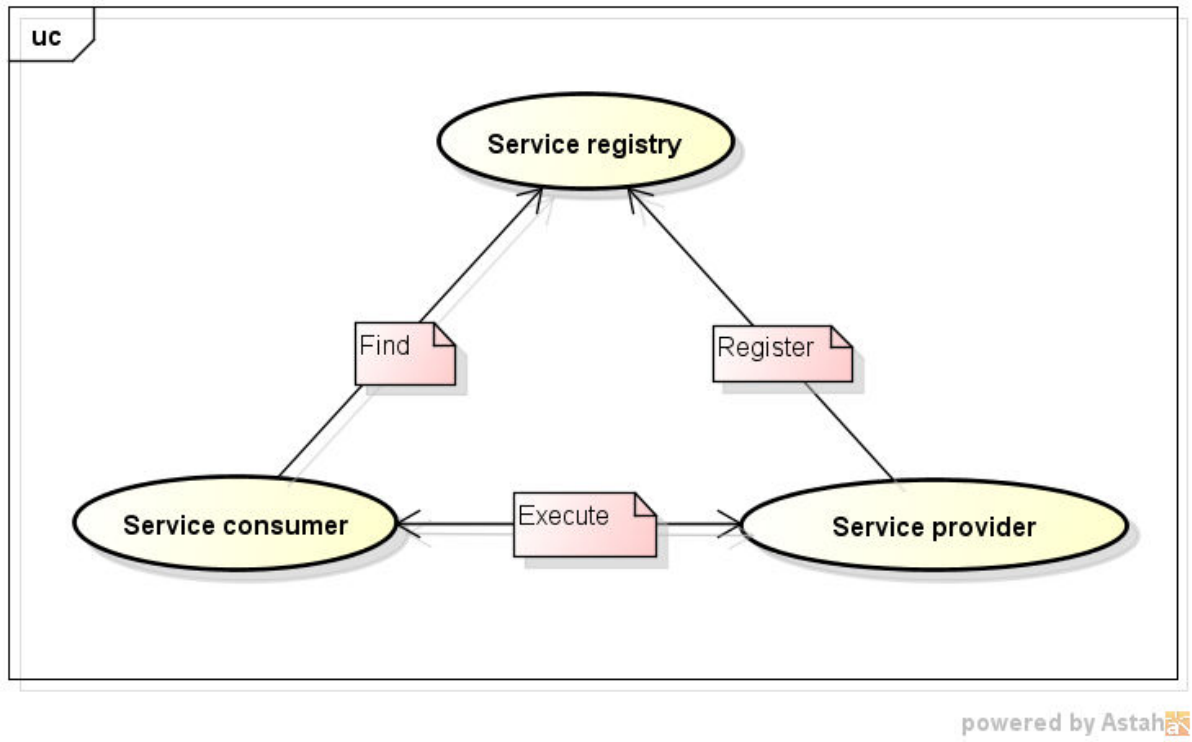
2.4.2. Roly v softvérových službách

Pri softvérových službách a SOA sa role dajú rozdeliť do troch skupín – poskytovateľ služieb, konzument služby a register služieb.

Poskytovateľ služby je zodpovedný za poskytovanie služby transparentným spôsobom. Službu poskytuje konzumentom, pričom jeho zodpovednosťou je taktiež zachovanie funkčnosti služby [5].

Konzument služby využíva službu ktorú poskytuje poskytovateľ služby v rámci podnikových procesov [5].

Register služieb umožňuje konzumentovi služby vyhľadať služby podľa daných kritérií. Taktiež umožňuje poskytovateľovi publikovať svoju službu a tak ju sprístupniť na používanie [5].



Obrázok 1 Diagram znázorňujúci role v SOA a softvérových službách [5].

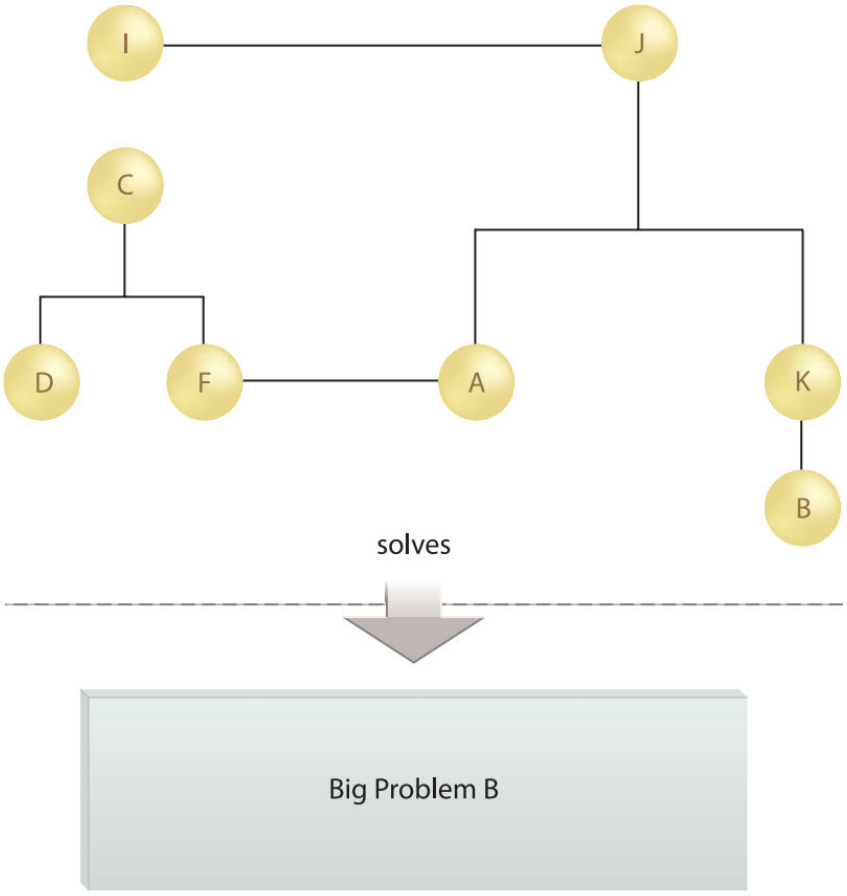
2.4.3. Kompozícia služieb

Ako bolo v predchádzajúcej kapitole povedané, jednou zo základných vlastností služieb je ich znovupoužiteľnosť. Táto vlastnosť sa využíva pri ich kombinácií do celkov ktoré tvoria aplikácie pre uspokojovanie potrieb biznisu. Kompozícia služieb je prístup, ktorým je táto kombinácia dosiahnutá [3].

Služby vznikajú dekompozíciou veľkej časti funkcionality do menších samostatných častí. Služby sú dekomponované z dôvodu jednoduchšej implementácie a taktiež aj preto, lebo existuje

predpoklad pre znovupoužitie dekomponovanej funkcionality aj v iných systémoch ako tých, v ktorých pre ktoré bola služba vytvorená. Keď je funkcionalita dekomponovaná, je možné ju spätne rekomponovať, prípadne využiť existujúce služby pre vytvorenie novej funkcionality. Služby sú bezstavové, autonómne, objaviteľné a znovupoužiteľné jednotky funkcionality. Na základe týchto charakteristík, vieme spájať samostatné služby do veľkých celkov ktoré slúžia na riešenie rozsiahleho, prípadne komplikovaného problému, ktorý by nebolo možné vyriešiť za použitia jednej služby [2].

Súbor komponovaných služieb sa môže nazývať služba, alebo presnejšie aj „Kompozícia služieb“, anglicky „composite service“. Služba ktorá sa zúčastňuje kompozície je označená ako „Komponovaná služba“ (anglicky „component service“). Taktiež je dôležité poznamenať, že aj jednotlivé kompozície služieb sa môžu zúčastňovať ďalších rozsiahlejších kompozícií ako komponované služby.



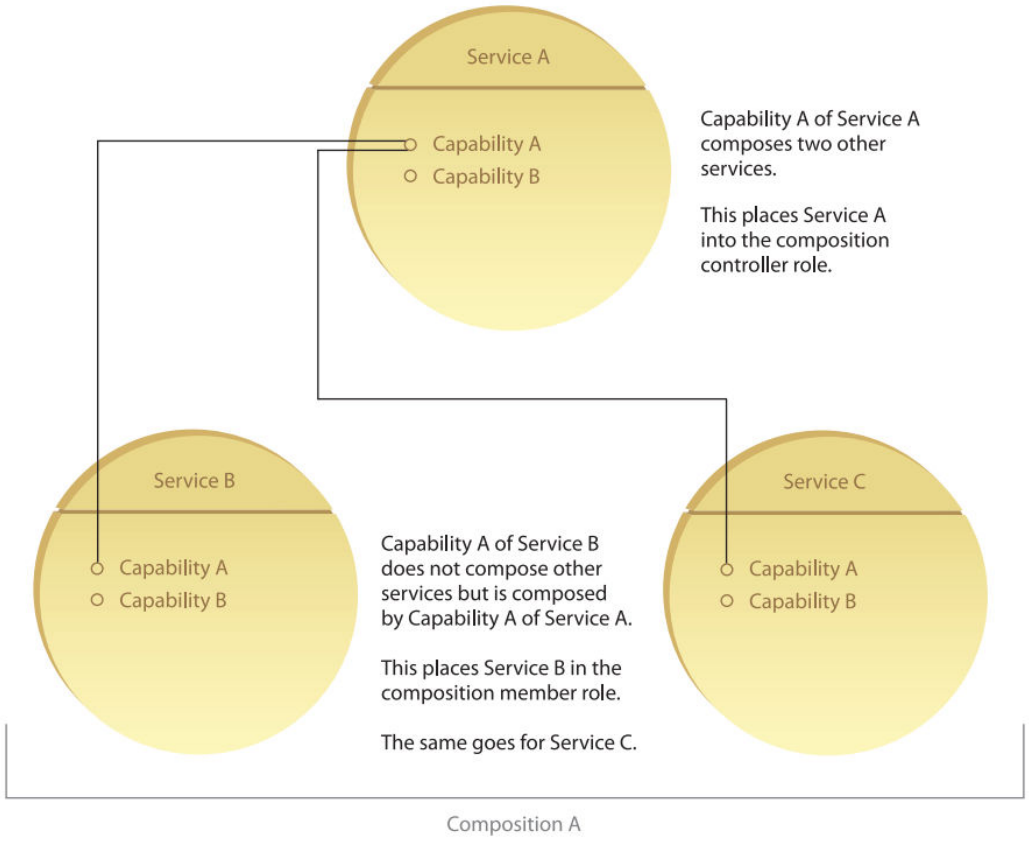
Obrázok 2 Ukážka kompozície služieb A-K pre vyriešenie veľkého problému B [2].

2.4.4. Role v kompozícií služieb

Pri zúčastňovaní sa kompozícií, môžu služby zastávať rozličné role v závislosti od toho, ako sú v kompozícií umiestnené a aký je ich účel. Úlohy jednotlivých rôl v kompozícií sú demonštrované na obrázku 3. V kompozícií služieb sa vyskytujú nasledovné role [2]:

Composition controller – vo voľnom preklade znamená „ten kto riadi kompozíciu“, pričom v tejto práci sa bude používať anglický termín. V tejto roli je služba pokiaľ sa nachádza na vrchole hierarchie kompozície. Táto služba obsahuje logiku ktorá riadi postup vykonávania jednotlivých služieb v kompozícií [2].

Composition member – vo voľnom preklade znamená „člen kompozície“, pričom v tejto práci sa bude používať anglický termín. Táto rola reprezentuje službu ktorá je komponovaná inou službou (jej funkcionalita je vyvolaná inou službou). Táto rola môže taktiež komponovať ďalšie služby. Keďže táto rola vyvoláva iné služby, je možné ju z hľadiska účasti v kompozícií považovať za typ composition controller-a. Z toho dôvodu je taktiež označovaná aj ako „sub-controller“ [2].



Obrázok 3 Demonštrácia úloh rolí v kompozícií služieb [2].

2.4.5. Typy služieb v kompozíciách

V kompozíciách služby vystupujú v rôznych roliach, avšak každá služba má v kompozícií aj svoj typ. Tento typ sa vyhodnocuje z hľadiska nutnosti vykonania jej funkcionality v danej kompozícií. Rozdeľujeme dva typy [6]:

Mandatory Service – vo voľnom preklade znamená „nutná služba“ , pričom v tejto práci sa bude používať anglický termín. Tento typ zodpovedá komponovanej službe ktorej funkcionalita musí byť v kompozícií služieb bezpodmienečne vykonaná [6]..

Optional Service - vo voľnom preklade znamená „nepovinná služba“ , pričom v tejto práci sa bude používať anglický termín. Keďže sa predpokladá, že komponované služby sú roztrúsené po sieti, spoľahlivosť vykonania komponovanej služby ovplyvňuje spoľahlivosť celej kompozície. Tento typ zodpovedá komponovanej službe, pri ktorej nie je nutné aby jej funkcionalita bol v kompozícií služieb vždy vykonaná, avšak pokiaľ to možné je, odporúča sa jej funkcionalitu vykonať [6].

2.4.6. Typy kompozícií

Kompozície služieb môžu byť rôzne členené a klasifikované z rôznych hľadísk. Z pohľadu tejto práce je ale dôležité to, kedy je kompozícia vytvorená. Na základe tohto pohľadu delíme kompozície služieb do dvoch typov [6]:

Proaktívna (Statická) kompozícia – pri tomto type je kompozícia vytváraná počas fázy návrhu, keď je vytváraná architektúra a dizajn plánovaného softvéru. Kompozícia je predkompilovaná a je vyvolaná po požiadavke klienta. Takáto kompozícia je funkčná pokým sa jednotlivé komponované služby nezmenia [6],[7].

Reaktívna (Dynamická) kompozícia – tento typ kompozície je vytvorený dynamicky počas behu programu. Oproti statickej kompozícii je vytvorenie náročnejšie, avšak má niekoľko výhod. Napríklad takto vytvorená kompozícia môže byť priamo vytvorená pre riešenie problémov klienta, pričom statická kompozícia používa len akúsi šablónu ktorá sa nemení [6].

2.5. Webové služby

V histórií informačných technológií bola vždy veľká diverzita systémov, pričom jednotlivé prostredia nie len že medzi sebou nekomunikovali, ale dokonca si navzájom konkurovali (ako príklad J2EE a .NET). Taktiež neexistoval žiadny komunikačný kanál cez ktorý by bola možná komunikácia medzi takýmito systémami (viac informácií v kapitole číslo 2.1).

Ako možné riešenie sa javilo využitie Internetu, ktorý sa začal rozvíjať na základe projektu ARPANET v roku 1969. Internet sa postupne rozširoval, avšak problémom bolo to, že neexistoval všeobecne uznávaný protokol pre komunikáciu medzi uzlami Internetu, avšak postupným vývojom a adaptáciou sa ako riešením tohto problému naskytol protokol TCP/IP,

ktorý už v čase masívneho rozvoja Internetu v roku 1994 bol všeobecne uznávaným štandardom. Týmto bola vyriešená otázka prepojenia uzlov v Internete, avšak stále nebol definovaný protokol pre výmenu správ. Existovalo viacero rôznych protokolov, ale ako najvhodnejší sa javil protokol HTTP, ktorý bol v roku 1997 štandardizovaný, pričom tento protokol sa stal všeobecne uznávaným štandardom pre výmenu správ.
Ďalším krokom bola potreba štandardu pre formát a obaľovanie správ. Riešením sa ukázal platformovo nezávislý meta-jazyk XML, ktorý bol schopný popisovať nie len posielané dáta, ale aj ich formát [8].

2.5.1. Definícia

W3C definuje webovú službu ako „Softvérový systém vytvorený za účelom podpory spolupráce strojov a interakcie cez sieť. Obsahuje rozhranie v strojovo-čitateľnom formáte. Iné systémy interagujú s webovou službou v zmysle predpísanom jej popisom za pomoci SOAP správ, zvyčajne s využitím HTTP a XML serializácie v spojení s inými sieťovo-prepojenými štandardami“ [9].
Za pomoci technológií a protokolov TCP/IP, HTTP a XML bolo možné vytvoriť technológiu pre implementáciu a realizáciu služieb cez Internet. S využitím tejto technológie je možné prepojiť rôzne systémy vytvorené na rôznych platformách tak, že všetky vlastnosti ktoré by mala služba mať sú zachované.
Implementáciu služieb za pomoci SOAP správ vytvorených a definovaných v jazyku XML za pomoci protokolu HTTP a TCP/IP nazývame webové služby.

2.5.2. SOAP

Simple Object Access Protocol (SOAP) býva často označovaný ako protokol pre vyvolávanie Remote Procedure Call (RPC) cez HTTP. Toto označenie ale nie je presné. SOAP je XML formát správ pre výmenu štruktúrovaných dát. Môže byť použitý pre RPC v klient-server aplikáciách, preposielanie XML dokumentov a implementáciu systémov na výmenu správ v publisher-subscriber modeloch (vo voľnom preklade „vydavateľ-odoberateľ model“. V tejto práci bude používaný anglický termín) [10].
Bez nadstavieb a rozšírení je SOAP bezstavová a jednosmerná paradigma pre výmenu správ. Je taktiež nezávislá od platformy a operačného systému. Je možné ju aj ďalej rozširovať a prispôsobovať si ju pre vlastné potreby a použitie [11].
Formát SOAP správy je XML dokument ktorý bol všeobecne štandardizovaný a uznaný. Celý dokument je zaobalený do tagu envelope (obálka), pričom jej obsah je rozdelený do dvoch častí – header (hlavička) a body (telo). V rámci tagu body sa nachádza samotná správa (zadefinovaná v presnom formáte) a v rámci tagu header je obsiahnutých viacero pomocných informácií ako napríklad inštrukcie pre spracovanie a podobne [11].

SOAP presne definuje akú štruktúru a formát majú mať jednotlivé položky vnútri body tagu – ako napríklad metódy, ich návratové hodnoty a parametre.
Keďže SOAP je XML správa, pre dynamickú funkcionalitu je potrebné vytvoriť zo SOAP správy vykonateľný kód. Komponent ktorý je za túto funkcionalitu zodpovedný sa nazýva „proxy komponent“. Tento komponent analyzuje SOAP správu v troch krokoch:

1. Deserializácia XML (SOAP) správy do natívneho kódu prostredia,
2. Vyvolanie kódu,
3. Serializácia odpovede do XML (SOAP) správy a jej odoslanie tomu, kto inicioval vyvolanie služby.

Keďže SOAP je platformovo nezávislé, je potrebné aby každá platforma mala vlastnú implementáciu SOAP proxy komponentu. Takto za pomoci jednotného formátu správy a rozličných analyzujúcich komponentov na strane prostredí, je možné prepájať rôzne systémy na základe posielania štandardizovaných presne formátovaných správ medzi nimi [8].

2.5.3. WSDL

Aby bolo možné webovú službu využívať, je potrebné vedieť formát jej správ a ako takúto službu využívať. Webové služby bývajú štandardne popísané za pomoci Web Service Description Language (WSDL). WSDL je špecifikácia ktorá definuje ako opísať webovú službu. Je to XML súbor ktorý popisuje port, názvy metód, argumenty a dátové typy webovej služby. Keďže WSDL je vo formáte XML, je možné WSDL súbor čítať strojom, ale je stále čitateľný aj pre človeka [8],[11].
WSDL popisuje štyri hlavné časti každej webovej služby [8]:

1. Informácie o rozhraniach verejne prístupných metód,
2. Informácie o dátových typoch všetkých požiadaviek a odpovedí správ webovej služby,
3. Informácie o využitom transportnom protokole,
4. Adresné informácie pre lokalizáciu špecifikovanej služby.

WSDL reprezentuje kontrakt medzi konzumentom služby a poskytovateľom služby. Za pomoci WSDL môže konzument zistiť aké verejné metódy webová služba ponúka a vyvolať ich [8].
Keďže WSDL je vo formáte XML ktorý je strojovo čitateľný, je možné automaticky vytvárať triedy zodpovedné za komunikáciu medzi webovými službami a to tak, že z WSDL sa za pomoci nástroja vygeneruje trieda, ktorá vie pracovať s SOAP webovou službou ktorú toto WSDL popisuje [8].

2.6. REST služby

Idea implementácie služieb cez Internet sa veľmi rýchlo ujala a bola využívaná vo veľkej miere veľmi rýchlo od jej počiatkov. Zaujímavosťou ale bolo to, že štandardy ktoré túto webové služby

implementovali, boli definované bez ohľadu na praktické využitie služby, dokonca niektoré technológie boli štandardizované skôr ako reálne implementované. To malo za následok skôr zmätok z príliš komplexného štandardu, oproti organizovanému poriadku ktorý štandardizácia zväčša dosiahla [12].

Reakciou bol alternatívny štýl webových služieb, založených na princípoch Internetu. Začali vznikať REST služby. Tieto služby sa pokúšali využívať bezpečné a osvedčené princípy Internetu pre svoju funkcionalitu.

REST služby sú založené na zdrojoch (anglicky „resources“). Zdroj v zmysle REST môžeme definovať ako čokoľvek čo je dostatočne dôležité na to, aby sa na to dalo odkazovať ako na celok. Zvyčajne to býva informácia s určitou výpovednou hodnotou. Napríklad riadok v databáze, výsledok algoritmu a podobne. Zdroj sa ale stáva zdrojom až vtedy, keď má aspoň jedno URI [13].

Základnými dvoma princípmi REST služieb je ich adresovateľnosť a bezstavovosť. Adresovateľnosť je dosiahnutá tým, že zdroje ktoré poskytuje REST služba, sú dosiahnuteľné pomocou ich URI. Keďže REST služby sú založené na princípoch Internetu, zväčša sa využíva HTTP ako transportný protokol (ale nie je to nutnosťou, pri REST je možné využiť aj iné protokoly). HTTP poskytuje osem metód (najpoužívanejšie sú „GET“ a “POST“) pre prístup ku URI. REST služby tieto metódy využívajú pre adresovateľnosť zdrojov. Každá REST služba je taktiež bezstavová. Každý HTTP request (v preklade požiadavka, v tejto práci sa bude využívať anglický termín) je nezávislý a izolovaný od ostatných [13].

REST služba sprístupňuje zdroje pomocou ich URI ktoré predstavuje ich adresu. Avšak REST poskytuje taktiež funkcionalitu odovzdávania parametrov pre daný zdroj. Tieto parametre sa odovzdávajú cez adresu zdroja. Každá adresa je zložená z dvoch častí – URI zdroja a parametre (taktiež nazývané aj query (v preklade „dopyt“, v tejto práci bude použitý anglický termín). V rámci tela requestu aj odpovede môže byť čistý text, serializovaný XML objekt, pričom aktuálne častým javom je aj JSON objekt [12],[13].

Rest služby sa taktiež nazývajú aj RESTful služby (RESTful services), pričom to, či ich možno označovať ako webové služby je námetom na mnohom diskusií. V tejto práci budeme uvažovať že REST služby sú typom webových služieb. Je to z toho dôvodu, že sú založené na princípoch Internetu a sú implementáciou služieb.

Výhodou REST služieb oproti tradičným webovým službám je ich jednoduchosť a flexibilnosť, avšak ich nevýhodou je, že nedokážu poskytovať toľko možností a funkcionality ako tradičné webové služby [12].

Práve pre väčšie možnosti využitia je táto práca zameraná primárne na tradičné webové služby (podrobnejšie spomínané v kapitole 2.5).

2.7. Sémantické dáta

S rozvojom Internetu sa jedným z prvých štandardov na prenos dát stal formát XML. XML poskytovalo strojovo spracovateľný a ľuďmi čitateľný formát pre výmenu údajov. Problémom avšak bolo, že počítače neboli schopné porozumieť tomu, aká informácia sa v XML nachádza. Riešením pre strojovo čitateľný formát dát sa ukázalo práve RDF a metadáta.

2.7.1. Resource Description Framework (RDF)

RDF je štandardom pre popis sémantických dát, pričom je založené na XML a XML Schema. Skladá sa z trojice [8] :

1. **Subject** (v preklade „Predmet“, v tejto práci budeme využívať anglický termín). Predstavuje zdroj ktorý sa popisuje. RDF zdroj je unikátne identifikovateľný pomocou URI.
2. **Predicate** (v preklade „Predikát“, v tejto práci budeme využívať anglický termín). Predstavuje vlastnosť subject-u. Taktiež každá vlastnosť je unikátne identifikovateľná pomocou URI.
3. **Object** (v preklade „Objekt“, v tejto práci budeme využívať anglický termín). Predstavuje hodnotu vlastnosti subject-u. Môže nadobúdať akúkoľvek hodnotu z RDF dátových typov, vrátane iného RDF. RDF poskytuje veľké množstvo rôznych dátových typov.

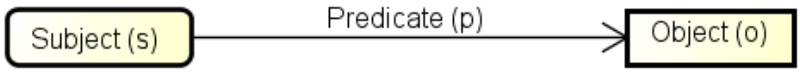
Informácia uložená v RDF sa dá preložiť ako:

Vec **[subject]** má vlastnosť **[predicate]** s hodnotou **[object]**

Touto jednoduchou konštrukciou je možné vyjadrovať veľmi veľké množstvo znalostí. Pre RDF sú zadefinované nasledovné pravidlá [8]:

1. Každá RDF trojica sa skladá z subject, predicate a object.
2. Každá RDF trojica predstavuje kompletný a unikátny fakt.
3. Každá RDF trojica sa môže spojiť s inými RDF trojicami, pričom si zachová vlastný význam.

Taktiež každé RDF môže byť reprezentované buď ako dokument, alebo ako graf s orientovanými hranami. Výhodou grafov je to, že je v nich ľahké čítať a vyhľadávať.



Obrázok 4 Grafová reprezentácia RDF trojice [8].

2.7.2. RDF schéma, ontológie a OWL

RDF poskytuje jednoduchý, ale veľmi silný nástroj pre popisovanie znalostí. Avšak zmysel týchto informácií je popísaný pomocou **RDF Schema** (RDFS). RDFS je založené na RDF, pričom obsahuje vlastné špeciálne konštrukcie, pomocou ktorých je možné rozširovať a spájať informácie v RDF. RDFS spája informácie v RDF do tried a kolekcií, pričom poskytuje možnosti viacnásobného dedenia medzi týmito triedami. Vďaka tomu je možné definovať rôzne triedy znalostí a ďalej s nimi pracovať. RDFS teda slúži na definovanie hierarchií [8].

Spájaním jednotlivých znalostí v RDF spolu s triedami a vzťahmi definovanými v RDFS vznikajú rôzne súbory prepojených informácií ktoré sa nazývajú **ontológie**. Ontológie sa využívajú na popísanie konceptov a vzťahov medzi nimi. Rozdiel medzi ontológiou a XML Schema je ten, že ontológia predstavuje reprezentáciu znalostí, pričom XML Schema formát správ [8].

Najrozšírenejším jazykom pre prácu s ontológiami je **Web Ontology Language** (OWL). OWL je štandardom pre prácu s ontológiami. Tento jazyk je postavený na RDFS, pričom definuje vlastné vzťahy a vlastnosti s ktorými je možné pracovať a využívať ich. OWL definuje napríklad vzťahy pre porovnávanie, typ parametrov (napríklad tranzitívne a funkcionálne), obmedzenia pre hodnoty a ďalšie vlastnosti s ktorými je možné ďalej efektívne pracovať [8].

2.7.3. Nástroje pre prácu so sémantickými dátami

S rozšírením sémantického webu a sémantického popisu informácií vo formáte RDF, vzniká veľké množstvo dát s ktorými je veľmi náročné pracovať, vyhľadávať v nich a filtrovať. Je potrebné aby existovali nástroje s ktorými je možné efektívne a jednoducho využívať informácie uložené v RDF.

2.7.4. SPARQL

Keďže RDF trojice je možné reprezentovať ako graf s orientovanými hranami, je možné v takom grafe efektívne vyhľadávať dáta. Jazykom ktorú to umožňuje je SPARQL. Je to štandard pre vyhľadávanie v sémantických dátach. SPARQL je podobný iným SQL jazykom v tom, že dokáže vykonávať štruktúrované dopyty, pričom je vytvorený pre prácu so sémantickými dátami. Tento jazyk ponúka veľké množstvo konštrukcií pomocou ktorých sa dá veľmi efektívne vyhľadávať nad sémantickými dátami vo formáte RDF. Výhodou je, že samotné vyhľadávanie nemusí byť vykonané len priamo nad lokálnymi dátami, ale SPARQL dokáže vyhľadávať aj vo vzdialených zdrojoch [24].

2.7.5. SPARQL-DL

Napriek tomu že SPARQL dokáže vykonávať dopyty nad sémantickými dátami vo formáte RDF, nedokáže pri vykonávaní týchto dopytov porozumieť hierarchií tried RDFS a konštrukciám OWL. Z toho dôvodu bolo vyvinuté SPARQL-DL ktoré dokáže spracovať informácie uchované v ontológiách, spolu s hierarchiami dedenia.

SPARQL-DL je vytvorené pre prácu s ontológiami vo formáte OWL, pričom funkčne predstavuje rozdielnu podmnožinu SPARQL. Taktiež ako iné SQL jazyky, dokáže vykonávať štruktúrované dopyty nad dátami, pričom dátovú množinu predstavujú ontológie. Tento jazyk obsahuje špeciálne konštrukcie, na základe ktorých je možné vyhľadávať vo vzťahoch definovaných v OWL.

SPARQL-DL odporúča W3C pre prácu s OWL, avšak nie je štandardom. Tento nástroj je vyvíjaný spoločnosťou Derivo [25].

2.8. Sémantické webové služby

Internet bol pôvodne vytvorený za účelom výmeny a zobrazovania informácií. Informácie boli zväčša zobrazované v HTML stránkach, ktoré avšak umožňujú len zobrazovanie informácií bez globálnej schémy. Človek vie prezerať tieto stránky a zistiť ich obsah, ale neexistuje stroj ktorý by bol schopný porozumieť obsahu týchto stránok a spracovávať ho. Existujú rôzne programy ktoré porozumejú určitej časti, ale tie sú priamo vytvárané pre určitú oblasť alebo doménu (prípadne časť internetového systému), ale neexistuje stroj ktorý by to bol schopný robiť po globálnej stránke [8].

Sémantický web sa pokúša riešiť tento problém. Sémantický web je nadstavbou existujúceho webu. Pokúša sa vniesť štruktúru do existujúceho webu, kde aj softvér bude vedieť porozumieť jeho obsahu a vďaka tomu lepšie a presnejšie vykonávať rôzne úlohy. Je možné povedať, že sémantický web sa stane abstraktnou reprezentáciou dát na webe. Štruktúra dát by bola reprezentovaná ontológiou. Ontológiu je možné si predstaviť ako nástroj pre umožnenie prístupu ku sémantickým dátam a ich spracovaniu. Aktuálne existuje viacero jazykov pre vytváranie ontológií, ale najpoužívanejším z nich je Web Ontology Language (OWL), ktorý sa vyvinul spojením úspešného projektu DARPA Agent Markup Language, ktorý bol jedným z prvých jazykov pre popis sémantického webu, s Ontology Infrastructure for the Semantic Web, ktorý predstavoval metódu pre popis ontológie sémantického webu [8],[14].

2.8.1. Prečo sémantické webové služby

Tak ako je sémantický web nadstavbou nad obyčajným webom, tak sú aj sémantické webové služby nadstavbou nad webovými službami. WSDL aktuálne poskytuje syntaktický popis služieb – metódy, počet a ich typ parametrov, návratové hodnoty a podobne. Avšak WSDL poskytuje

minimálny, pokiaľ vôbec nejaký sémantický opis webovej služby. Pokiaľ by existoval sémantický opis webových služieb, softvér ktorý by s týmito službami pracoval, by ich vedel efektívnejšie využívať (pretože by si bol vedomý ich funkcionality). Ďalším veľkým prínosom by bolo efektívnejšie vytváranie kompozícií. Keďže by boli dostupné sémantické opisy služieb, kompozícia by mohla byť automatizovaná. Softvér vytvárajúci kompozíciu by si bol vedomý cieľu ktorý sa pri kompozícií pokúša dosiahnuť a keďže by vedel aj sémantický opis funkcionality jednotlivých služieb ktoré má k dispozícií, vedel by vytvoriť kompozíciu oveľa presnejšie ako bez využitia sémantického opisu [14].

Sémantický web dosahuje vytvorenie sémantických vzťahov na webe pomocou ontológie, najčastejšie pomocou jazyka OWL. Sémantické webové služby potrebujú taktiež spôsob na základe ktorého sa dajú sémantické vzťahy služieb modelovať a opisovať a na dosiahnutie tohto spôsobu sa využíva taktiež ontológia. Existujú dva hlavné jazyky ktoré umožňujú vytváranie ontológií nad webovými službami. Prvou je Web Ontology Language for Services (OWL-S) a druhou Web Service Modeling Ontology (WSMO). Ontológia sémantickej služby má za úlohu poskytovať službe ktorú opisuje tri automatizované služby [8]:

1. Vyhľadateľnosť služby,
2. Vykonanie služby,
3. Komponovanie služby.

2.8.2. OWL-S

Cieľom OWL-S je vytvorenie ontológie pre webovú službu tak, aby služba bola viacej prístupná pre stroje v zmysle vykonania nasledovných úloh automaticky:

Úloha	Popis
Objavenie	Vyhľadanie webovej služby.
Vyvolanie	Vykonanie služby konzumentom, prípadne inou službou.
Znovupoužiteľnosť	Minimalizovanie prekážok znovupoužiteľnosti a automatické vkladanie parametrov správ.
Kompozícia	Vytváranie nových služieb automatickým výberom a kompozíciou existujúcich.
Overenie	Overenie vlastností služby.
Monitorovanie vykonávania	Sledovanie vykonávania úlohy v kompozícií a identifikácia prípadov kedy vykonanie kompozície zlyhalo.

Tabuľka 1 Zoznam úloh vykonateľných nad sémantickou webovou službou za pomoci OWL-S [8].

OWL-S definuje súbor tried a vlastností pre popísanie služby. Na najvyššom stupni v ontológií sa nachádza trieda Service. Ontológia poskytuje tri základné typy znalostí o službe [8]:

1. Trieda Service prezentuje triedu ServiceProfile (v preklade Profil Služby, v práci sa bude využívať anglický termín) – čo služba poskytuje a čo vyžaduje od konzumentov,
2. Trieda Service je popísaná triedou ServiceModel (v preklade Model Služby, v práci sa bude využívať anglický termín) – ako služba funguje,
3. Trieda Service podporuje triedu ServiceGrounding (v preklade Základy Služby, v práci sa bude využívať anglický termín) – ako ku službe pristupovať.

ServiceProfile

Táto trieda poskytuje základné informácie o službe na základe ktorých sa môže konzument rozhodnúť či služba približne spĺňa jeho požiadavky ohľadom funkcionality, zabezpečenia, lokalizácie a požiadaviek na kvalitu. Trieda obsahuje pre človeka a stroj čitateľný popis služby, špecifikáciu funkcionality služby a parametre potrebné pre jej funkcionalitu [8].

ServiceModel

Trieda umožňuje konzumentom vykonanie hĺbkovej analýzy na zistenie, či služba spĺňa jeho požiadavky. Taktiež umožňuje komponovanie popisu služieb na vykonanie určitej úlohy, koordinovanie aktivít viacerých konzumentov a monitorovanie vykonávania služby [8].

Detailný pohľad na službu je možné vnímať ako pohľad na proces. Tento proces sa modeluje triedou ProcessModel (v preklade Procesný Model, v tejto práci sa bude využívať anglický termín), ktorá je podtriedou ServiceModel-u. Tento ProcessModel sa skladá z dvoch častí [8].

1. Ontológia procesu ktorá popisuje službu v zmysle jej vstupov, výstupov, predpokladov, efektov a podprocesových komponentov. Obsahuje informácie potrebné pre vykonanie služby,
2. Ontológia kontroly procesu ktorá popisuje každý stav každého procesu služby vrátane prvotnej aktivácie, vykonania a ukončenia. Táto ontológia je zodpovedná za monitoring vykonávania procesu služby.

ServiceGrounding

Táto trieda špecifikuje ako pristupovať ku službe – aký protokol použiť, formát správ a taktiež aj poskytuje informácie o serializácií, prenose a adresovaní správ.

V spojení SOAP, jej syntaktického opisu vo forme WSDL a sémantického opisu vo forme ontológie poskytovanej vďaka OWL-S, je možné automaticky vyvolávať, vyhľadávať a komponovať služby aj pomocou iných služieb alebo programových komponentov.

3. Odporúčanie služieb

3.1. Úvod

S čoraz väčšou popularitou a zväčšujúcou mierou využitia servisne orientovanej architektúry vzniká veľké množstvo rôznych webových služieb. Tieto služby majú rozdielne vlastnosti, sú vytvorené pre uspokojenie rôznych potrieb a boli vytvorené rôznymi poskytovateľmi. Taktiež vzniká veľké množstvo podobných služieb ktoré riešia určitý problém, ale s rôznym prístupom, iným spôsobom riešenia, prípadne sa na problém pozerajú inak ako iné služby. Ďalej môžu existovať webové služby ktoré sú poskytované rôznymi poskytovateľmi, boli vyvinuté rozdielnymi vývojármi, ale spojením ich funkcionality do kompozície môže vzniknúť práve tá služba, ktorú konzument služby potrebuje.
Keďže aktuálne existuje veľké množstvo webových služieb ktoré sú verejne prístupné, je pre konzumenta služby ťažké nájsť službu ktorá vyhovuje jeho požiadavkám. Tento problém je možné riešiť odporúčaním služieb.

3.2. Typy odporúčania

Existuje viacero prístupov ku odporúčaniu webových služieb. Vo všeobecnosti sa ale dajú klasifikovať do troch skupín podľa toho, ako pristupujú ku odporúčaniu a aké parametre pri tom využívajú. Sú to nasledovné skupiny [15]:

1. Collaborative Filtering (vo voľnom preklade „filtrovanie podľa kolaborácie“, v tejto práci sa bude využívať anglický termín),
2. Content-based approach (vo voľnom preklade „prístup na základe obsahu“, v tejto práci sa bude využívať anglický termín),
3. Hybrid approach (vo voľnom preklade „hybridný prístup“, v tejto práci sa bude využívať anglický termín).

3.2.1. Collaborative Filtering

Collaborative Filtering (CF) predstavuje prístup ku odporúčaniu služieb na základe predpokladu, že podobní používatelia majú podobné požiadavky. Na základe predchádzajúcich požiadaviek a akcií používateľov sa vytvára ich model, pričom pokiaľ sú modely dvoch používateľov podobné, predpokladá sa že títo používatelia majú podobné požiadavky a teda vyhľadávajú podobné služby. Na základe tohto predpokladu sa odporúčajú používateľovi také služby, aké sa páčili používateľovi ktorý bol označený ako jemu podobný. Tento prístup sa využíva hlavne v komerčných aplikáciách, internetových obchodoch (napríklad E-Bay a Amazon). Jeho nevýhodou je výskyt Cold-start problem-u (vo voľnom preklade – „problém s nedostatkom

počiatočných dát“, v práci sa bude využívať anglický termín), pri ktorom nie je na začiatku dostatok dát na základe ktorých by sa mohlo vykonať odporúčanie. Ďalšou nevýhodou je to, že tento prístup nezohľadňuje iné požiadavky na službu, jej vlastnosti a ani iné atribúty [15].

3.2.2. Content-based approach

Tento prístup pristupuje ku odporúčaniu na základe samotného opisu služby. Systém ktorý odporúča služby týmto spôsobom si uchováva informácie o službách ktoré používateľ označil tak že sa mu páčia a pokúša sa mu odporúčať také, ktoré sú im podobné. Podobnosť služieb sa zisťuje na základe ich opisu. Nevýhodou tohto prístupu je ako v prípade CF prístupu Cold-start problem, ale keďže sa služby porovnávajú na základe ich opisov je možné pri tomto prístupe brať do úvahy aj ich vlastnosti [15].

3.2.3. Hybrid approach

Tento prístup je kombináciou predošlých dvoch prístupov. Rôzni autori kombinujú tieto prístupy v rôznej miere. Pokúšajú sa tak dosiahnuť čo najlepšie odporúčania služieb, pričom sa pokúšajú brať do úvahy tie najlepšie vlastnosti jednotlivých prístupov spolu s pokusom o elimináciu ich nevýhod [15].

3.3. Kontext

3.3.1. Definícia kontextu

Aktuálne neexistuje presná štandardizovaná a všeobecne uznávaná definícia kontextu. Rôzni autori definujú kontext rôzne, pričom veľmi často sa kontext definuje s previazaním voči obsahu (takzvané context-content definície). Je to z toho dôvodu, že nie vždy je možné presne vymedziť hranicu medzi kontextom a obsahom [15].

Existuje viacero opisov kontextu, avšak formálna definícia ktorá je uznávaná veľkou časťou výskumníkov definuje kontext ako „Akákoľvek informácia ktorá môže byť použitá pre charakterizáciu situácie a stavu entity. Entita môže byť osoba, miesto alebo objekt ktorý je relevantný pre interakciu medzi používateľom a aplikáciou, vrátane používateľa a aplikácie“ [16].

Kontext a jeho samotná definícia je oblasťou intenzívneho výskumu v posledných rokoch, pričom výskum sa zameriava aj na jeho modelovanie, využitie a podobne. Kontext môže v sebe zahŕňať funkcionálne aj nefunkcionálne atribúty entity, pričom môže poskytovať informácie o používateľovi a informácie o službe ktoré môžu byť použité pri vyhľadávaní, kompozícií a odporúčaní služieb [15],[16].

Autori najčastejšie definujú dva druhy kontextu – kontext používateľa a kontext služby. Tieto definície nie sú štandardizované, každý autor definuje kontext podľa problému ktorý rieši, prípadne ho definuje tak, aby čo najviac zapadal do jeho koncepcie riešenia problému.

Kontext používateľa (často nazývaný aj ako kontext použitia) sa používa na modelovanie kontextu používateľa ktorý je najčastejšie v roli konzumenta služby. Tento typ kontextu najčastejšie opisuje stav používateľa, jeho polohu (fyzická poloha ako napríklad doma, alebo v práci), správanie, priority, znalosti, platforma pre vyvolanie služby, zámer využitia (pre potešenie, za účelom vzdelávania, z pracovných dôvodov...), frekvencia používania (každodenne, raz týždenne a podobne) a čas použitia (doobeda, večer, ráno,...). V závislosti na vybraných atribútoch kontextu je možné aj eliminovať výskyt Cold-start problému [15],[16].

Kontext služby sa používa na modelovanie kontextu služby ktorá je dostupné pre použitie. Taktiež môže obsahovať rôzne atribúty. Najčastejšími sú typ použitia (pre potešenie, za účelom vzdelávania, z pracovných dôvodov...), funkcionálna kategória (matematika, aritmetika, biológia, ekonómia...), typ služby (pre priame použitie, alebo či sa môže vyskytovať v kompozícií), výstup služby (výsledok algoritmu, vykonateľný kód), dostupnosť služby (neustále dostupná, dostupná v určitých časoch, dostupná len z určitého miesta) a úroveň zabezpečenia [15].

Môže nastať situácia, že z popisu služby vyplýva to, že služba je použiteľná pre všetky možné prípady použitia (ako príklad možno demonštrovať službu ktorá implementuje jednoduché aritmetické operácie). Takáto služba sa označuje ako kontextovo nezávislá (po anglicky „context-free service“) [15].

3.3.2. Aplikácia kontextu v tejto práci

Cieľom tejto práce je vytvorenie systému pre odporúčanie webových služieb. Každá webová služba je vytvorená pre riešenie istého problému, pričom riešenie ktoré táto služba ponúka má isté špecifiká. Tieto špecifiká je možné využiť v odporúčaní tak, aby bolo odporúčanie čo najpresnejšie. Pre reprezentáciu týchto vlastností služby bude slúžiť práve kontext služby.

Každý používateľ má taktiež vlastné špecifiká ktorými je odlíšiteľný od ostatných používateľov. Tieto špecifiká budú využité práve pre personalizované odporúčanie, pričom budú modelované kontextom používateľa. Formálny popis kontextu je spomenutý v kapitole 3.3.1 pričom definícia kontextu služby a kontextu používateľa v tejto práci spolu s ďalším popisom je spomenutá v kapitole 4.3.

3.3.3. Využitie kontextu v odporúčaní

Prístupy ku odporúčaniu služieb uvedené v kapitole 3.2 zlyhávali vo využívaní kontextu pri odporúčaní. Táto práca sa zameriava na využitie moderného a rozvíjajúceho sa prístupu ku odporúčaniu služieb – kontextovo orientovaný prístup.

Bolo zistené, že s využitím kontextu je možné pri odporúčaní služieb dodávať oveľa presnejšie výsledky ako bez neho. Väčšina aktuálneho výskumu odporúčania služieb na základe kontextu sa zameriavala na mobilné zariadenia, pričom klasické webové služby neboli pri tomto prístupe často využívané. Tieto prístupy sa zameriavali hlavne na informácie ktoré je možné získať z mobilných zariadení (ako napríklad poloha, sila signálu a podobne) [15].
S využitím kontextu je možné potlačiť výskyt Cold-start problému. Podľa zvoleného typu kontextu a jeho parametrov je možné monitorovať taký kontext, pre ktorého modelovanie sú potrebné rozsiahle údaje o histórií používateľa, prípadne objektu ktorého kontext modelujeme.
Prístup navrhovaný v tejto práci ku odporúčaniu webových služieb sa ale nezameriava len na mobilné zariadenia, ale na webové služby ako celok. Pri odporúčaní sa využíva kontext používateľa a kontext služby. Modely týchto dvoch typov kontextov sa pri odporúčaní porovnávajú tak, aby boli používateľovi odporúčané čo najpresnejšie služby podľa jeho požiadaviek.

3.4. Softvérový agent

Z dôvodu veľkého množstva webových služieb je pri prehľadávaní, odporúčaní a kompozícií služieb potrebné vykonať značné úsilie. Veľkou výhodou by bolo, pokiaľ by bolo možné tento proces automatizovať. Pri klasických webových službách ktoré neobsahujú sémantický, ale iba syntaktický opis by bola táto úloha značne obmedzená, pokiaľ nie nemožná. Softvér ktorý by automatizoval kompozíciu a odporúčanie služieb by si síce bol vedomý ich syntaktického opisu (získaného z WSDL, spomínané v kapitole 2.5.3), ale nevedel by nič o funkcionalite a účele týchto služieb. Avšak s využitím sémantických webových služieb ktoré okrem syntaktického ponúkajú aj sémantický opis je možné tento proces automatizovať. Softvér by si bol na základe sémantického opisu služby (napríklad vo forme OWL-S, spomínané v kapitole 2.8.2) vedomý jej funkčnej kategórie a účelu. Takto by vedel sám na základe definovaných pravidiel sa rozhodovať o výbere a postupnosti služieb v kompozícií a následnom odporúčaní.
Softvér ktorý dokáže automatizovať tento proces je nazvaný softvérový agent (agent). Softvérový agent možno definovať ako „bežiaci proces ktorý umožňuje implementáciu a prístup ku webovým službám ako ku výpočtovým zdrojom, ktorý ku nim pristupuje ako používateľ a správa sa ako používateľ“ [17].
Agent je teda softvérový komponent ktorý simuluje správanie sa používateľa pri vyhľadávaní a kompozícií služieb, pričom tento proces automatizuje. Je to umožnené na základe využitia syntaktických a hlavne sémantických opisov služieb. Za pomoci automatizácie možno proces odporúčania a kompozície značne urýchliť. Agent si môže byť vedomý nie len sémantických a syntaktických popisov služieb, ale aj ich kontextu. Takto s využitím kontextu služby a kontextu

problému dokáže vylepšiť efektívnosť svojho odporúčania. Taktiež dokáže vytvárať reaktívne kompozície služieb tak, aby výsledná služba spĺňala kritériá používateľa.
Používateľ zadá agentovi jeho požiadavky na službu a následne agent sám vyhľadá a vytvorí kompozície pre používateľa, pričom mu odporučí tie, ktoré vyhodnotí ako najvhodnejšie pre používateľa na základe jeho požiadaviek a kontextu.

3.5. Zhrnutie

Z dôvodu veľkého množstva dostupných služieb na Internete je ich odporúčanie veľmi zaujímavou a dôležitou oblasťou. Odporúčanie na základe kontextu predstavuje nový prístup ku odporúčaniu služieb, pričom bolo zistené že tento prístup dokáže generovať presnejšie a správnejšie výsledky ako bez využitia kontextu [15]. Na základe využitia sémantických webových služieb a agentov je možné tento proces taktiež automatizovať, čo vedie ku zvýšeniu efektivity tohto procesu.

4. Navrhované riešenie

4.1. Špecifikácia problému

Existuje obrovské množstvo rozdielnych webových služieb. Tieto služby sú rozdielne svojou funkcionalitou, poskytovateľom, polohou, zameraním a aj inými atribútmi.
Napriek týmto vlastnostiam, výber služby pre použitie je veľkým problémom, pretože dostupné opisy služieb sa zameriavajú na ich syntaktickú časť, opis funkcionality je často len človekom čitateľný (pokiaľ nejaký existuje) a teda nie je možné tento proces automatizovať a odbremeniť používateľa. Používateľ trávi dlhý čas vyberaním tej správnej služby, zväčša metódou pokus-omyl. Pokiaľ by existoval systém ktorý by vedel používateľom dynamicky odporúčať webové služby, bolo by to veľkým prínosom z hľadiska ušetreného času a efektivity práce. Ako možným riešením tohto problému sa javí sémantický web a s ním spojené sémantické webové služby. Tie poskytujú okrem syntaktickej informácie o službách aj ich sémantický opis ktorý je strojovo čitateľný a vďaka tomu je možné vytvoriť softvér, ktorý bude vedieť efektívnejšie odporúčať služby, keďže si bude vedomý aj správania sa a funkcionality služieb.
Existuje viacero prístupov ku odporúčaniu webových služieb, avšak všetky majú svoje výhody a nevýhody. Najčastejšou nevýhodou je existencia Cold-start problému. Ďalším problémom je, že väčšina prístupov sa zameriava len na odporúčanie na základe podobností používateľov a objektov, pričom aplikovanie odporúčania na základe viacerých parametrov často zlyháva. Takto odporúčané služby nebývajú odporúčané práve pre daného používateľa, ale skôr pre skupinu používateľov, pričom nevýhodou tohto prístupu je nedostatočná personalizácia zameraná na samostatného používateľa. Odporúčanie je totižto aplikovateľné na skupinu používateľov ako celok.

4.2. Popis navrhovaného riešenia

V tejto práci navrhujeme nový prístup ku odporúčaniu sémantických webových služieb s využitím kontextu. Náš prístup ku odporúčaniu služieb je založený na sémantických opisoch, pričom taktiež plánujeme využiť existenciu kontextu. Kontext modelujeme v dvoch pohľadoch – kontext používateľa a kontext služby (kapitola 3.3.1).
Navrhované riešenie predstavuje hybridný prístup ku odporúčaniu služieb. Odporúčanie pomocou content-based approach bude vykonávané na základe sémantického opisu služby. Sémantický opis služby popisuje funkcionalitu služby, takže predstavuje jej obsah na základe ktorého je možné vykonávať odporúčanie. Odporúčanie pomocou collaborative filtering bude

vykonávané na základe histórie vyhľadávaní služieb podobnými používateľmi. Podobnosť používateľov sa bude určovať na základe ich kontextovej podobnosti.

Keďže v prístupe ku odporúčaniu v tejto práci navrhujeme odporúčať služby pre špecifického používateľa a nie len pre skupinu používateľov, je nutné aby sa pri odporúčaní vyskytovalo ďalšie kritérium. V tejto práci navrhujeme odporúčať aj s využitím kontextových informácií špecifického používateľa a špecifickej služby. Tento prístup ku odporúčaniu sa nazýva kontextovo-orientovaný prístup. Keďže vyhľadanie služby iniciuje používateľ podľa vlastných požiadaviek, pričom do samotného odporúčania vstupujú aj atribúty ktoré popisujú priamo daného používateľa, možno hovoriť o tom, že odporúčanie je personalizované práve pre tohto jedného používateľa a tým pádom existuje predpoklad, že výsledky odporúčania budú oveľa presnejšie ako keby bolo odporúčanie vykonané pre skupinu používateľov.

Využitie tohto prístupu nebráni aplikovaniu výhod iných prístupov ku odporúčaniu služieb. Veľkým plusom tradičných prístupov je to, že na základe histórie používateľov dokážu presnejšie odporúčať služby, pretože používateľ ohodnotil služby ktoré mu boli predtým odporúčané. Takto je možné dosiahnuť efekt učenia sa a lepšie odporúčať služby v budúcnosti. Túto výhodu využívame za pomoci využitia collaborative filtering. Veľkou nevýhodou týchto prístupov je ale Cold-start problem. Na začiatku odporúčania nemajú tieto prístupy dostatok informácií o používateľovi a tak nemajú na základe čoho vytvárať odporúčanie. V našom prístupe je ale hneď zo začiatku možné vytvárať odporúčanie, pretože kontext služby a problému je prítomný hneď od začiatku. S postupným používaním a nazbieraním dostatku informácií o používateľovi je možné tento kontext vylepšiť, keďže bude známa jeho história a aj to, aké odporúčané služby vyhodnotil ako užitočné a neužitočné. Na základe týchto informácií bude možné vylepšiť kontext používateľa a služby tak, aby výsledky odporúčania boli čo najpresnejšie.

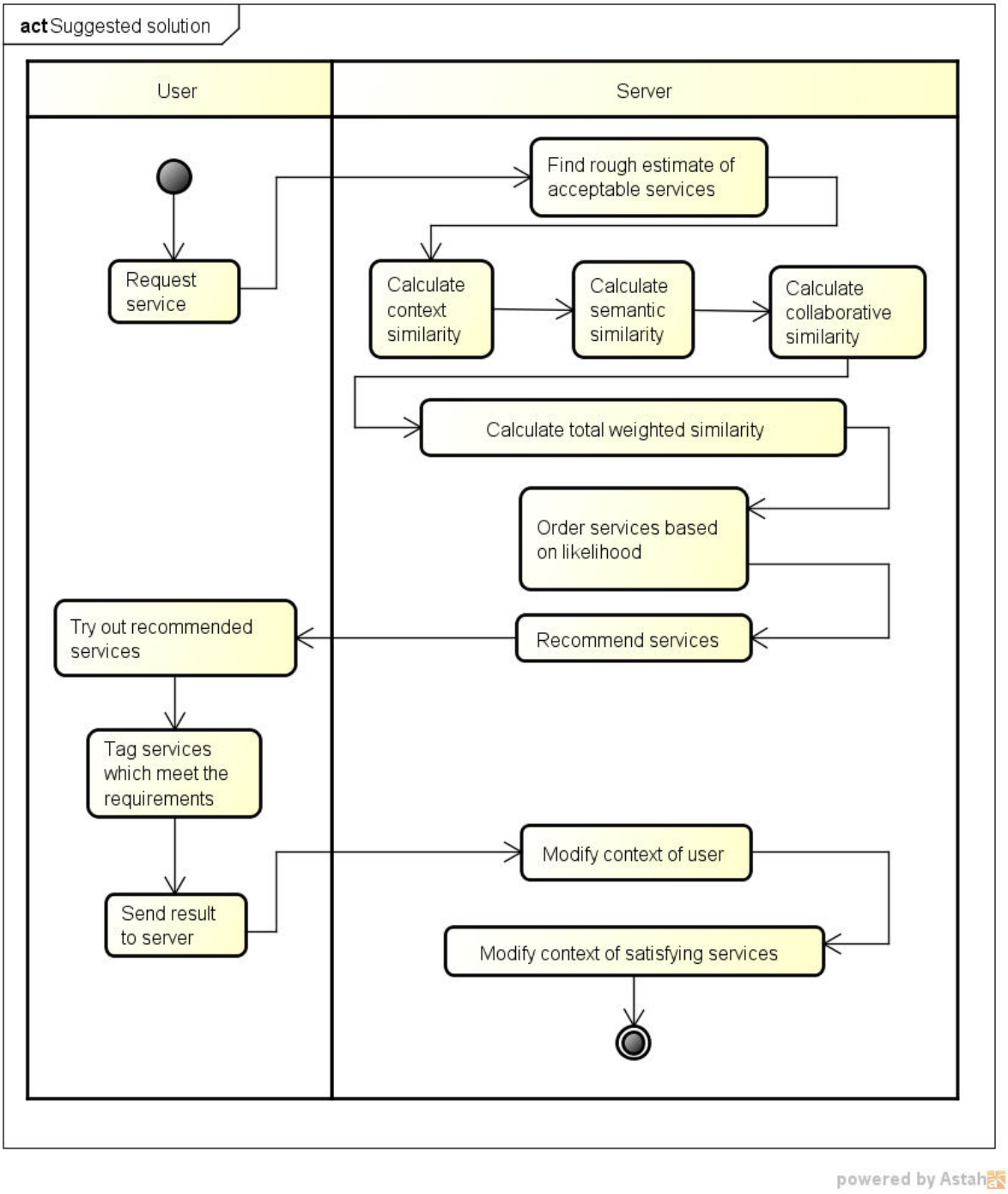
Navrhované riešenie je založené na tradičnom modeli klient-server. Server je zodpovedný za odporúčanie služieb a všetku aplikačnú logiku, klient túto funkcionalitu len vyvoláva. Zjednodušený diagram ktorý popisuje navrhované riešenie je zobrazený na obrázku 5. Navrhované riešenie predpokladá že používateľ sformuluje svoje požiadavky na službu a požiada server o odporúčané služby. Samotné odporúčanie prebieha v niekoľkých krokoch:

1. Server na základe požiadavky vytvorí hrubý odhad služieb ktoré aspoň čiastočne spĺňajú požiadavky používateľa.
2. Následne pre všetky služby z odhadu vypočíta pravdepodobnosť na základe ktorej predpokladá že by mohli používateľovi vyhovovať. Tento výpočet sa skladá z troch častí:
    a. Sémantická podobnosť – podobnosť požiadavky používateľa voči popisu služby.
    b. Kontextová podobnosť – podobnosť kontextu používateľa s kontextom služby.

c. Kolaboratívna podobnosť – podobnosť kontextu služby s kontextom služieb ktoré ohodnotili používatelia podobní používateľovi ktorý vyvolal odporúčanie ako uspokojivé.

Tieto tri segmenty pravdepodobnosti sú následne ováhované a je vypočítaná celková pravdepodobnosť vhodnosti služby.

3. Následne sú služby zoradené podľa vypočítanej pravdepodobnosti.
4. Zoradené služby sú používateľovi odporučené.



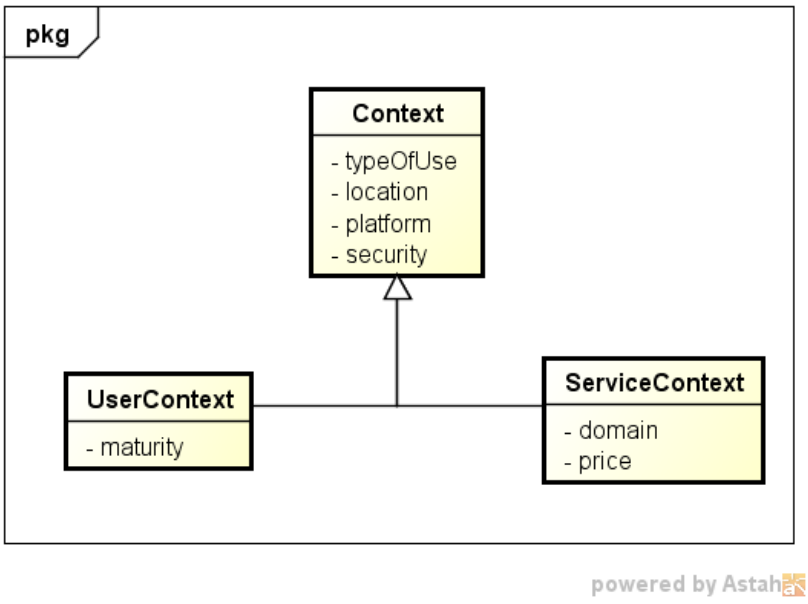
Obrázok 5 Activity diagram zobrazujúci priebeh odporúčania

Následne predpokladáme že používateľ vyberie isté služby zo zoznamu ktorý mu bol odporúčaný a označí tie ktoré spĺňali jeho požiadavky. Na základe týchto akcií sa ďalej ohodnotí zoznam služieb ktoré používateľ vyskúšal. Spracovanie týchto využitých služieb prebieha v dvoch bodoch:

1. Úprava kontextu používateľa – v tomto prípade sa upraví kontext používateľa ktorý vyvolal odporúčanie tak, aby čo najlepšie korešpondoval s kontextom služieb ktoré v histórií pozitívne využil za použitia špecifického vzorca. Tento krok sa vykoná pokiaľ sú splnené špecifické podmienky.
2. Úprava kontextu služby – v tomto prípade sa upraví kontext každej služby ktorá bola používateľom vyskúšaná. Tento proces prebehne len za predpokladu že sú splnené isté špecifické podmienky pre každú službu samostatne. Pokiaľ sú splnené podmienky, tak na základe kontextu používateľov ktorí v histórií využili danú službu, sa upraví kontext danej služby na základe špecifického vzorca.

4.3. Využitie kontextu

Ako bolo spomínané, v tejto práci sa zameriavame na modelovanie, využitie a spresňovanie špecifických atribútov každého používateľa a služby osobitne za pomoci kontextu. Diagram tried ktorý zobrazuje definíciu kontextu v tejto práci je zobrazený na obrázku 6.



Obrázok 6 Class Diagram zobrazujúci definíciu kontextu v tejto práci

Je nutné poznamenať že sa jedná o definíciu kontextu pre túto prácu. Hodnoty atribútov sú zvolené z dôvodu demonštrácie princípu. Je možné si vytvoriť vlastné atribúty, prípadne im aj priradiť vlastné hodnoty. Okrem hodnôt atribútov ktoré sú popísané nižšie, každý atribút môže nadobúdať ďalšiu hodnotu – nulovú. V takom prípade daný kontext nemá zadefinovaný daný atribút.

Model kontextu je všeobecný pre všetky služby a ďalší model pre všetkých používateľov, avšak každá služba a každý používateľ majú vlastný unikátny kontext na základe ktorého je možné

personalizovať odporúčanie práve pre daného používateľa. Z diagramu je vidno že kontext používateľa aj služby dedí od všeobecného kontextu. Každý druh kontextu má teda isté atribúty spoločné a isté špecifické.

Zaujímavými informáciami ktoré by mohli byť súčasťou kontextu sa javia informácie z Quality of Services (QoS, v preklade „kvalita služieb“. V tejto práci sa bude využívať anglický termín). QoS reprezentujú nefunkcionálne atribúty služieb. Pre webové služby QoS sa skladajú primárne z atribútov ktoré popisujú ich výkonnosť ako napríklad priepustnosť, dostupnosť a podobne [26]. Autori v [26] taktiež zistili, že tieto atribúty sú späté s polohou používateľa a služby v sieti – čím je vzdialenosť nižšia, tým sú aj hodnoty týchto atribútov vyššie. Z toho dôvodu je oveľa výhodnejšie modelovať polohu používateľa a služby, namiesto všetkých atribútov pre službu. Avšak problémom bolo ako modelovať atribúty z QoS pre používateľa. Tieto atribúty popisujú skúsenosti používateľa so službou, tým pádom sú priradené k službe a nepopisujú používateľa. Je ale možné modelovať požiadavky používateľa. Ako bolo napísané v [27], existuje veľké množstvo atribútov ktorými je možné modelovať QoS. Napriek tomu že popisujú služby, je nimi možné popisovať požiadavky používateľa na službu.

Z domény QoS boli vyextrahované atribúty polohy a bezpečnosti, ktoré boli pridané do všeobecného kontextu. Taktiež bol vyextrahovaný aj atribút ceny za službu, ktorý bol pridaný do kontextu služby.

Spoločné atribúty sú:

1. **Type of use** (v preklade „typ použitia“) – Tento atribút slúži na zadefinovanie typu činnosti pre aký ktorý bola služba vytvorená, prípadne aký typ funkcionality používateľ požaduje. V tejto práci predpokladáme, že môže mať nasledovné hodnoty:
    a. Business (v preklade „Pre prácu“. V tejto práci sa bude využívať anglický termín) – výstupom služby v tomto prípade predpokladáme viac informácií ako len priamočiaru odpoveď. Taktiež požadujeme vyššiu mieru dostupnosti, avšak môžeme predpokladať že čas potrebný na získanie odpovede môže byť väčší.
    b. Personal (v preklade „Pre osobné účely“. V tejto práci sa bude využívať anglický termín) – pri službách tohto druhu požadujeme priamočiarejšie a jednoduchšie výstupy služieb ako pri business type služieb. Taktiež požadujeme aby bola odpoveď rýchlejšia, avšak predpokladáme že služba nemusí byť stále dostupná. Pri tomto type služieb teda požadujeme rýchle a stručné odpovede.
    c. Casual (v preklade „Pre neformálne účely“. V tejto práci sa bude využívať anglický termín) – pri využití tohto typu služieb rozsah odpovede nie je kritériom. Taktiež služba nemusí byť vždy dostupná, avšak kritériom je rýchla odpoveď služby na požiadavku.

2. **Location** (v preklade „poloha“) – Tento atribút slúži na definovanie polohy, prípadne oblasti pre ktorú bola služba vytvorená, alebo pre ktorú dáva najpresnejšie výsledky. Ako bolo vyššie spomenuté QoS atribúty je možné modelovať cez polohu služby a používateľa. Keby bolo používateľovi umožnené vyhľadávať služby pomocou atribútov QoS ako napríklad dostupnosť a rýchlosť, používatelia by vybrali tie najlepšie hodnoty a modelovanie týchto hodnôt by stratilo význam. Aj z toho dôvodu bol atribút polohy zahrnutý vo všeobecnom kontexte, pretože predstavuje polohu služby, ale aj polohu používateľa a na základe týchto polôh je možné odvodiť atribúty QoS [26]. V tejto práci boli zvolené nasledovné možné polohy : Bratislava, Trnava, Trenčín, Žilina, Liptovský Mikuláš, Poprad, Kysak, Košice. Tieto polohy boli zvolené čisto z dôvodu otestovania funkcionality.
3. **Platform** (v preklade „platforma“) – Tento atribút slúži na definovanie konečnej platformy pre ktorú bola služba vyvinutá. Webové služby sú postavené na princípe nezávislosti od platformy. Napriek tomu faktom je, že každá platforma má svoje špecifiká. Pri modelovaní atribútu platformy v rámci kontextu nie je cieľom istú službu pri odporúčaní vylúčiť, ale vytvoriť predpoklad že pokiaľ bola daná služba vytvorená pre špecifickú platformu, tak pravdepodobnosť že bude spĺňať požiadavky používateľa ktorý požaduje službu práve pre danú platformu je vyššia, ako pri službe ktorá bola vytvorená pre inú platformu. V tejto práci môže atribút platformy nadobúdať nasledovné hodnoty:
    a. Server – predpokladá sa že služba bola vytvorená pre serverovskú platformu.
    b. Tablet - predpokladá sa že služba bola vytvorená pre využitie na tabletoch.
    c. Mobile – predpokladá sa že služba bola vytvorená pre využitie na mobilných zariadeniach, špecificky pre smartphone (v preklade „múdry telefón“, v tejto práci sa bude využívať anglický termín).
    d. PC – predpokladá sa že služba bola vytvorená pre využitie na tradičných počítačoch. Tento atribút zahŕňa v sebe notebooky, aj stolové počítače, pretože z hľadiska funkčnosti nie je medzi týmito typmi počítačov rozdiel.
4. **Security** (v preklade „bezpečnosť“) – Tento atribút je založený na informácií z QoS a predstavuje požadovanú úroveň zabezpečenia služby (z pohľadu kontextu používateľa) a taktiež aj reálnu úroveň zabezpečenia (z pohľadu kontextu služby). Používateľ môže požadovať rôznu úroveň zabezpečenia ktorá je nutná pre jeho potreby, pričom túto požiadavku je možné modelovať ako atribút jeho kontextu. Taktiež každá služba má istú úroveň zabezpečenia ktorú tiež možno modelovať ako jej atribút kontextu. Keďže je možné modelovať tento atribút aj pre službu, aj pre používateľa a taktiež je možné hodnoty tohto atribútu porovnávať, je teda možné zasadiť tento atribút do všeobecného kontextu.

Kontext nie je nemenný. Je možné ho meniť, upravovať a spresňovať.

4.3.1. Kontext služby

Kontext služby obsahuje vlastné špecifické atribúty **domain** (v preklade „doména“) a **price** (v preklade „cena“).

Atribút domény predstavuje oblasť pre ktorú bola služba vytvorená z pohľadu funkcionality. V tejto práci môže tento atribút nadobúdať nasledovné hodnoty:

- Communication (v preklade „komunikácia“)
- Economy (v preklade „ekonómia“)
- Education (v preklade „vzdelanie“)
- Food (v preklade „jedlo“)
- Geography (v preklade „geografia“)
- Medical (v preklade „medicína“)
- Simulation (v preklade „simulácia“)
- Travel (v preklade „cestovanie“)
- Weapon (v preklade „zbraň“)

Tento atribút sa definuje špecificky pre kontext služby a nie pre všeobecný kontext z toho dôvodu, že daná služba môže spadať do určitej funkcionálnej domény, avšak používateľ do funkcionálnej domény nespadá. Používateľ môže vyhľadávať službu v danej doméne, avšak atribút domény nie je priradený pre daného používateľa, jedine pre vyhľadávanie ktoré inicioval.

Atribút ceny služby reprezentuje hodnotu ktoré bude musieť používateľ zaplatiť za využívanie služby. Cena sa modeluje špecificky len pre kontext služby, pretože služba môže mať definovanú vyžadovaný poplatok za využitie, ale používateľ nie. Rôzne služby ktoré môžu vyhovovať požiadavkám používateľa môžu mať rôznu cenu a len na základe uváženia používateľa je možné zistiť či je ochotný danú cenu za využívanie zaplatiť. Z toho dôvodu sa atribút ceny modeluje len pre kontext služby a nie kontext používateľa.

Kontext služby môže vzniknúť dvoma spôsobmi:

1. Zadaný vývojárom – v tomto prípade vytvorí kontext služby vývojár v čase keď službu sprístupní pre odporúčanie.
2. Získaný počas odporúčania – v tomto prípade je kontext služby získaný na základe kontextu používateľov, ktorí danú službu v histórií využili. Pri tejto službe nebol kontext pôvodne zadaný (jeho atribúty boli nulové) a postupne sa vytváral na základe využití danej služby.

Oproti kontextu používateľa, môže byť kontext služby menný alebo nemenný. Kontext služby môže byť nemenný jedine vtedy, ak ho zadefinoval vývojár služby (nebol získaný počas odporúčania) a vývojár zadefinoval že kontext je nemenný. Vývojár dokáže totižto najlepšie definovať pre aký kontext je daná služba najvhodnejšia [15].

4.3.2. Kontext používateľa

Kontext používateľa obsahuje okrem atribútov zdedených od všeobecného kontextu aj špecifický atribút – **maturity** (v preklade „vyspelosť“). Tento atribút modeluje počítačovú gramotnosť používateľa. Každý používateľ má odlišnú úroveň počítačových znalostí, ich využití, ovládania, architektúr a podobne. Z toho dôvodu je vhodné definovať úroveň znalosti, keďže môžeme predpokladať že pre používateľa s nižšou úrovňou bude výhodnejšie odporúčať služby s menším počtom vstupných a výstupných parametrov a pre používateľa s vyššou úrovňou to môžu byť služby s komplikovanejším vstupom a výstupom. V tejto práci môže tento atribút nadobúdať nasledovné hodnoty:

- Low (v preklade „malá“)
- Medium (v preklade „stredná“)
- Mature (v preklade „vyspelá“)
- High (v preklade „vysoká“)

Kontext používateľa si každý používateľ zvolí sám podľa jeho vlastných preferencií. Kontext každého používateľa sa mení v priebehu používania tak aby čo najpresnejšie predstavoval reálny kontext používateľa a tým pádom bolo možné čo najpresnejšie používateľovi odporúčať služby.

4.3.3. Reprezentácia kontextu

Problém ktorý bolo nutné vyriešiť bol problém reprezentácie kontextu. Jedným z možných riešení bolo popísať kontext formou sémantických dát a vytvoriť tak doplnkový popis ku sémantickému opisu služby [15]. Toto riešenie ale nie je vyhovujúce pre túto prácu, pretože táto práca počíta s tým že kontext nie je finálny, ale môže sa meniť. Keďže kontext sa môže meniť, je taktiež nutné uchovávať pre dodatočnú analýzu (prípadne aj štatistiku) históriu kontextu pre danú entitu. Z tohto dôvodu bol zvolený spôsob reprezentácie a uchovávania kontextu vo forme údajov v relačnej databáze. Takto je možné uchovávať kontext pre používateľa aj pre službu, pričom história kontextu sa zachováva a nestratí sa.

4.4. Riešenie problému

V tejto kapitole bude detailnejšie popísané naše riešenie problému odporúčania služieb s využitím kontextu. Definícia kontextu použitého v tejto práci je v kapitole číslo 4.3.
Riešenie navrhované v tejto práci prebieha v troch krokoch:

1. Príprava na odporúčanie
2. Odporúčanie
3. Modifikácia kontextu

Táto práca využíva na odporúčanie služieb sémantické webové služby popísané vo formáte OWL-S (viac v kapitole 2.8.2), pričom využité ontológie sú popísané za pomoci jazyka OWL.

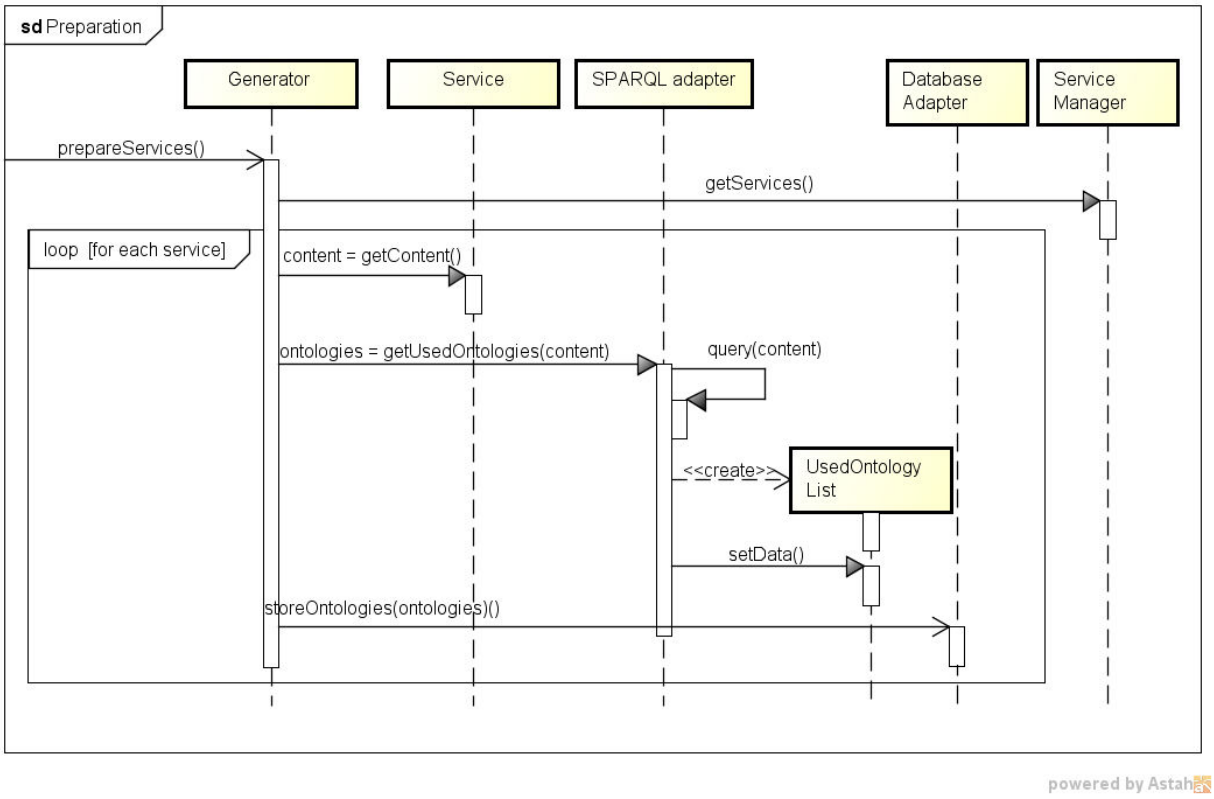
Taktiež predpokladáme že systému na odporúčanie sú prístupné nie len OWL-S služby nad ktorými pracuje, ale taktiež aj OWL ontológie ktoré obsahujú sémantické dáta o použitých službách.

4.4.1. Príprava na odporúčanie

Tento krok prebieha z dôvodu zlepšenia efektivity odporúčania. Spočíva v pred-pripravení údajov z dostupných služieb. Pri každom odporúčaní by bolo nutné prehliadať všetky dostupné služby pre zistenie či služba vyhovuje požiadavkám klienta. Tento proces by sa neustále opakoval, pričom by bolo jednoduchšie extrahovať požadované dáta zo služieb a urýchliť tak vyhľadávanie. Taktiež využité služby sú dostupné cez sieť, pričom vyhľadávanie vo všetkých službách cez sieť môže byť značne pomalé. Navrhované riešenie tohto problému spočíva v príprave na vyhľadávanie pri ktorom sa uložia relevantné údaje do databázy, nad ktorou je možné rýchlo a efektívne vyhľadávať.

Pri kontrolovaní služby či je vhodná pre klienta sa berie do úvahy jej sémantický opis (viac v kapitole 4.4.2). Naše riešenie spočíva v extrahovaní tohto sémantického opisu a jeho uskladnení v databáze, kde bude možné v budúcnosti efektívnejšie a rýchlejšie zisťovať či daná služba vyhovuje požiadavkám používateľa. Zo sémantického opisu služby vo formáte OWL-S extrahujeme vstup a výstup služby. Sémantika vstupných aj výstupných parametrov je opísaná za pomoci tried ontológií v jazyku OWL. Sekvenčný diagram ktorý znázorňuje proces prípravy na odporúčanie je zobrazený na obrázku 7.

Na diagrame je vidno že proces prebieha vo viacerých krokoch. Najprv sa získa zoznam všetkých služieb ktoré má systém k dispozícií. Následne sa pre každú službu za pomoci SPARQL získajú informácie ohľadom vstupných a výstupných triedach pre danú službu. Keďže popis služby je definovaný v jazyku OWL-S ktorý vychádza z RDF, je možné pre tento dopyt využiť SPARQL. Získané súbor informácií ktorý obsahuje vstupné a výstupné triedy ontológií asociované na príslušné služby je následne uložený v databáze pre ďalšie použitie.



Obrázok 7 Sequence diagram zobrazujúci proces prípravy na odporúčanie.

4.4.2. Odporúčanie

Tento krok slúži na priame odporúčanie služieb pre používateľa, pričom výstupom je zoradený zoznam služieb ktoré sú používateľovi odporúčané podľa predpokladanej pravdepodobnosti toho, či mu budú vyhovovať. Pri vstupe je potrebné aby používateľ definoval sémantický (funkčný) popis služby ktorú požaduje. Tento popis modelujeme ontológiou pre sémantický opis a taktiež aj funkčnou doménou do ktorej má služba spadať.

Vstupná ontológia je zadefinovaná v jazyku OWL a je potrebné aby obsahovala minimálne nasledovné dve triedy:

1. Input (v preklade „vstup“) – slúži na definovanie sémantického opisu ktorý má spĺňať vstup služby ktorú používateľ požaduje.
2. Output (v preklade „výstup“) - slúži na definovanie sémantického opisu ktorý má spĺňať výstup služby ktorú používateľ požaduje.

Tento typ popisu požiadavky na službu bol zvolený preto, lebo OWL je štandardom na popis sémantických dát, pričom v správnom použití je ním možné modelovať požiadavky na vstup a výstup. Taktiež ontológia nie je obmedzená na špecifickú implementáciu a je možné ju teda využiť ako všeobecnú požiadavku na službu nezávisle od výstupného formátu a typu služby. OWL taktiež podporuje viacnásobné dedenie, tým pádom je možné aby požadovaná trieda dedila

od rôznych iných tried, čím je možné dosiahnuť presne zadefinovanú požiadavku. Používateľ taktiež nemusí vždy nanovo definovať svoju vlastnú triedu ontológie, ale dokáže využiť existujúce dostupné ontológie a ich triedy ktoré mu vyhovujú za pomoci dedenia.

Vstupom do odporúčania je teda ontológia ktorá obsahuje triedy ktoré definujú požiadavky na vstup a výstup služby. Tieto triedy možno upraviť tak, aby vyhovovali akýmkoľvek požiadavkám za pomoci dedenia. Problém však nastáva vtedy, pokiaľ je nutné implementačne vyhľadať použiteľné služby. Z popisu služieb vieme ktoré triedy ontológie danú službu využívajú, avšak manuálne prehľadávať všetky služby a všetky ontológie pri každej požiadavke na službu by bolo veľmi neefektívne.

Z objektovej paradigmy programovania vyplýva, že všetky podtriedy triedy „A“ možno považovať ako typ triedy „A“. Tým pádom je nutné nájsť všetky možné služby, ktorých sémantický opis vo forme OWL triedy je podtriedou triedy ktorá je definovaná v požiadavke na službu. Prvým krokom je teda nájdenie týchto podtried. Prípravou na odporúčanie sme zistili ktoré služby sú popísané ktorými triedami ontológií a tieto informácie uložili do databázy. Po tom ako získame súbor všetkých podtried, môžeme v databáze vyhľadať služby ktorých sémantický popis sa nachádza v súbore podtried získanom z požiadavky. Následné ohodnotenie služieb a ich zoradenie už nie je náročné, ani zložité.

Samotné odporúčanie je rozdelené do troch krokov:

1. Zistenie možných tried ontológií zo vstupnej ontológie - v tomto kroku sa na základe vstupu vyberú všetky podtriedy vstupnej ontológie.
2. Výber dostupných služieb ktoré spĺňajú podmienky – v tomto kroku sa zo všetkých služieb ktoré má systém dostupné vyberú tie, ktorých triedy ontológie sa nachádzajú v súbore podtried získaných v predošlom kroku.
3. Ohodnotenie a zoradenie služieb – v tomto kroku sa pre každú službu ohodnotí percento jej vhodnosti pre klienta a zoradia sa podľa tohto ohodnotenia.

Zistenie možných tried ontológií

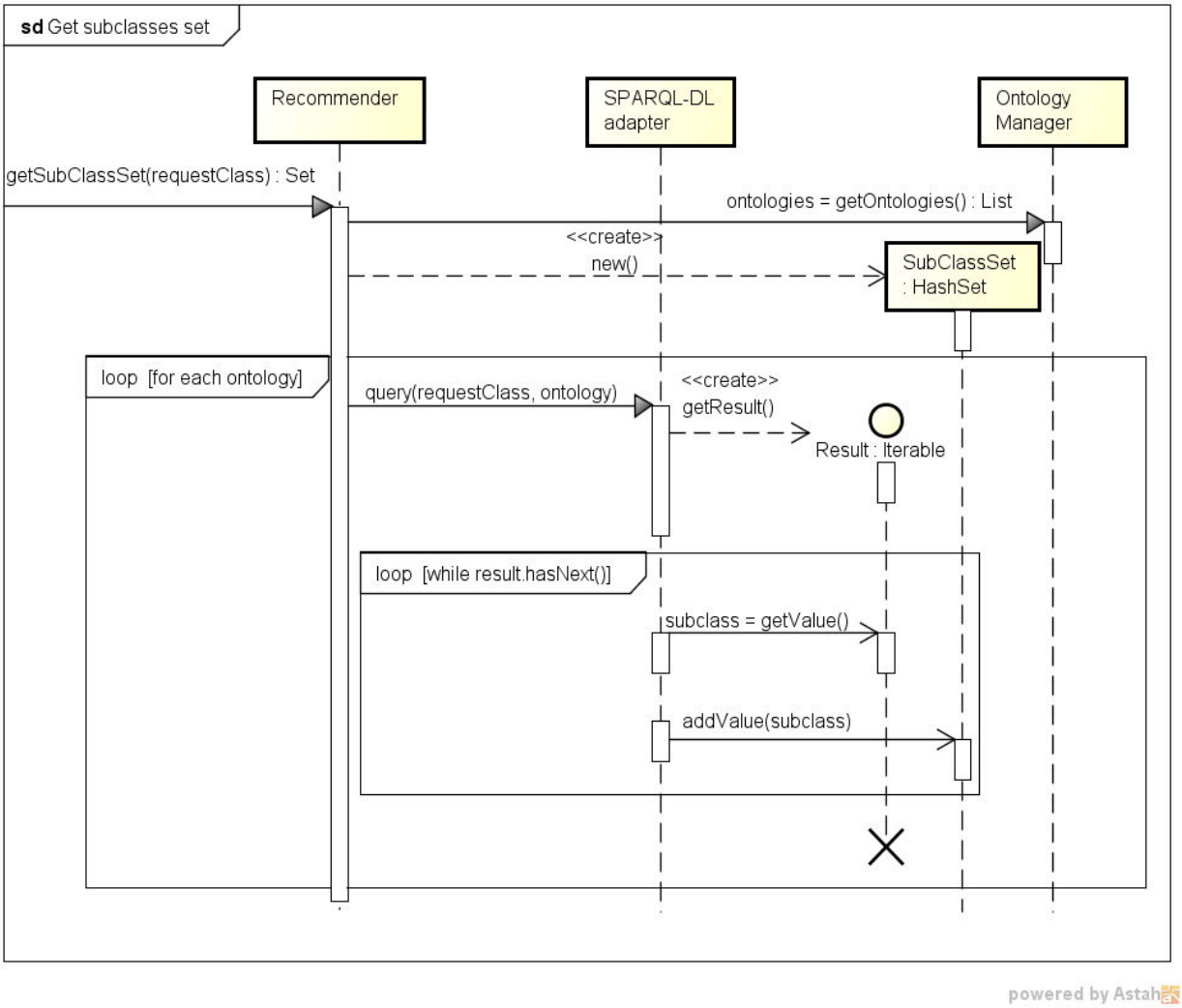
SPARQL slúži na dopytovanie sa nad sémantickými dátami, avšak nie je schopné pracovať so schémami RDFS a logikou vo vnútri OWL (ako napríklad dedenie). Manuálne prehľadávanie všetkých dostupných ontológií by bolo veľmi náročné. Z toho dôvodu je na riešenie tohto problému využité SPARQL-DL. SPARQL-DL dokáže spracovať aj informácie vo vnútri OWL a tak je možné automaticky a jednoducho vyhľadávať v dostupných ontológiách.

Tento krok teda slúži na získanie všetkých možných tried ontológií ktoré spĺňajú požiadavky používateľa na službu. Sekvenčný diagram ktorý znázorňuje proces zistenia možných podtried je zobrazený na obrázku 8. Proces zobrazený na diagrame prebieha pre triedy ktoré reprezentuje vstup a výstup požadovanej služby. Tento proces prebieha v nasledovných krokoch:

1. Získanie všetkých prístupných ontológií.

2. Pre všetky ontológie sa za pomoci SPARQL-DL vykoná dopyt ktorý má vyhľadať všetky podtriedy vstupnej triedy v danej ontológií.
3. Všetky výsledky dopytu sa uložia do súboru implementovaného ako HashSet aby sa predišlo duplicite tried.
4. Tento súbor naplnený všetkými vyhovujúcimi triedami sa vráti ako výstup.

Tento proces nie je možné vykonať v príprave na odporúčanie, pretože kritériom podľa ktorého sa vyhľadáva v ontológiách je samotná požiadavka používateľa a teda nie je možné ho zovšeobecniť pre všetky požiadavky.



Obrázok 8 Sequence diagram zobrazujúci vyhľadanie všetkých podtried.

Výber služieb

Po tom ako boli získané všetky možné triedy ontológií ktoré vyhovujú požiadavkám používateľa na vstup a výstup služby, je možné vyhľadať služby ktoré spĺňajú používateľove požiadavky na základe týchto tried. Toto vyhľadávanie sa vykoná na základe údajov získaných v procese prípravy na odporúčanie (bližšie vysvetlené v kapitole 4.4.1). Vyhľadávanie prebieha spôsobom SQL dopytu do databázy, kde vstupom je súbor všetkých možných podtried a výstupom sú tie služby, ktorých vstup alebo výstup (v závislosti od toho či podtrieda definuje vstup alebo výstup) je popísaný nejakou triedou z tohto súboru. Takto sa získa zoznam obsahujúci tie služby, ktoré aspoň v minimálnej miere spĺňajú požiadavky používateľa na službu. V tomto zozname sa nachádzajú aj tie služby, ktoré požiadavky spĺňajú len čiastočne (napríklad môže spĺňať požiadavku na vstup, ale na výstup nie). Tento zoznam označujeme ako **prvotný odhad**.
Keďže prvotný odhad už čiastočne spĺňa požiadavky používateľa, už ten mu môže byť teoreticky odporúčaný. Avšak pre zlepšenie tohto odporúčania a dosiahnutia čo najpresnejších výsledkov sa tento odhad ďalej spresňuje.

Ohodnotenie a zoradenie služieb

V tomto kroku sa prvotný odhad služieb získaný v predošlom kroku spresňuje a zoraďuje. Je to z dôvodu získania čo najvhodnejšieho odporúčania. Spresňovanie prebieha spôsobom výpočtu nasledovných položiek:

- Podobnosť požiadavky používateľa na službu a jej sémantického opisu (sémantická). Hodnota tejto podobnosti predstavuje percentuálnu zhodu požiadaviek používateľa na vstup a výstup služby, s aktuálnym vstupom a výstupom služby. Pre ilustráciu možno demonštrovať príklad, v ktorom používateľ požaduje službu, kde trieda ontológie ktorá popisuje vstup dedí od triedy A a trieda ktorá popisuje výstup od triedy B a C. Predpokladajme že pre istú službu je jej vstup popísaný triedou A‘ ktorá je podtriedou A a jej výstup triedou B‘ ktorá je podtriedou B. Daná služba teda spĺňa požiadavky používateľa z dvoch tretín, pretože spĺňa požiadavku na vstup, ale požiadavku na výstup spĺňa len čiastočne (vyhovujú dve triedy z troch). Percentuálne by to bola teda 66.6%-ná zhoda. Pokiaľ ale predpokladáme odlišnú službu, ktorej vstup je popísaný triedou A‘ ktorá je podtriedou A a jej výstup popísaný triedou B‘ ktorá je podtriedou B, triedou C‘ ktorá je podtriedou C a navyše triedou D, tak napriek tomu že výstup služby je popísaný o jednou triedou navyše, predstavuje z hľadiska sémantickej podobnosti presnú zhodu, pretože plne spĺňa požiadavky klienta na službu (všetky tri položky z troch) – percentuálne 100%.
- Podobnosť kontextu používateľa a kontextu služby (kontextová). Keďže kontext služby a kontext používateľa má spoločný základ (viac v kapitole 4.3), je možné tieto kontexty porovnávať. Okrem kontextových parametrov používateľa sa porovnáva aj atribút domény kontextu služby a požiadavka na doménu ktorú špecifikoval používateľ pri

požiadavke na odporúčanie a taktiež aj atribút ceny kontextu služby a cena ktorá bola špecifikovaná v požiadavke. Tak ako pri sémantickej podobnosti, aj tu sa výsledok vypočítava na základe percentuálnej zhody oboch kontextov, pričom princíp je popísaný v predošlej časti.

- Podobnosť kontextu služby s kontextami služieb ktoré používatelia podobní používateľovi v histórií využili (kolaboratívna). Podobnosť používateľov definujeme na základe podobnosti ich kontextov. Používateľ je podobný druhému používateľovi práve vtedy, pokiaľ sa ich kontextové informácie zhodujú, alebo odlišujú maximálne v jednej položke. Podobných používateľov získame veľmi ľahko a efektívne na základe jednoduchého SQL dopytu do databázy. Získame teda kontexty služieb ktoré podobní používatelia v minulosti využili a boli s nimi spokojní (tiež s využitím jednoduchého SQL dopytu). Z kontextov týchto služieb vytvoríme všeobecný kontext. Hodnoty atribútov tohto kontextu tvoria tie atribúty, ktoré mali v kontextoch služieb najpočetnejšie zastúpenie. Tieto kontexty následne porovnáme a zistíme percentuálnu zhodu tak, ako pri výpočte predošlých podobností.

Po tom ako získame všetky tri typy podobností, vypočítame celkovú vhodnosť. Táto vhodnosť predstavuje predpokladanú percentuálnu pravdepodobnosť toho, že používateľ bude s danou službou spokojný (že mu bude úspešne odporučená). Vypočíta sa ako aritmetický priemer jednotlivých podobností, pričom každú podobnosť je možné ováhovať. Vzorec tejto vhodnosť je zobrazený v rovnici 1.

$$likelihood_{total} = \frac{\begin{array}{l}(weight_{\text{semantic}} * likelihood_{\text{semantic}}) + \\ (weight_{\text{context}} * likelihood_{\text{context}}) + \\ (weight_{\text{collaborative}} * likelihood_{\text{collaborative}})\end{array}}{3}$$

Rovnica 1 Rovnica výpočtu celkovej vhodnosti služby.

Váhy jednotlivých podobností zadáva používateľ a slúžia na definovanie, aká podobnosť má pre používateľa aký význam. Môže sa stať že používateľ požaduje len tie služby, ktoré spĺňajú sémantickú podobnosť. V tom prípade označí váhu sémantickej podobnosti na 3 a váhy ostatných podobností na 0.

Po tom ako majú všetky služby v zozname vypočítanú ich vhodnosť, je možné ich na základe tohto ohodnotenia zoradiť od najväčšieho po najmenšie (od najväčšej pravdepodobnosti vhodnosti po najmenšiu). Tento zoradený zoznam je následne vrátený používateľovi ako finálne odporúčanie služieb.

4.5. Modifikácia kontextu

Po tom ako je používateľovi odporúčaný zoznam služieb zoradený podľa predpokladanej úrovne vhodnosti, používateľ vyberie služby z tohto zoznamu. Po tom ako využije ich funkcionalitu, označí či bol s využitými službami spokojný. Tento zoznam ktorý obsahuje služby ktoré boli nevyužité a využité (spolu s ohodnotením či vyhovovali) sa vráti do systému pre ďalšie spracovanie. V tomto kroku nastáva modifikácia kontextov tak, aby kontext čo najpresnejšie modeloval reálny kontext služby, alebo používateľa. Tento proces nastáva z dôvodu upresňovania, pričom predpokladáme že so spresnením kontextových údajov bude možné presnejšie odporúčať služby v budúcnosti.

Zo zoznamu ktorý používateľ vráti po využití služieb sa uložia tie ktoré používateľ využil, pričom sa v databáze uložia nasledovné informácie:

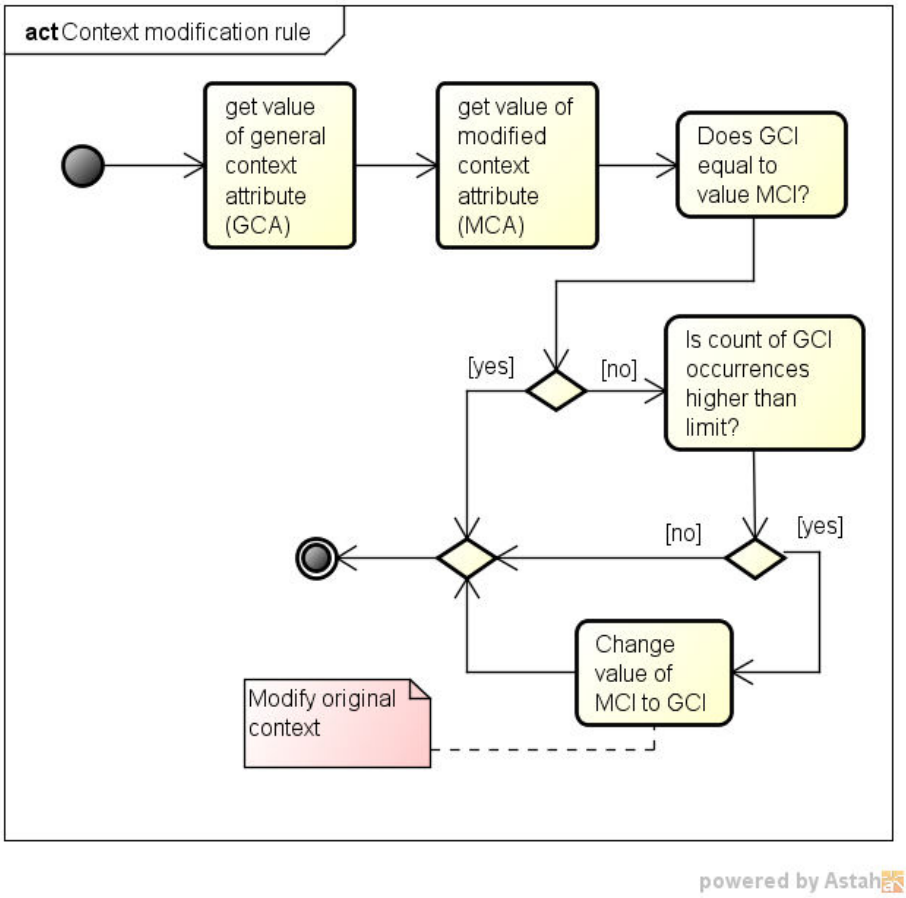
- Ktorý používateľ danú službu využil.
- Ktorá služba bola využitá.
- Spokojnosť používateľa s danou službou.
- Aká bola požiadavka používateľa na službu.

Po tom ako sa uložia tieto informácie prebieha modifikácia kontextu.

4.5.1. Proces modifikácie kontextu

Kontext sa vždy modifikuje na základe kontextu všeobecného a kontextu modifikovaného. **Všeobecný kontext** sa vytvára spôsobom podobným ako pri všeobecnom kontexte služieb pri kolaboratívnej podobnosti. Tento všeobecný kontext je vytvorený z atribútov typov kontextov, ktorých hodnoty sa zovšeobecňujú. Hodnoty jeho atribútov sú tvorené najpočetnejšími hodnotami kontextov z ktorých sa tento všeobecný kontext vytvára, pričom početnosť týchto najpočetnejších hodnôt je taktiež zachovaná.

Atribút kontextu ktorý má byť spresnený sa modifikuje vždy na základe definovaného procesu. Tento proces je znázornený na obrázku 9.



Obrázok 9 Activity diagram zobrazujúci proces modifikácie kontextu.

Vstupom do tohto procesu je hodnota atribútu všeobecného kontextu (GCA), jeho početnosť a hodnota atribútu modifikovaného kontextu (MCA). Tento proces prebieha vždy pre všetky atribúty kontextu ktorý je spresňovaný. Proces prebieha v nasledovných krokoch:

1. Ak sú hodnoty GCA a MCA totožné, tak hodnota všeobecného kontextu je totožná s hodnotou modifikovaného kontextu a nie je nutná zmena a proces končí. Inak sa pokračuje do ďalšieho kroku.
2. Pokiaľ je početnosť hodnoty GCA nižšia ako minimálna početnosť pre zmenu kontextu, tak proces končí, inak sa pokračuje ďalej. Minimálna početnosť je definovaná ako percento početnosti hodnoty GCA oproti početnosti všetkých hodnôt atribútu (početnosť GCA vzhľadom na počet kontextov z ktorých bol vytvorený všeobecný kontext). V tejto práci má hodnotu 85%, pričom v nastaveniach systému je možné túto hodnotu modifikovať.
3. Pokiaľ proces pokračuje až do tohto kroku, tak boli splnené všetky podmienky na zmenu kontextu a MCA je zmenený na GCA – nastala modifikácia atribútu kontextu.

4.5.2. Modifikácia kontextu používateľa

Kontext používateľa sa modifikuje na základe kontextu služieb, ktoré v minulosti využil a bol s nimi spokojný. Keďže informácie o využití služieb sa archivujú v databáze, je možné zoznam kontextov služieb, ktoré používateľ v minulosti využil a bol s nimi spokojný získať jednoduchým SQL dopytom. Všeobecný kontext sa vytvorí na základe kontextu všetkých služieb ktoré kedy používateľ využil a bol s nimi spokojný. Následne je kontext modifikovaný procesom popísaným v kapitole 4.5.1.

4.5.3. Modifikácia kontextu služby

Pre všetky služby ktoré boli využité používateľom, pričom bol s nimi spokojný, začína proces modifikácie ich kontextu. Ich kontext sa však mení len vtedy, pokiaľ ho je možné zmeniť. Vývojár služby totižto môže zadať, že služba ktorú vyvinul má kontext nemenný a teda nie je možné ho ďalej automaticky modifikovať (viac v kapitole 4.3.1). Pokiaľ je možné kontext služby meniť, tak sa najprv pre danú službu vygeneruje všeobecný kontext. Ten je pre danú službu vytvorený z kontextov všetkých používateľov, ktorí túto službu využili a boli s ňou spokojný. Tieto informácie sú uložené v databáze, je ich možné jednoducho získavať SQL dopytom. Následne je kontext modifikovaný procesom popísaným v kapitole 4.5.1.

4.6. Zhodnotenie

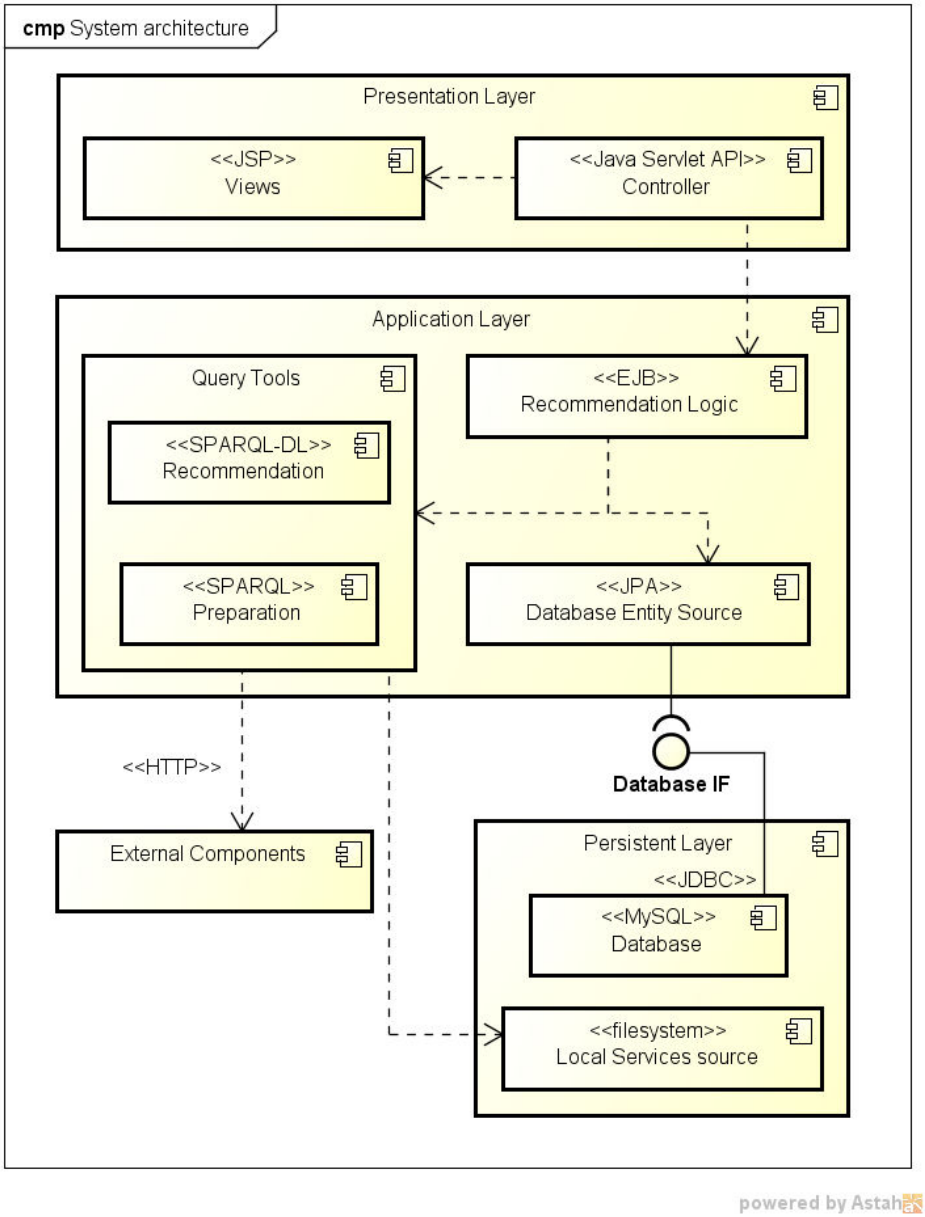
Bol vytvorený prístup ku odporúčaniu s využitím kontextu. Nie len že využitie kontextu bolo v predošlých prácach zväčša zanedbávané, náš prístup je navyše unikátny tým, že dokáže automaticky a autonómne upresňovať jednotlivé kontexty, dôsledkom čoho predpokladáme dosiahnutie presnejšieho vyhľadávania.

Väčšina prístupov ku odporúčaniu v histórií sa zväčša orientovala len na vytvorenie metódy, avšak ku samotnej implementácií nedošlo. Náš prístup bol taktiež úspešne implementovaný (viac o implementácii v kapitole 5) a je teda možné navrhovaný metódu aj prakticky overiť (viac v kapitole 6).

5. Popis architektúry a implementácie

5.1. Popis

Zvolená architektúra systému vychádza z architektonického štýlu Klient - Server, pričom sa jedná o tradičnú trojvrstvovú architektúru. Popis architektúry je zobrazený na obrázku 10. Implementácia prebiehala v jazyku Java, pričom boli využité moderné štandardy pre vývoj Enterprise projektov. Taktiež boli využité voľne dostupné knižnice pre prácu so sémantickými dátami. Systém bol vytvorený na aplikačnom serveri JBoss.



Obrázok 10 Component diagram zobrazujúci architektúru systému.

V systéme bol taktiež využitý architektonický štýl Model – View – Controller (MVC). Komponent prezentačnej vrstvy „Views“ tu predstavuje MVC komponent „View“, MVC controller je reprezentovaný komponentom „Controller“ a MVC „Model“ je reprezentovaný komponentom EJB - „Recommendation Logic“.

5.1.1. Prezentačná vrstva

Vrchná vrstva predstavuje prezentačnú vrstvu, pričom je tu využitý štýl MVC. Bola implementovaná za použitia štandardných technológií JSP a Java Servlet API. Skladá sa z nasledovných komponentov:

1. Views (v preklade, „pohľady“) – tento komponent je implementovaný technológiou JSP a slúži pre zadefinovanie obrazoviek cez ktoré bude používateľ interagovať so systémom.
2. Controller (v preklade, „kontrolér“) – slúži pre spracovávanie požiadaviek používateľa, vyvolávanie funkcionality uloženej v aplikačnej vrstve a taktiež aj na úpravu a obnovenie dát v komponente „Views“. Tento komponent je implementovaný za pomoci Java Servlet API.

Pre vizuálnu stránku prezentačnej vrstvy bol využitý voľne dostupný template zo stránky w3layouts.com [30].

5.1.2. Aplikačná vrstva

Táto vrstva je implementovaná za pomoci technológie EJB vo verzii 3.1. Technológia EJB bola oproti iným tradičným frameworkom ako napríklad SPRING [18] zvolená primárne z dôvodu lepšieho oddelenia aplikačnej a prezentačnej vrstvy, dôsledkom čoho je možné tieto dve časti systému vyvíjať samostatne a presnejšie sa zamerať na ich požadovanú funkcionalitu. Taktiež je týmto rozhodnutím podporený princíp softvérového inžinierstva modularity a oddelení záležitostí. Ďalším dôvodom bola podpora automatických transakcií, mapovania, monitoringu a logovania.

Aplikačná vrstva sa skladá z troch komponentov:

1. Recommendation Logic (v preklade, „logika odporúčania“) – tento komponent je zodpovedný za samotné odporúčanie služieb.
2. Database Entity Source (v preklade, „ zdroj databázových entít“) – tento komponent slúžil pre mapovanie hodnôt uložených v databázových tabuľkách do objektov za pomoci technológie JPA.
3. Query Tools (v preklade, „nástroje pre dopytovanie“) – tento komponent sa skladal z dvoch podkomponentov ktoré mali za úlohu vykonávať dopyty nad sémantickými dátami a službami. Boli to nasledovné komponenty.

a. Recommendation (v preklade „odporúčanie“) – komponent ktorý mal za úlohu vykonávať dopyty v priebehu odporúčania, pričom bol implementovaný za pomoci SPARQL-DL. Implementácia SPARQL-DL bola z frameworku vytvoreným Derivo Semantic Systems [21].
b. Preparation (v preklade „príprava“) – komponent ktorý mal za úlohu vykonávať dopyty v priebehu prípravy na odporúčanie. Implementovaný bol za pomoci SPARQL zo systému Apache Jena [19].

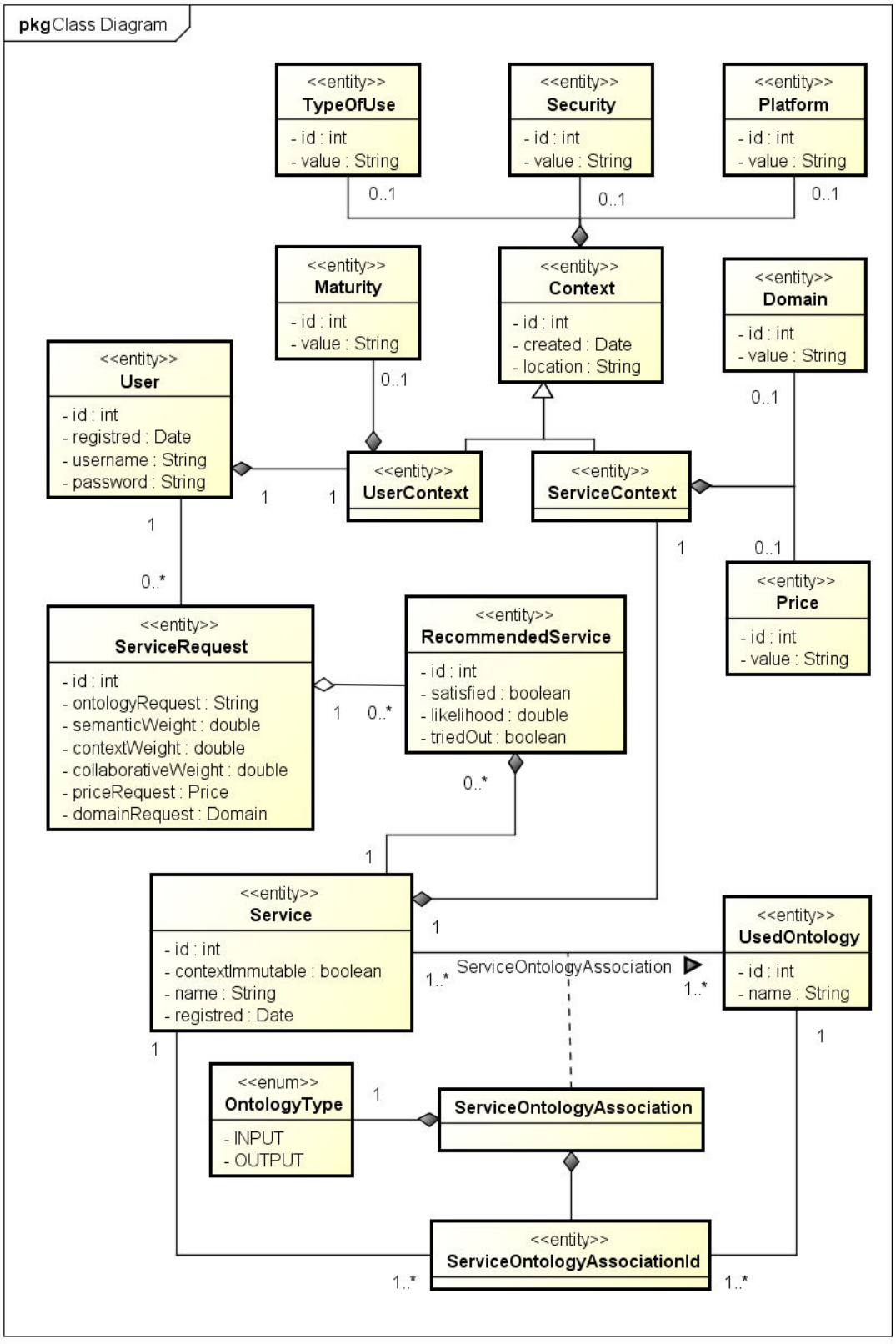
Pre prácu so sémantickými dátami boli zvolené systém Apache Jena [19] a knižnica pre prácu so sémantickými službami OWL API [20]. Oba zdroje sú verejne dostupné, pričom je nutné poznamenať že Apache Jena má veľmi dobrý súbor príkladov, dokumentácie a návodov na Internete. Jej problémom ale je, že sa zameriava na prácu so sémantickými dátami a nie službami. Nebolo teda možné naplno využiť potenciál tohto systému. Naproti tomu, práca s ostatnými externými zdrojmi bola veľmi komplikovaná, pretože prakticky neexistovali návody a ukážky a navyše dokumentácia bola veľmi stručná. Napriek tomu bola implementácia úspešná.

Diagram zobrazujúci dátovú štruktúru v systéme je na obrázku 11. Je tu možno vidieť už spomínanú definíciu kontextu, pričom každý kontext si zachovával informáciu ohľadom toho kedy vznikol (atribút „*timeCreated*“) aby bolo možné analyzovať vývoj kontextu služby, prípadne používateľa.

Z diagramu taktiež vyplýva že každý používateľ a aj služba majú asociovaný práve jeden kontext, pričom každý kontext je asociovaný práve jednému používateľovi alebo službe. Taktiež služby sú asociované s triedami ontológií („UsedOntologyClass“). Tieto triedy ontológií popisujú služby s ktorými sú asociované (viac o tomto vzťahu a jeho princípe v kapitole 4.4.1). Každá trieda ontológie môže popisovať viacero služieb, pričom každá služba môže byť popísaná viacerými triedami ontológií.

Požiadavka používateľa na službu je reprezentovaná triedou „ServiceRequest“. V tejto triede sú uložené všetky parametre ktoré môže používateľ špecifikovať v požiadavke na odporúčanie. Taktiež je znázornené že používateľ môže mať viacero požiadaviek a každá je priradená práve jednému používateľovi.

Ďalším objektom ktorý sa vyskytuje je trieda „RecommendedService“ ktorá slúži na reprezentáciu odporučenej služby ako odpoveď na požiadavku. To či ju používateľ využil modeluje atribút „*triedOut*“ a to či bol s ňou spokojný modeluje atribút „*satisfied*“. Trieda taktiež uchováva aj informáciu vypočítanej vhodnosti tejto služby („*likelihood*“) a je spätá s požiadavkou ktorá iniciovala vytvorenie odporučenej služby a taktiež aj so samotnou službou ktorá bola odporučená.



Obrázok 11 Class diagram zobrazujúci dátovú štruktúru tried v systéme.

5.1.3. Perzistentná vrstva

Táto vrstva slúži na ukladanie dát ktoré sú využívané v systéme. Skladá sa z relačnej MySQL databázy a lokálneho úložiska ontológií a služieb. Tieto služby a ontológie sú uložené v lokálnom súborovom systéme.

5.1.4. Externé komponenty

Keďže pri zadávaní požiadavky na odporúčanie môže používateľ taktiež využiť existujúce ontológie ktoré sa nemusia nachádzať v perzistentnej vrstve systému ale mimo neho, je potrebné aby ich mal systém prístupný. Tieto externé komponenty ktoré môžu zahŕňať napríklad webové služby alebo ontológie sú prístupné cez Internet za pomoci protokolu HTTP.

5.2. Popis použitých dát

V tejto práci boli využité sémantické webové služby popísané vo formáte OWL-S za pomoci ontológií vo formáte OWL. Zdrojom týchto služieb bol verejne dostupný balík OWL-S TC3 [22]. Tento balík obsahoval 1083 rôznych webových služieb z deviatich rôznych funkčných kategórií (komunikácia, ekonómia, vzdelanie, gastronómia, geografia, medicína, simulácia, cestovanie a zbrane).

Tieto služby boli popísané pomocou 38 rôznych ontológií, pričom každá z ontológií obsahovala desiatky až stovky rôznych tried ontológií.

Používatelia systému boli náhodne vygenerovaní systémom. Kontext používateľov a služieb bol taktiež náhodne vygenerovaný funkcionalitou odporúčacieho systému navrhovaného v tejto práci. Kontexty boli vygenerované s využitím pravdepodobnosti výskytu kontextu pre službu alebo používateľa, pričom pravdepodobnosť výskytu jednotlivých atribútov kontextov bolo tiež možné ovplyvniť spolu s ich hodnotami.

5.3. Problémy pri implementácii

Pri väčšine projektov zameraných na odporúčanie služieb skončí práca pri návrhu metódy, pričom ku samotnej implementácií dospeje veľmi málo projektov. Problematická bola teda samotná implementácia. Na Internete sa prakticky nenachádzali zdroje ktorými by bolo možná sa inšpirovať, moderné systémy ktoré by boli dobre zdokumentované spolu s praktickými ukážkami taktiež v podstate neexistovali.

Problémom teda bolo nájsť existujúce riešenia ktoré by bolo možné využiť a zakomponovať do práce. Z dôvodu nedostatku dokumentácie, bolo nutné systém implementovať po častiach spôsobom pokus - omyl.

Ďalším veľkým problémom bol zvláštny prístup W3C [9] ku sémantickým webovým službám. Ako prvý návrh pre štandard W3C pre sémantické webové služby bolo navrhnuté OWL-S. Tento spôsob popisu služieb vzbudil veľké očakávania u vývojárov, pričom veľká časť komunity sa začala práve na tento typ popisu orientovať. Následne avšak W3C prestala s týmto formátom pracovať. Ďalším návrhom pre štandard sa stal SA-WSDL, ktorý sa nakoniec aj štandardom stal. Problém však bolo, že tento typ popisu nevzbudil u vývojárov také nadšenie ako OWL-S a teda aj dostupných softvérových nástrojov pre prácu s SA-WSDL bolo malé množstvo. Nástrojov pre prácu s OWL-S bolo množstvo väčšie, avšak keďže W3C sa rozhodlo pre SA-WSDL, tak tieto nástroje sa prestali udržiavať a ďalej vyvíjať. Postupne prestali byť použiteľné a keďže zameranie W3C sa prenieslo na SA-WSDL ktoré nevzbudilo veľký záujem vývojárov, tak nájsť dostupné nástroje pre prácu so sémantickými webovými službami bolo veľmi zložité.

6. Overenie

6.1. Popis overenia

Cieľom práce je navrhnúť a vytvoriť systém na odporúčanie služieb s využitím kontextu. V práci sa taktiež zameriavame aj na spresňovanie kontextovej informácie. Z toho dôvodu sa overenie zameriava na dve otázky:

1. Dosahuje odporúčanie s využitím kontextu presnejšie výsledky ako prístupy ktoré nevyužívajú kontext? (overenie prístupu)
2. Má zmysel spresňovanie kontextu? (celkové overenie)

6.2. Použité metriky

Pri overení bol využitý nástroj SME v2.2 [28]. Tento nástroj slúži na vyhodnocovanie efektivity jednotlivých odporúčacích systémov. Jednotlivé odporúčacie systémy je možné do neho vkladať vo forme pluginov. V tejto práci sa pri overení využije metrika ktorú používa práve nástroj SME. Použité metriky sú precision, recall a F1. Rovnice výpočtu jednotlivých metrík sú zobrazené v rovnici 2.

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot recall \cdot precision}{recall \cdot precision}$$

Rovnica 2 Rovnice metrík využitých pri overení systému.

Pri tejto metrike sa využívajú štyri rôzne triedy ktoré reprezentujú všetky možné výsledky odporúčania pre službu. Odporučené služby sú zadelené do danej triedy podľa toho či mala byť služba odporučená a či bola odporučená. Sú to nasledovné triedy:

- True positive (TP, v preklade „správne pozitívna“) – služba mala byť odporučená a bola odporučená,
- False positive (FP, v preklade „nesprávne pozitívna“) – služba nemala byť odporučená a bola odporučená,
- True negative (TN, v preklade „správne negatívna“) – služba nemala byť odporučená a nebola odporučená,
- False negative (FN, v preklade „nesprávne negatívna“) – služba mala byť odporučená a nebola odporučená.

Po tom ako odporúčací systéme vo forme plugin-u do SME odporučí služby, tak výsledky odporúčania sú zaradené do príslušných tried a následne sa vypočíta hodnota jednotlivých metrík. Pre overenie bola využitá sad služieb OWL-S TC3 [22]. Táto sada služieb obsahuje okrem dát popísaných v kapitole 5.2 aj 43 rôznych dopytov. Pre každý dopyt je definovaný aká má byť požiadavka na službu, nad akou množinou služieb sa má dopyt vykonať a taktiež aj očakávaný výsledok. Každý výsledok dopytu je ohodnotený atribútom „*grade*“. Tento atribút predstavuje očakávanú vhodnosť služby. Môže nadobúdať hodnoty – 0, 1, 2 a 3. Čím je hodnota tohto atribúty vyššia, tým sa aj očakáva lepšie ohodnotenie služby.

6.3. Overenie prístupu

Overenie prístupu bolo realizované porovnaním výsledkom nášho systému s výsledkami iných odporúčacích systémov. Pre porovnanie bol využitý systém SME a metriky ktoré využíva. Ako odporúčací systém na porovnanie bol zvolený plugin do SME a to iSeM v1.1 [29]. Tento odporúčací systém dokáže odporúčať na základe viacerých metód. Pre porovnanie boli zvolené rôzne metódy s rôznymi prístupmi, aby bolo možné spoľahlivo overiť náš prístup s rôznymi inými. Boli vybrané nasledovné metódy:

- iSeM approx. logic-based – na základe logických štruktúr porovnáva vstupy a výstupy služby a požiadavky na službu. Vyhľadáva logické konštrukcie v požiadavke a pokúša sa ich párovať s popisom služby, pričom predpokladá že štruktúry nemusia byť úplne podobné, ale môžu byť medzi nimi aj menšie nezrovnalosti.
- iSeM logic-based – podobne ako pri predošlom prístupe, avšak vyhľadáva logické konštrukcie ktoré sa líšia v čo najmenšej možnej miere.
- iSeM structure – porovnáva štruktúru požiadavky na službu so štruktúrou popisu služby. Odporúča tie služby, ktorých štruktúra sa líši čo najmenej.

Pri evaluácií boli vybrané štyri dopyty z OWL-S TC3, ktoré boli administrované nášmu systému a taktiež aj všetkým trom metódam s ktorými mal byť systém porovnaný. Každý dopyt bol definovaný pre rôzny počet služieb. Taktiež každá služba sa mohla vyskytovať vo viacerých sadách ktoré boli definované pre dopyty. Dopyty boli vybrané preto, lebo sa mali vykonať nad rôznym počtom služieb, boli v rôznom formáte, boli z rôznych aj podobných domén a taktiež aj počet očakávaným výsledkov sa líšil. Na základe takého širokého okruhu bolo možné dostatočne otestovať funkcionalitu systému. Vybrané boli nasledovné dopyty (v zátvorkách sú udávané identifikátory jednotlivých dopytov v sade OWL-S TC3):

A. PersonBookPrice (id 2) – dopyt sa má vykonať nad 194 službami. Na základe účtu používateľa mu požadovaná služba vyhľadá cenu za knihu ktorú požaduje.
B. BookPriceService (id 4) – dopyt nad 198 službami. Požiadavkou je nájdenie služby, ktorá dodá cenu knihy, špecifikovanej ako parameter.

C. CityCountaryHotelService (id 6) – dopyt nad 112 službami. Požiadavkou je nájdenie služby ktorá vyhľadá hotel v okrese špecifikovanom ako parameter.
D. Publication-number PublicationService (id 19) – dopyt nad 19 službami. Požiadavkou je nájdenie služby ktorá vyhľadá publikáciu na základe špecifikovaného čísla publikácie.

Jednotlivým systémom boli dopyty administrované samostatne, aby sa predišlo riziku ovplyvňovania aktuálnych výsledkov odporúčania predošlými dátami. Taktiež pri odporúčaní naším systémom, bola po každom odporúčaní premazaná databáza používateľov a služieb. Cieľom bolo simulovať odporúčanie, pri ktorom nie sú známe žiadne predošlé interakcie so systémom.

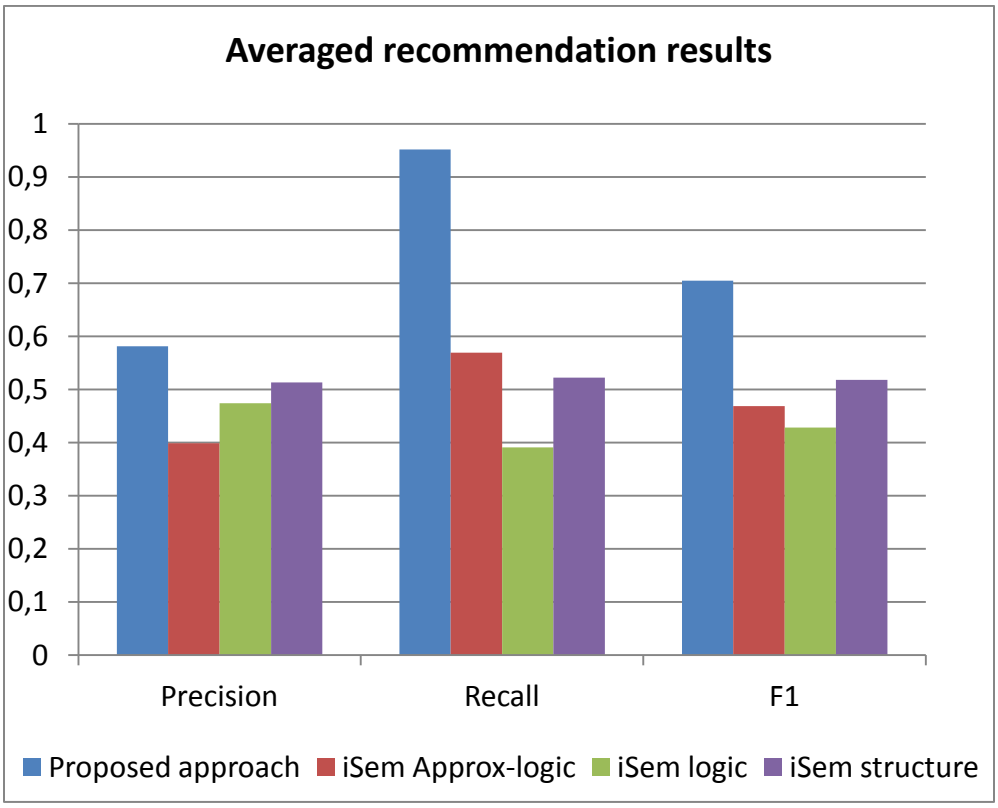
Pred odporúčaním vykonaným našim systémom, bolo nutné vygenerovať používateľa a jeho kontext. Taktiež bolo nutné importovať služby ktoré mali byť použité v dopytoch do systému a aj im vygenerovať kontext. Keďže predpokladáme že pri službách ktoré vyhovujú používateľovi má ich kontext byť čo najpodobnejší s kontextom používateľa, tak kontexty služieb boli vytvorené tak, aby odzrkadlovali očakávanú vhodnosť služby (*grade*). Čím bola hodnota atribútu *grade* vyššia, tým viac bol kontext služby podobný s kontextom používateľa. Po tom ako bol vytvorený používateľ, jeho kontext a taktiež aj služby boli importované a bol im vytvorený kontext, mohlo začať odporúčanie. Váhy jednotlivých zložiek odporúčania v našom systéme mali rovnaké hodnoty (žiadna zložka sa pri výpočte nepreferovala).

Všetkým trom systémom vybraným na porovnanie a aj nášmu boli postupne administrované dopyty. Po odporučení boli výsledky vyhodnotené metrikou systému SME. Následne bola premazaná databáza a akákoľvek história odporúčaní a proces sa postupne opakoval pre všetky dopyty. Nástroj SME vyhodnocoval odporučené služby na základe parametra *grade* očakávaného výsledku. Pokiaľ hodnota parametra nebola nulová, tak služba mala byť odporučená, inak odporučená byť nemala. V našom systéme sme teda porovnávali hodnotu parametra *grade* s hodnotou *likelihood* ktorá bola vrátená našim systémom a na základe toho priradili odporučený službu do jednej zo štyroch tried (TN, TP, FN, FP) ktoré sa využívajú pri evaluácií podľa hodnôt v tabuľke 2:

Likelihood / Grade	**0**	**1, 2, 3**
0	TN	FN
(0,1>	FP	TP

Tabuľka 2 Jednoduché priradenie odporučenej služby do výslednej triedy.

Po vykonaní všetkých odporúčaní nad všetkými systémami, bol z jednotlivých výsledkov vypočítaný aritmetický priemer. Dosiahnuté výsledky sú zobrazené na grafe 1.



Graf 1 Výsledky porovnania systému s inými systémami.

Pri overení prístupu bolo vyhodnotenie vykonané aj druhým spôsobom pri ktorom sa bral do úvahy atribút *grade*. Výsledky zostali nemenné, iba zaradenie služieb odporúčaných naším systémom do jednotlivých štyroch tried (TN, TP, FN, FP) sa odvíjalo aj od hodnoty *grade,* nie len od toho či bola hodnota nulová alebo nie. Nástroj SME má vlastné vyhodnotenie odporúčania do ktorého nemožno zasahovať ani ho ovplyvniť. Je možné len využiť jeho metriky a porovnať výsledky. V predošlom vyhodnotení nebola braná do úvahy hodnota atribútu *grade.* V ďalšom vyhodnotení boli odporučené služby zaradené do jednotlivých tried na základe hodnôt v tabuľke 3:

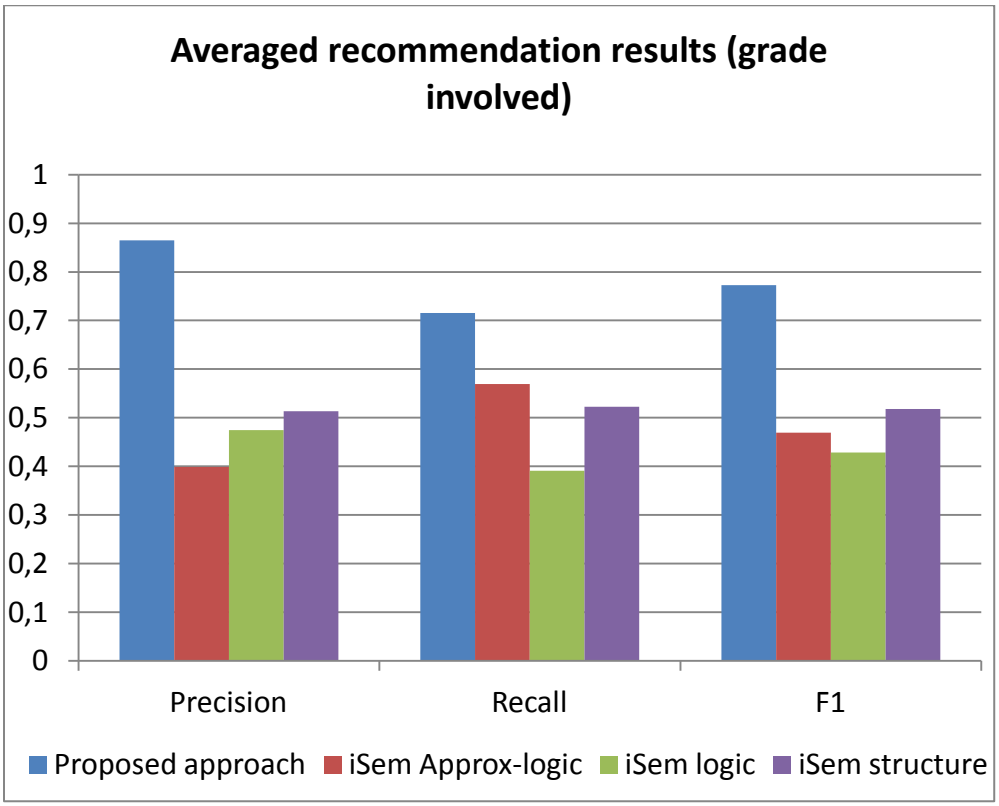
<table>
<tr><th>Likelihood / Grade</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td><1 ; 0,66></td><td>TP</td><td>FP</td><td rowspan="2">FP</td><td rowspan="3">FP</td></tr>
<tr><td>(0,66 ; 0,5></td><td rowspan="3">FN</td><td>TP</td></tr>
<tr><td>(0,5 ; 0,3></td><td rowspan="2">FN</td><td>TP</td></tr>
<tr><td>(0,3 ; 0)</td><td>FN</td><td>TN</td></tr>
</table>

Tabuľka 3 Priradenie odporučenej služby do výslednej triedy na základe hodnoty parametra *grade.*

Prvý interval má hodnotu <1 ; 0,66> z toho dôvodu, že nie je možné aby pri prvotnom odporúčaní dosiahla služba väčšie ohodnotenie *likelihood* ako 0,66. Keďže všetky váhy sú nastavené na rovnakú hodnotu a nie sú predošlé odporúčania ktoré by mohli ovplyvňovať

výsledok, tak nenulovú hodnotu môže mať len sémantická a kontextuálna zložka. Kolaboratívna je nulová a z toho dôvodu nie je možné dosiahnuť pri danom nastavení požiadavky odporúčania lepšie ohodnotenie ako 0,66.

Po následnom vyhodnotení hodnôt metrík boli dosiahnuté nasledovné výsledky:



Graf 2 Výsledky porovnania systému s inými systémami, s prihliadnutím na hodnoty *grade*.

Ako je vidno na oboch grafoch (Graf 1, Graf 2), pri oboch vyhodnoteniach boli naším systémom dosiahnuté lepšie výsledky vo všetkých metrikách. Navrhovaný systém dokázal pri odporúčaní bez predošlých interakcií produkovať lepšie výsledky. S využitím kontextovej informácie je teda možné preklenúť Cold-start problém.

6.4. Celkové overenie

Pri tomto overení sa zameriavame na celkové overenie prístupu, pri ktorom predpokladáme že sa mení kontext a taktiež aj ho využívajú rôzni používatelia a teda kolaboratívna zložka nebude nulová. Zameriavame sa teda na celkové zlepšovanie odporúčania a mieru zlepšenia pri upresňovaní kontextu.

Overenie bolo vykonané postupným administrovaním viacerých požiadaviek a vyhodnocovaním odporučených služieb. Pri tomto overení boli taktiež vybrané dopyty zo sady OWL-S TC3. Boli

vybrané dva dopyty ktoré boli definované z rozličných domén a mali sa vykonať nad rozličnými službami. Vybrané boli nasledovné dopyty:

A. CityCountaryHotelService (id 6)
B. Publication-number PublicationService (id 19)

Oba dopyty boli využité aj pri overené prístupu v kapitole 6.3. Pri tomto overení boli taktiež využité metriky precision, recall a F1. Boli vygenerovaní traja používateľa (používateľ A, B a C). Používatelia B a C mali vyplnené všetky atribúty ich kontextov a používateľ A mal vyplnené všetky atribúty okrem atribútu *typeOfUse*. Kontexty používateľov boli vygenerované tak, aby boli podobné. Kontext používateľa A je ten, ktorý chceme v overení spresňovať.

Služby definované pre dopyty A a B boli taktiež importované do systému, pričom taktiež ako v overení prístupu im bol vytvorený kontext na základe ich ohodnotenia parametrom *grade*. Pre všetky služby bol kontext vytvorený a označený ako nemenný z dôvodu zjednodušenia – demonštruje sa len zmena kontextu používateľa.

Pri celkovom overení boli postupne zasielané požiadavky na službu jednotlivými používateľmi. Pri všetkých požiadavkách mali váhy jednotlivých zložiek rovnaké hodnoty. Postupnosť požiadaviek je zapísaná v tabuľke 4:

Iterácia	**Používateľ**	**Požiadavka**
1	A	A
2	B	A
3	C	A
4	A	A
5	B	B
6	C	B
7	A	A

Tabuľka 4 Iterácie odporúčaní pri celkovom overení.

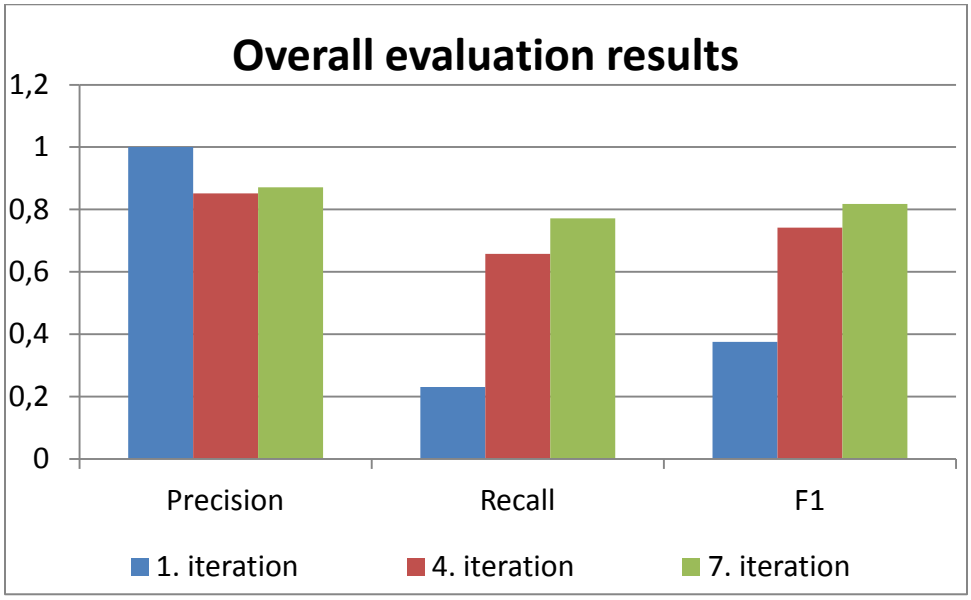
Vyhodnocované boli len výsledky odporúčania pre používateľa A. Tento používateľ vždy požadoval tú istú požiadavku, aby bolo možné vyhodnotiť zlepšenie v odporúčaní. Pri overení sme taktiež simulovali zvyšujúce sa nároky používateľov na služby. Používatelia pri ich prvom odporúčaní ohodnotili pozitívne tie služby, ktoré mali vlastnosť *likelihood* väčšiu alebo rovnú 60%. Pri ich každej ďalšej interakcií so systémom sa tento nárok zvýšil o 12% (teda pri druhej bola požadovaná hodnota 72% a tak ďalej).

Zaradenie odporúčaných služieb do jednej zo štyroch tried bolo vykonané na základe hodnôt v tabuľke 5.

Likelihood / Grade	3	2	1	0
<1;0,8)	TP	FP	FP	FP
<0,8 ; 0,5)	FN	TP		
<0,5 ; 0,3)		FN	TP	
<0,3 ; 0>			FN	TN

Tabuľka 5 Priradenie odporúčanej služby do výslednej triedy v celkovom overení

Postupne vyhodnotené výsledky odporúčania pre používateľa A sú zobrazené na grafe 3.



Graf 3 Výsledky celkového overenia systému.

Pri prvej iterácií neboli žiadne predošlé dáta o odporúčaní a teda kolaboratívna zložka bola nulová a taktiež aj používateľ nemal zadané všetky atribúty kontextu. Na grafe je teda vidno že v prvej iterácií neboli výsledky podľa celkovej metriky F1 veľmi uspokojivé.

Pri štvrtej iterácií používateľ zadal druhú požiadavku. V minulosti už boli uskutočnené aj odporúčania inými používateľmi, kolaboratívna zložka teda už nebola nulová. Vo výsledkoch je možné pozorovať rapídne zlepšenie (podľa F1 metriky o 36%). Pri tejto požiadavke ešte používateľ nemal spresnené hodnoty kontextu. Avšak pri vyhodnocovaní tohto odporúčania mu boli hodnoty kontextu spresnené a teda pri jeho ďalšom odporúčaní bude môcť systém naplno využiť všetky zložky podobnosti.

Pri siedmej iterácií v ktorej používateľ zadáva jeho tretiu požiadavku už je v databáze dostatok predošlých interakcií pre výpočet kolaboratívnej podobnosti a taktiež používateľ má spresnený kontext a každý atribút jeho kontextu nadobúda hodnotu. Je badať zlepšenie vo výsledkoch všetkých metrík oproti výsledkom po štvrtej iterácií.

Vo výsledkoch je teda možné pozorovať že spresňovanie kontextuálnej informácie spolu s využívaním systému vedie ku presnejším výsledkom. Využívaním je teda možné dosiahnuť oveľa lepšie výsledky, aj za predpokladu že kontextová informácia je nesprávna, pretože sa po čase spresní.

7. Záver

7.1. Porovnanie s inými prístupmi

Keďže prístupov ku odporúčaniu je veľké množstvo, porovnáme náš prístup s prístupmi ktoré sa pokúšali využiť pri odporúčaní kontextovú informáciu.

Prístup s využitím QoS navrhovaný autormi X.Chen, Z. Zheng, X. Liu, Z. Huang a H. Sun

V práci „Personalized QoS-Aware Web Service Recommendation and Visualization“ [23] sa autori zameriavajú okrem iného aj na odporúčanie služieb. Ich prístup spočíva vo využití algoritmov pre collaborative filtering (CF) a taktiež využitie informácií získaných z QoS. Na základe QoS autori získavajú atribúty služieb ktoré chcú využiť pri spresňovaní odporúčania. V práci sa zameriavajú špecificky na polohu služby a polohu používateľa. Poloha používateľa je známa na základe jeho IP adresy a polohu služby zadáva vývojár. Tento atribút je podobný atribútu polohy z kontextu služby a kontextu používateľa využitých v tejto práci. V spomínanej práci chcú autori docieliť spresnenie a zefektívnenie odporúčania. Ich prístup spočíva vo filtrovaní služieb na základe atribútov z QoS (demonštrujú na príklade polohy) a následne aplikovaní algoritmov pre CF.

Prístup v našej práci naproti tomu ráta aj so sémantickým opisom služby a priamou požiadavkou používateľa na službu, čím sme schopní odporúčať služby aj napriek Cold-Start problému. V našej práci, rovnako ako v spomínanej práci, plánujeme využívať CF pri odporúčaní. V našej práci zahŕňame QoS atribúty do kontextu, avšak kontext obsahuje oveľa viac informácií. Kontext môže totižto v sebe zahŕňať všetky informácie z QoS, avšak QoS nedokáže reprezentovať všetky informácie získané z kontextu – kontext má širší zámer. V našej práci je taktiež možné hodnoty kontextu upresňovať, pričom hodnoty QoS bývajú často dané, prípadne nie je možné na ne priamo vplývať. V našej práci taktiež ponúkame možnosť ováhovania jednotlivých zložiek ktoré majú vplyv na odporúčanie, čím sa vieme lepšie prispôsobiť požiadavkám používateľa.

Prístup s využitím kontextu navrhovaný autormi L. Liu, F. Lecue, N. Mehandjiev

V práci „Semantic Content-Based Recommendation of Software Services using Context“ [15] sa autori zameriavajú na odporúčanie s využitím content-based approach, pričom taktiež pri odporúčaní modelujú kontext. V danej práci nerozlišujú medzi kontextom používateľa a služby. Vstupom do odporúčania v danej práci je sémantický opis požadovanej služby vo formáte SA-WSDL a taktiež aj požiadavka na kontextovú informáciu. V procese odporúčania sa analyzujú dostupné služby a porovnáva sa ich kontextová a sémantická časť s kontextovou a sémantickou časťou požiadavky.

V našej práci modelujeme špecifický kontext pre používateľa a službu, pretože napriek tomu že majú rovnaký základ, používateľ a služba majú rozdielne vlastnosti ktoré nemožno vzájomne

prepájať (napríklad doména popísaná v kapitole 4.3.1), pričom do kontextu zahŕňame aj informácie z QoS. V našej práci taktiež predpokladáme že kontext nie je finálny, ale môže sa ďalej upresňovať. Týmto plánujeme dosiahnuť väčšiu mieru presnosti pri odporúčaní. Ďalej v našej práci využívame pri odporúčaní okrem sémantickej (content-based approach) a kontextovej podobnosti, aj kolaboratívnu podobnosť (CF), čím plánujeme dosiahnuť presnejšie výsledky pri odporúčaní. Ďalším rozdielom je formát požiadavky na službu. V spomínanej práci sa predpokladá že požiadavkou je SA-WSDL dokument. V našej práci je požiadavkou OWL ontológia, ktorá je všeobecnejšia, neviaže sa na presný formát služby, pričom pri požiadavke je možné využiť už existujúce ontológie a tým uľahčiť prácu používateľa. Taktiež v našej práci ponúkame možnosť ováhovania jednotlivých zložiek ktoré majú vplyv na odporúčanie, čím sa vieme lepšie prispôsobiť požiadavkám používateľa.

V oboch spomínaných riešeniach sa služby vyhľadávali prechádzaním zoznamu dostupných služieb. V našej práci demonštrujeme efektívnejší, všeobecnejší a jednoduchší spôsob s využitím SPARQL a SPARQL-DL.

7.2. Ďalšia práca

V ďalšej práci v tomto projekte sa plánujeme zamerať na algoritmus odporúčania služieb. Aktuálne je sémantická podobnosť vypočítavaná na základe dedičností tried ontológií. Jazyk OWL ponúka viac možností pre popis sémantických vzťahov a bolo by veľmi výhodné pokiaľ by sa sémantická vhodnosť služby vypočítavala aj na základe týchto rozšírených možností. Z toho dôvodu je prvým krokom navrhovaným v ďalšej práci vylepšenie výpočtu sémantickej vhodnosti služby.

Pri porovnávaní kontextov sa aktuálne atribút polohy porovnáva jednoduchým porovnaním hodnôt, pričom sa neberie do úvahy vzdialenosť polôh od seba a taktiež polohy sú reprezentované v jednoduchej forme. Zaujímavý prístup ku tomuto problému navrhujú autori práce „Location-Aware Collaborative Filtering for QoS-Based Service Recommendation“ [26], v ktorej navrhujú zadeľovať používateľov a služby do regiónov na základe ich polohy v sieti. Navrhujú prístup, v ktorom podľa vzdialeností jednotlivých regiónov dokážu vypočítať hodnoty QoS používateľa pre ním požadovanú službu. Zapracovaním navrhovaného riešenia by sa vyriešil problém reprezentácie polohy v kontexte a taktiež aj výpočet kontextuálnej zložky pri podobnosti polôh by bol efektívnejší.

7.3. Zhodnotenie

V tejto práci sme navrhli a implementovali systém na odporúčanie webových služieb s využitím kontextu. Nie len že sme pri odporúčaní využili kontextovú informáciu služieb ktorá v doterajších prístupoch nebola bežne využívaná, taktiež sme aj navrhli a implementovali spôsob ktorým je možné tento kontext automaticky spresňovať. V práci bol využitý hybridný spôsob odporúčania pri ktorom sa kladie dôraz nie len na samotný obsah služby, ale aj na aktivitu podobných používateľov.

V systéme sú využité moderné štandardy a prístupy pre prácu so sémantickými dátami. Architektúra systému bola navrhnutá s prihliadaním na moderné štandardy a princípy softvérového inžinierstva. Funkcionalita systému je popísaná za pomoci jazyka UML, pričom sú dodané aj textové opisy aby bolo možné rýchlo a jednoducho pochopiť vytvorenú a navrhovanú funkcionalitu. Vďaka moderným technológiám je odporúčanie poskytované rýchlo. Formát požiadavky je abstraktný a v požiadavke je možné využívať už existujúce riešenia a tým odbremeniť používateľa od nutnosti zdĺhavého definovania vlastnej požiadavky.

Pri overení systému sme sa zamerali na overenie prístupu a celkové overenie. Boli využité metriky, ktoré ponúka externý systém pre porovnávanie odporúčaní, metriky teda neboli nami navrhnuté. V overení prístupu sme porovnali náš systém s inými systémami ktoré nevyužívali kontext v odporúčaní. Odporúčania boli vytvárané bez predošlých interakcií so systémami. Výsledkami bolo dokázané, že náš systém dokáže odporúčať presnejšie ako iné systémy. Taktiež bolo dokázané, že s využitím kontextovej informácie je možné preklenúť Cold-start problém. Pri celkovom overení sme sa zamerali na výhody spresňovania kontextu a tiež aj na zlepšenie odporúčania pri využívaní systému. Bolo dokázané, že so spresňovaním kontextu je možné dosiahnuť lepšie výsledky a taktiež aj že pri využívaní systému viacerými používateľmi odporúčací systém dodáva lepšie výsledky.

8. Zdroje

[1] Erl, T.: Service-Oriented Architecture: Concepts, Technology, and Design. 1. vyd. Prentice Hall Professional Technical Reference, 2005. 760 s. ISBN 0-13-185858-0

[2] Erl, T.: SOA Principles of Service Design. 1. vyd. Prentice Hall Professional Technical Reference, 2008. 573s. ISBN 0-13-234482-3

[3] Rosen, M., Lublinsky, B., Smith, K., Balcer, M.: Applied SOA: Service-Oriented Architecture and Design Strategies. 1. vyd. Indianapolis: Wiley Publishing, Inc., 2008. 657 s. ISBN 978-0-470-22365-9

[4] Erl, T., Utschig, C., Maier, B., Normann, H., Trops, B.: Next Generation SOA: A Real-World Guide to Modern Service-Oriented Computing. 1. vyd. Prentice Hall Computer, 2014. 400s.

[5] Rozinajová, V.: Princípy Informačných Systémov: Architektúra informačných systémov, 5. prednáška. Bratislava: Slovenská technická univerzita, Fakulta informatiky a informačných technológií, 2011.

[6] Maamar, Z., Mostefaoui, S.K., Yahyaoui, H.: Toward an agent-based and context-oriented approach for Web services composition. In: IEEE Transactions. Knowledge and Data Engineering. IEEE, May 2005, s. 686 – 697.

[7] Dustdar, S., Scheiner, W.: A survey on web services composition, In: Int. J. Web and Grid Services. Inderscience Enterprises Ltd., 2005, s. 1-30.

[8] Alesso, H.P.,Smith. C.F.: Developing Semantic Web Services. 1. vyd. Canada:A K Peters, Ltd., 2005. 445 s. ISBN 1-56881-212-4.

[9] W3C. http://www.w3.org/ , 3.12.2014

[10] Pierce. M., Fox. G.: SOAP I: Intro and Message Formats. http://www.ggf.org/GGF15/presentations/NMApps/SOAP1.pdf , 3.5.2014.

[11] Suda. B.: SOAP Web Services. http://suda.co.uk/publications/MSc/brian.suda.thesis.pdf , 3.5.2014.

[12] Adamczyk. P., Smith. H. P., Johnson. E. R., Hafiz. M.:REST and Web Services: In Theory and in Practice. In: REST: From Research to Practice. Springer Science+Business Media, 2011, kapitola 2.

[13] Richardson. L., Ruby. S.: RESTful Web Services: Web Services for the Real World. 1. vyd. O'Reilly Media, Inc., May 2007. ISBN – 0-596-052926-0.

[14] Medjahed. B., Bouguettaya. A., Elmagarmind K. A.: Composing Web services on the Semantic Web. In: The VLDB Journal — The International Journal on Very Large Data Base. Vol 12. Issue 4. USA: Springer, 2003, s. 333-351.

[15] Liu. L., Lecue. F., Mehandjiev. N.: Semantic content-based recommendation of software services using context. In ACM Transactions on the Web, Volume 7 Issue 3, 2013. ISSN 978-1-4503-0784-0.

[16] Na. Y., Jian. L., Yinliang. Z.: A Context-based Framework for Web Services Discovery. In Advanced Computer Control, Harbin, Volume 3, 2011. s. 238 – 241. E-ISBN 978-1-4244-8810-0.

[17] Neiat. G. A., Mohsenzadeh. M., Forsati. R., Rahmani. M. A.: An Agent-based Semantic Web Service Discovery Framework. In Computer Modeling and Simulation, Macau, 2009. s. 194 – 198. E-ISBN 978-1-4244-3561-6.

[18] SPRING. http://spring.io/ , 3.12.2014

[19] Apache Software Foundation.: Apache JENA. https://jena.apache.org/ , 3.12.2014

[20] OWL API. http://owlapi.sourceforge.net/, 29.4.2015

[21] Derivo Semantic Systems.: SPARQL-DL. http://www.derivo.de/en/resources/sparql-dl-api.html , 3.12.2014

[22] Klusch. B., Khalid. A. M., Vasileski . M., Kapahnke. P.: OWLS-TC. http://projects.semwebcentral.org/projects/owls-tc/ , 3.12.2014

[23] X.Chen., Z. Zheng., X. Liu., Z. Huang., H. Sun.: Personalized QoS-Aware Web Service Recommendation and Visualization . In IEEE Transactions on services computing. Volume 6. No. 1., 2013

[24] R. Gupta., S. K. Malik.: SPARQL semantics and execution analysis in Semantic Web using Various Tools. In Communication Systems and Network Technologies, Katra, Jammu, 2011. s. 278 – 282. E-ISBN 978-0-7695-4437-3 .

[25] Derivo Semantic Systems. http://www.derivo.de/en/resources/sparql-dl-api/ 9.12.2014.

[26] Tang. M., Jiang. Y., Liu. J., Liu. X.: Location-Aware Collaborative Filtering for QoS-Based Service Recommendation. In Web Services (ICWS), s. 202-209. Honolulu. 2012

[27] Al-Moayed. A., Hollunder, B.: Quality of Service Attributes in Web services. In Software Engineering Advances (ICSEA). s. 367 – 372. Nice. 2010

[28] The Semantic Web Service Matchmaker Evaluation Enviroment: SME v2.2. http://www.semwebcentral.org/projects/sme2/ , 29.4.2015

[29] Klush. M., Kapahnke. P.: The iSeM matchmaker: A flexible approach for adaptive hybrid semantic service selection. In Web Semantics: Science, Services and Agents on the World Wide Web. Volume 15. s. 1-14, 2012

[30] w3layouts.com, http://w3layouts.com/preview/?l=/min-app-a-mobile-app-based-flat-bootstrap-responsive-web-template/ 11.5.2015

9. Zoznam použitých skratiek

SOA	**S**ervice **O**riented **A**rchitecture – Servisne Orientovaná Architektúra, podrobnejšie opísaná v kapitole 2.
RPC	**R**emote **P**rocedure **C**all – Vyvolanie vzdialenej metódy.
HTTP	**H**yper**T**ext **T**ransfer **P**rotocol - Hypertextový prenosový protokol.
TCP/IP	**T**ransmission **C**ontrol **P**rotocol/**I**nternet **P**rotocol - Primárny Prenosový Protokol/Protokol Sieťovej Vrstvy.
WS	**W**eb **S**ervice – Webová služba, podrobnejšie opísaná v kapitole 2.5.
Služba	Skratka termínu softvérová služba, podrobnejšie opísaná v kapitole 2.4.
XML	e**X**tensible **M**arkup **L**anguage – rozšíriteľný popisovací jazyk, podrobnejšie opísaný v kapitole 2.5.
SOAP	**S**imple **O**bject **A**ccess **P**rotokol – protokol pre jednoduchý prístup ku objektom. Popísaný v kapitole 2.5.2.
WSDL	**W**eb **S**ervice **D**escription **L**anguage – Jazyk pre popis webovej služby, podrobnejšie opísaný v kapitole 2.5.3.
REST	**Re**presentational **S**tate **T**ransfer – druh implementácie služieb cez internet, podrobnejšie popísaný v kapitole 2.6.
URI	**U**niform **R**esource **I**dentifier.
JSON	**J**ava**S**cript **O**bject **N**otation – Objekt v Javascript notácií.
OWL	**W**eb **O**ntology **L**anguage – jazyk pre vytváranie ontológií na webe.
OWL-S	**W**eb **O**ntology **L**anguage for **S**ervices - jazyk pre vytváranie ontológií služieb, podrobnejšie opísaný v kapitole 2.8.2.

WSMO	**W**eb **S**ervice **M**odeling **O**ntology – ontológia pre modelovanie webových služieb.
CF	**C**ollaborative **F**iltering - filtrovanie podľa kolaborácie, podrobnejšie opísané v kapitole 3.2.1.
Agent	**S**oftvérový **A**gent – softvér ktorý umožňuje automatizáciu kompozície a odporúčanie služieb, podrobnejšie opísaný v kapitole 3.4.
SQL	**S**tructured **Q**uery **L**anguage – štruktúrovaný jazyk pre prácu s relačnou databázou.
SPARQL	**S**PARQL **P**rotocol and **R**DF **Q**uery **L**anguage – jazyk pre vykonávanie dopytov nad sémantickými dátami v RDF formáte.
SPARQL-DL	Jazyk pre vykonávanie dopytov nad sémantickými dátami, schémami a ontológiami.
RDF	**R**esource **D**escription **F**ramework – W3C špecifikácia pre popis sémantických dát.
GCA	**G**eneral **C**ontext **A**ttribute – atribút všeobecného kontextu ktorý sa využíva pri modifikácií kontextu.
MCA	**M**odified **C**ontext **A**ttribute – atribút modifikovaného kontextu ktorý sa využíva pri modifikácií kontextu.
EJB	**E**nterprise **J**ava**B**eans – spravovaný komponent na strane servera pre modulárnu konštrukciu enterprise projektov.
JPA	**J**ava **P**ersistence **A**PI – štandardné rozhranie ktoré popisuje spravovanie dát v relačných databázach.
JSF	**J**ava**S**erver **F**aces – štandardná špecifikácia pre vývoj komponentového používateľského rozhrania pre web stránky.
W3C	World Wide Web Consortium - konzorcium produkujúce štandardy pre web
QoS	**Q**uality **o**f **S**ervice – označuje celkovú výkonnosť siete z pohľadu jej používateľov.

SA-WSDL	**S**emantic **A**nnotations for **WSDL** and XML Schema – súbor definícií ktorý umožňuje popis sémantiky vo webových službách definovaných pomocou WSDL.
MySQL	Svetovo najpoužívanejšia voľná relačná databáza.
UML	**U**nified **M**odeling **L**anguage – grafický jazyk pre vizualizáciu a špecifikáciu softvérových systémov.
RDFS	**R**esource **D**escription **F**ramework **S**chema – W3C špecifikácia pre hierarchiu RDF znalostí.
QoS	**Q**uality **o**f **S**ervices – atribúty popisujúce nefunkcionálne požiadavky služieb
MVC	**M**odel – **V**iew – **C**ontroller – architektonický štýl štruktúry systému
TP	**T**rue **p**ositive – trieda služby ktorá mala byť odporučená a bola odporučená
FP	**F**alse **p**ositive – trieda služby ktorá nemala byť odporučená a bola odporučená
TN	**T**rue **n**egative – trieda služby ktorá nemala byť odporučená a nebola odporučená
FN	**F**alse **n**egative – trieda služby ktorá mala byť odporučená a nebola odporučená.

10. Prílohy

V práci sa nachádzajú nasledovné prílohy:

Príloha A. Technická dokumentácia
Príloha B. Používateľská príručka
Príloha C. Inštalačná príručka
Príloha D. Obsah elektronického média
Príloha E. Výskumný článok

10.1. Technická dokumentácia

10.1.1. Popis systému

Systém na odporúčanie služieb je vytvorený ako webová aplikácia. Uchováva si informácie o používateľoch, službách ktoré má prístupné a o uskutočnených odporúčaniach. Jeho funkcionalita je prístupná cez webový prehliadač, ale je ju možné vyvolať aj cez REST služby. Systém obsahuje vlastný autentifikačný a autorizačný modul. Funkcionalita je implementované v jazyku Java s využitím štandardných technológií pre vývoj Enterprise projektov. Odporúčací systém beží na aplikačnom a webovom serveri JBoss.

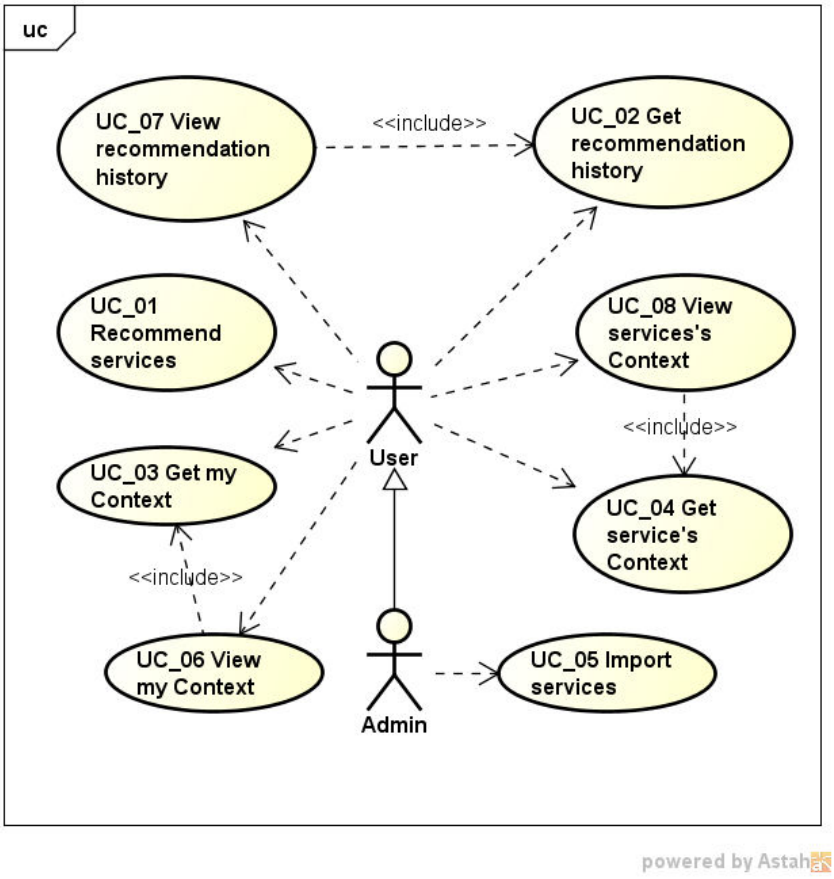
V systéme je implementované logovanie za pomoci framework-u Apache Log4j, ktoré informuje o funkcionalite systému.

10.1.2. Roly používateľov v systéme a prípady použitia

V systéme sú definované dva druhy používateľov:

1. Používateľ – akýkoľvek používateľ ktorý je zaregistrovaný v systéme.
2. Administrátor – správca systému ktorý má okrem funkcionality prístupnej používateľovi aj možnosť pridávania nových služieb do systému.

Diagram prípadov použitia zobrazujúci jednotlivých používateľov spolu s ich prípadmi použitia je zobrazený na obrázku 12. Funkcionalitu ktorá je prístupná používateľovi môže používateľ využiť buď cez webové rozhranie, alebo pomocou volaní REST služieb. Funkcionalita nahrávania služieb do systému je pre administrátora prístupná len cez webové rozhranie. V systéme sú definované nasledovné prípady použitia:



Obrázok 12 Diagram prípadov použitia zobrazujúci role v systéme spolu s ich prípadmi použitia.

Prípad použitia: UC_01 Recommend services

Prihlásený používateľ môže zaslať požiadavku o službu a systém mu služby odporučí.

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi sú odporučené služby
Účastníci	Používateľ
Opis	Používateľ vyžiada služby a systém mu ich odporučí

Hlavný tok : Odporuč služby

1. Účastník zvolí funkcionalitu výberu služieb.
2. Účastník zadá parametre požiadavky:
   - Požiadavku na funkcionalitu služby vo forme ontológie
   - Požadované atribúty kontextu služby ceny a domény
   - Váhy pre ohodnotenie jednotlivých zložiek výslednej vhodnosti
3. Účastník zašle požiadavku.
4. Systém vyhodnotí požiadavku a odporučí služby.
5. Používateľovi sú služby odporučené.
6. Prípad použitia končí.

Alternatívny tok : Zle špecifikované váhy

1. Tok sa aktivuje v kroku 4 hlavného toku v prípade, že súčet váh špecifikovaných používateľom sa nerovná trom.
2. Systém vráti chybu s popisom o zle zadaných váhach.
3. Prípad použitia končí.

Prípad použitia: UC_02 Get recommendation history

Prihlásený používateľ si vyžiada informácie o službách ktoré mu boli odporučené a systém mu ich vráti (pokiaľ existujú).

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi sú vrátené jeho služby na zobrazenie
Účastníci	Používateľ
Opis	Používateľ vyžiada jemu odporučené služby a systém mu ich vráti

Hlavný tok : Vráť odporučené služby

1. Účastník zvolí funkcionalitu vrátenia odporučených služieb.
2. Účastník môže zadať nasledovné parametre:
   - Časový úsek v ktorom mu boli služby odporučené (pokiaľ nezadá, sú mu vrátené všetky jemu odporučené služby)
   - Či majú byť vrátené len služby ktoré využil
   - Či majú byť vrátené len služby s ktorými bol spokojný
3. Účastník zašle požiadavku.
4. Systém nájde všetky služby podľa zadaných parametrov a zoradí ich od najnovších odporúčaní po najstaršie.
5. Systém vráti používateľovi nájdené odporučené služby.
6. Prípad použitia končí.

Alternatívny tok : Neexistujú odporučené služby

1. Tok sa aktivuje v kroku 4 hlavného toku v prípade, že používateľ nemá odporučené služby.
2. Systém vráti informáciu o tom, že používateľ nemá odporučené služby.
3. Prípad použitia končí.

Prípad použitia: UC_03 Get my Context

Prihlásený používateľ si vyžiada informácie o jeho kontexte a systém mu ich vráti (spolu s jeho všetkými kontextami z jeho histórie).

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi je vrátený jeho kontext
Účastníci	Používateľ
Opis	Používateľ vyžiada jeho kontext a systém mu ho vráti

Hlavný tok : Vráť môj kontext

1. Účastník zvolí funkcionalitu vrátenia jeho kontextu
2. Účastník môže zadať nasledovné parametre:
   - Časový úsek v ktorom mu boli služby odporučené (pokiaľ nezadá, sú mu vrátené všetky jemu odporučené služby)
   - Či majú byť vrátené len služby ktoré využil
   - Či majú byť vrátené len služby s ktorými bol spokojný
3. Účastník zašle požiadavku.
4. Systém nájde všetky služby podľa zadaných parametrov a zoradí ich od najnovších odporúčaní po najstaršie.
5. Systém vráti používateľovi nájdené odporučené služby.
6. Prípad použitia končí.

Alternatívny tok : Neexistujúci kontext používateľa

1. Tok sa aktivuje v kroku 4 hlavného toku v prípade, že používateľ nemá definovaný kontext.
2. Systém vráti informáciu o tom, že používateľ nemá definovaný kontext.
3. Prípad použitia končí.

Prípad použitia: UC_04 Get service‘s Context

Prihlásený používateľ si vyžiada informácie o kontexte vybranej služby a systém mu ich vráti (spolu so všetkými kontextami z histórie danej služby).

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi je vrátený kontext služby
Účastníci	Používateľ
Opis	Používateľ vyžiada kontext služby a systém mu ho vráti

Hlavný tok : Vráť kontext služby

1. Účastník zvolí funkcionalitu vrátenia kontextu služby.
2. Účastník vyberie službu zo zoznamu dostupných služieb.
3. Účastník zašle požiadavku.
4. Systém nájde vybranú službu a získa všetky jej kontexty.
5. Systém vráti používateľovi nájdené kontexty služby
6. Prípad použitia končí.

Alternatívny tok : Neexistuje kontext služby

1. Tok sa aktivuje v kroku 4 hlavného toku v prípade, že služba nemá definovaný kontext
2. Systém vráti informáciu o tom, že služba nemá definovaný kontext.
3. Prípad použitia končí.

Prípad použitia: UC_05 Import services

Administrátor môže pridať do systému webové služby, ktoré budú prístupné na odporúčanie.

Predpoklady	Administrátor je prihlásený v systéme
Dôsledky	Služba je importovaná do systému.
Účastníci	Administrátor
Opis	Administrátor zadá polohu služby a jej kontext.

Hlavný tok : Pridať službu do systému

1. Účastník zvolí funkcionalitu importu služby.
2. Účastník zadá adresu služby a atribúty kontextu služby (pokiaľ služba kontext má). Pokiaľ zadáva kontext, tak účastník aj špecifikuje či má byť zadaný kontext nemenný.
3. Účastník zašle požiadavku na import služby.
4. Systém získa službu s dodanej adresy a spracuje ju. Pokiaľ je zadaný kontext, tak ho uloží a priradí ku službe.
5. Systém účastníka informuje o úspešnom importe služby.
6. Prípad použitia končí.

Prípad použitia: UC_06 View my Context

Systém zobrazí používateľovi jeho kontext (spolu s jeho všetkými kontextami z jeho histórie).

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi je zobrazený jeho kontext
Účastníci	Používateľ
Opis	Používateľ vyžiada jeho kontext a systém mu ho zobrazí

Hlavný tok : Zobrazenie kontextu

1. Účastník zvolí funkcionalitu zobrazenia jeho kontextu.
2. Vyvolá sa prípad použitia UC_03 Get my Context.
3. Systém účastníkovi zobrazí jeho kontextuálnu informáciu získanú v predošlom kroku.
4. Prípad použitia končí.

Prípad použitia: UC_07 View recommendation history

Systém zobrazí používateľovi služby ktoré mu boli v histórií odporučené.

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi sú zobrazené jemu odporučené služby.
Účastníci	Používateľ
Opis	Používateľ vyžiada jemu odporučené služby a systém mu ich zobrazí

Hlavný tok : Zobrazenie odporučených služieb

1. Účastník zvolí funkcionalitu zobrazenia jemu odporučených služieb.
2. Vyvolá sa prípad použitia UC_02 Get recommendation history.
3. Systém účastníkovi zobrazí jemu odporučené službyzískané v predošlom kroku.
4. Prípad použitia končí.

Prípad použitia: UC_08 View service‘s Context

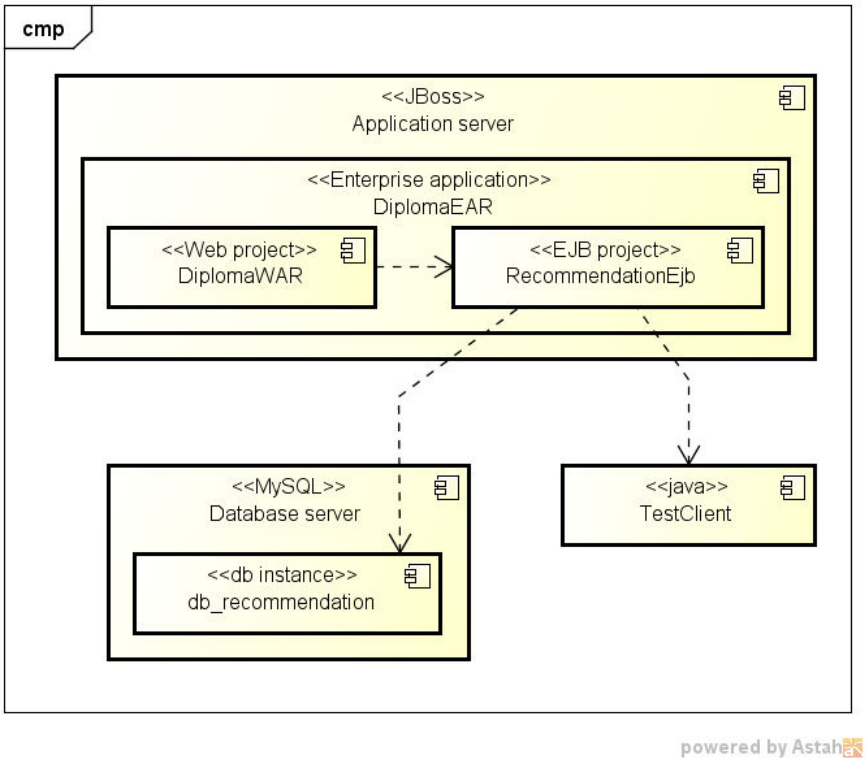
Systém zobrazí používateľovi kontext vybranej služby (spolu so všetkými kontextami služby z jej histórie).

Predpoklady	Používateľ je prihlásený v systéme
Dôsledky	Používateľovi je zobrazený kontext služby
Účastníci	Používateľ
Opis	Používateľ vyžiada kontext služby a systém mu ho zobrazí

Hlavný tok : Zobrazenie kontextu služby

1. Účastník zvolí funkcionalitu zobrazenia kontextu služby.
2. Vyvolá sa prípad použitia UC_04 Get service‘s Context.
3. Systém účastníkovi zobrazí kontextuálnu informáciu služby získanú v predošlom kroku.
4. Prípad použitia končí.

10.1.3. Komponentový návrh



Obrázok 13 Diagram nasadenia systému.

Systém je nasadený na aplikačnom serveri JBoss a je tvorený Enterprise aplikáciou „DimplomaEAR“. Aplikácia sa skladá z dvoch projektov – DiplomaWAR a RecommendationEjb.

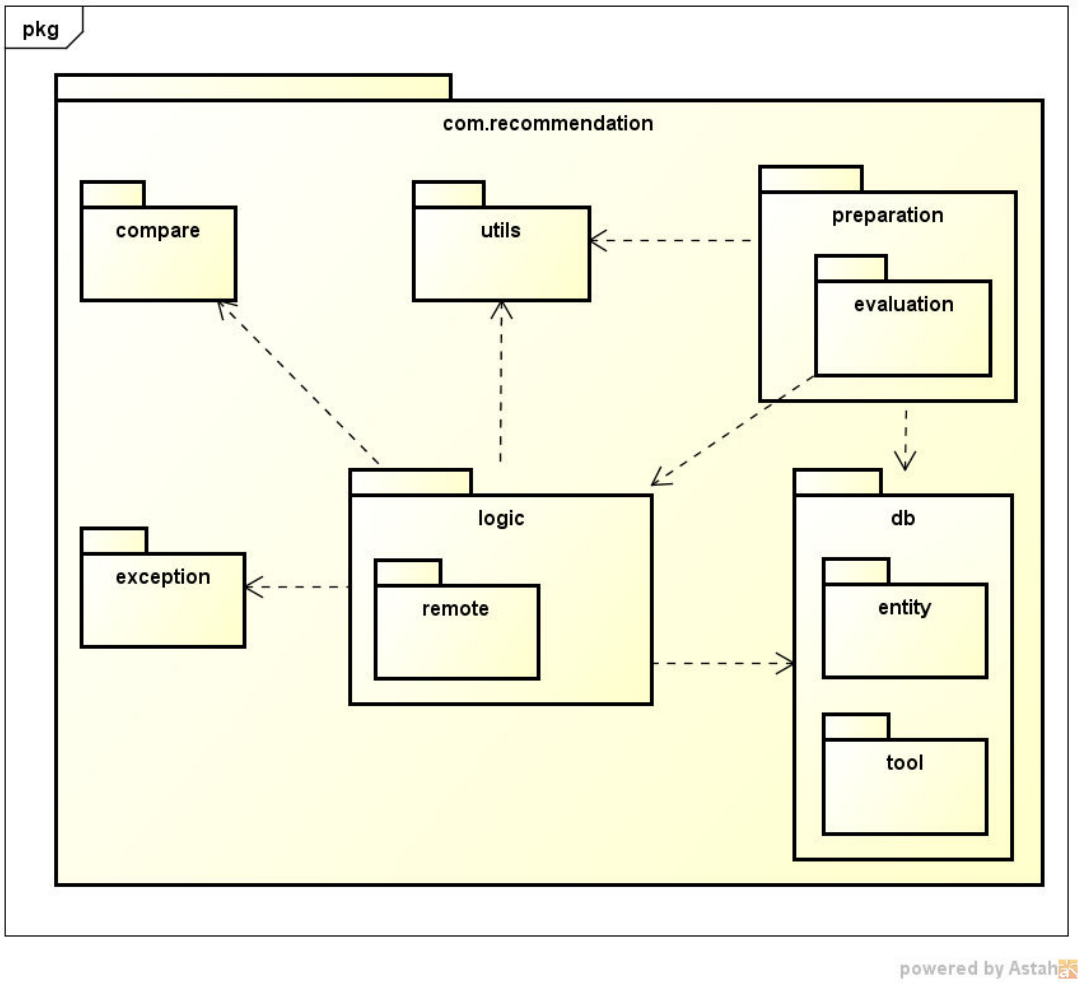
DiplomaWAR predstavuje webový projekt cez ktorý je prístupná funkcionalita systému. RecommendationEjb projekt obsahuje aplikačnú logiku odporúčania implementovaný za použitia technológie EJB. Pri odporúčaní sa využíva databáza „db_recommendation“, ktorá sa nachádza na MySQL databázovom serveri.

Ku systému je dodaný taktiež aj jednoduchý Java klient „TextClient“, ktorý slúži na rýchle otestovanie funkcionality odporúčania a overenie systému. Komponent bol využívaný pri vývoji.

Diagram balíkov zobrazujúci štruktúru balíkov v projekte ktorý obsahuje aplikačnú logiku je zobrazený na obrázku 14. Štruktúra je vnorená do balíka „com.recommendation“ a je nasledovná:

- *compare* – obsahuje triedy pre výpočet jednotlivých vhodností a nástroj pre tvorbu všeobecného kontextu a početnosti všeobecného kontextu
- *utils* – obsahuje pomocné nástroje pre prácu so službami, kontextami a používateľmi
- *exception* – obsahuje definície vlastných výnimiek definovaných v systéme
- *db* – obsahuje triedy asociované s databázou
    - *entity* – obsahuje JPA entity

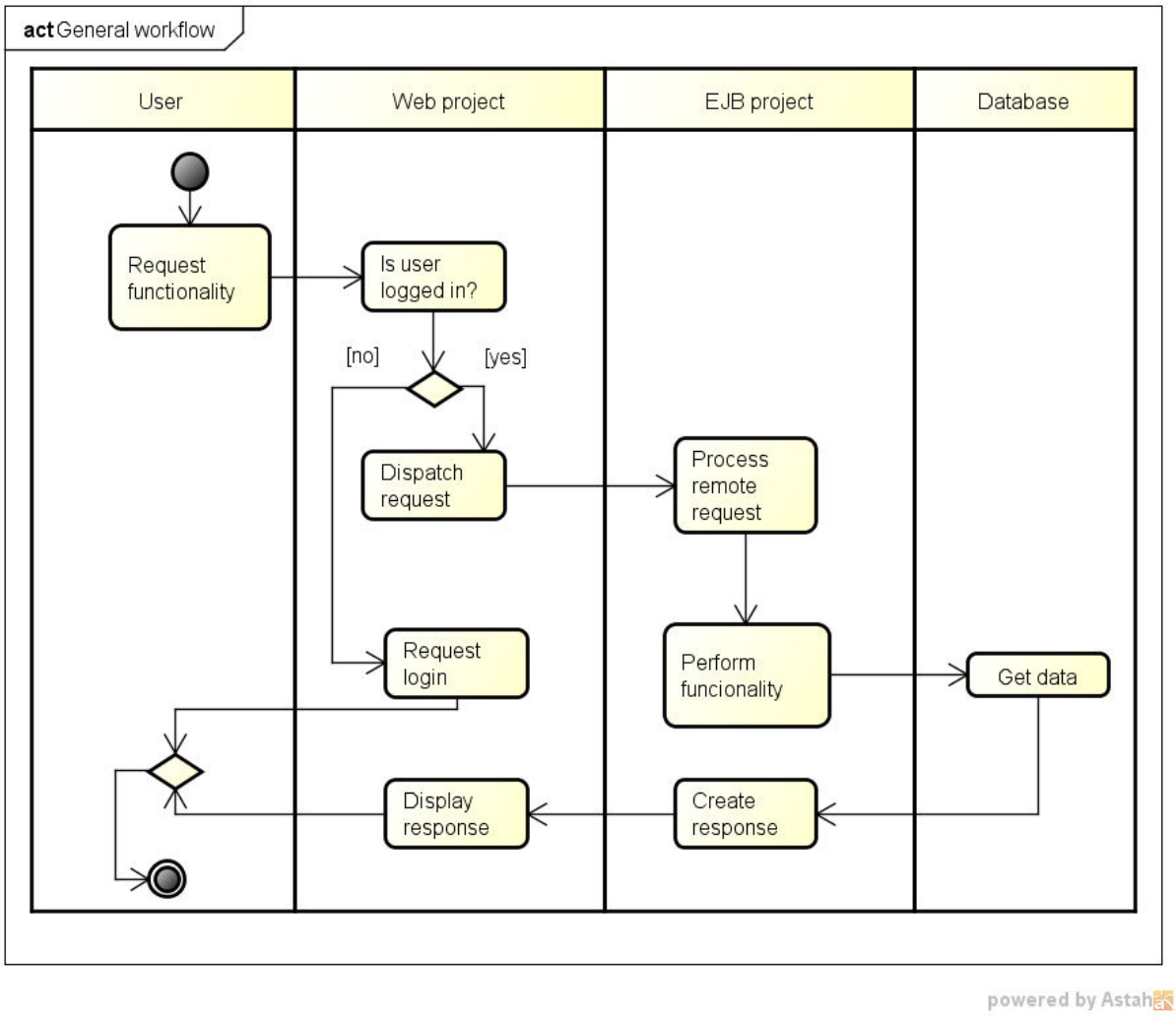
    - *tool* – obsahuje nástroje pre prácu s databázou
- *logic* – obsahuje triedy zodpovedné za funkcionalitu odporúčania a definíciu váh použitých v systéme
    - *remote* – obsahuje triedy cez ktoré je funkcionalita balíka *logic* prístupná mimo projektu RecommendationEjb
- *preparation* – obsahuje triedy pre importovanie a generovanie používateľov, služieb a ich kontexty
    - *evaluation* – obsahuje triedy ktoré boli využité pri overovaní



Obrázok 14 Diagram balíkov v projekte RecommendationEjb.

10.1.4. Popis funkcionality

Implementovaná funkcionalita odporúčania služieb je popísaná v kapitole 4.4. Všetky požiadavky používateľa sú vykonávané na základe abstraktnej schémy popísanej na obrázku 15. Pre vykonávanie akejkoľvek funkcionality musí byť používateľ prihlásený, z toho dôvodu sa najskôr kontroluje či je prihlásený. Ak áno, tak sa jeho požiadavka ďalej spracováva, inak je vyzvaný na prihlásenie. Pokiaľ je používateľ prihlásený, tak webový komponent pošle požiadavku do EJB komponentu kde je požiadavka rozbalená a odoslaná do internej aplikačnej logiky. Tu sa za pomoci dát v databáze požiadavka vyrieši a pripraví odpoveď, ktorá je odoslaná do webového komponentu. V ňom sa odpoveď zobrazí používateľovi.



Obrázok 15 Diagram aktivít zobrazujúci všeobecné tok spracovania pri vykonávaní požiadavky.

10.1.5. Popis aplikačnej logiky

Všetky triedy, metódy a atribúty tried sú popísané za pomocou technológie Javadoc. Hlavné triedy ktoré sú popísané nižšie:

Aplikačné logika je uložená v EJB komponentoch. V práci sú definované dve rozhrania ktoré slúžia pre vyvolanie aplikačnej logiky.

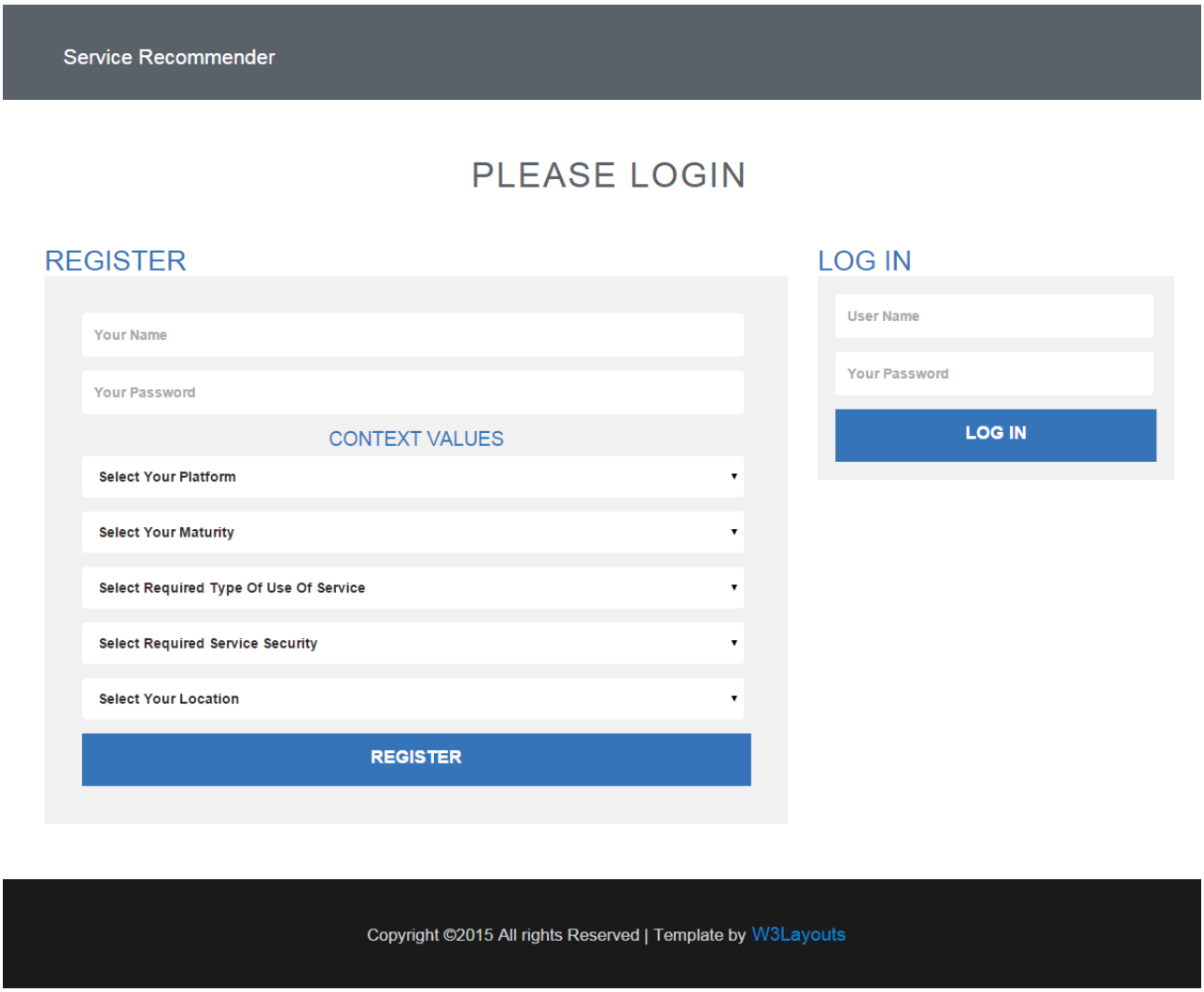
1. *ContextGenerator* – toto rozhranie implementuje trieda *ContextGeneratorBean*. Toto rozhranie definuje metódy ktoré sú potrebné pre prípravu na odporúčanie (príprava je bližšie popísaná v kapitole 4.4.1). Sú to nasledovné metódy:
    a. *generateServicesContext(int contextPercentage, int immutablePercentage, int contextItemPercentage)* – slúži na vygenerovanie kontextu pre služby uložené v systéme. Parametre (v poradí deklarovanom) slúžia na definovanie percenta služieb ktorým má byť vygenerovaný kontext, percenta služieb ktoré majú mať kontext nemenný a percentuálne zastúpenie atribútov v kontexte.
    b. *storeServices(String bankHome)* – slúži na uloženie všetkých služieb ktoré sa nachádzajú na danej adrese do systému, pričom sa z nich za pomoci SPARQL extrahujú tie triedy ontológií, ktoré služby využívajú.
    c. *generateUsers(int usersCount)* – slúži na vygenerovanie daného počtu používateľov.
    d. *generateUsersContext(int contextItemPercentage*) – slúži na vygenerovanie kontextu pre používateľov systému. Parameter slúži na definovanie percenta výskytu atribútov v kontexte používateľov.
2. *ServiceRecommender* – toto rozhranie implementuje trieda *ServiceRecommenderBean*. Rozhranie definuje metódy ktoré slúžia na odporúčanie služieb. Sú to nasledovné metódy:
    a. *proposeServiceList(ServiceRequest service)* – slúži na samotné odporúčanie služieb. Parametrami (v poradí deklarovanom) sú adresa na ontológiu ktorá definuje sémantickú požiadavku na službu, ID používateľa, požiadavka na doménu do ktorej má služba spadať a váhy ktoré majú byť využité v odporúčaní.
    b. *rateRecommendedServices(List<RecommendedService> recommendedServices)* – slúži na spracovanie zoznamu služieb ktorý bol používateľovi odporúčaný a z ktorého používateľ vybral a ohodnotil služby.
3. *ServiceRecommenderRemote* – rozhranie implementuje trieda *ServiceRecommenderRemoteBean* a slúži na prístup ku funkcionalite odporúčania zo vzdialených zdrojov. Rozhranie definuje nasledovné metódy:
    a. *proposeServiceList(int userId, int domainRequest, int priceRequest, String ontologyRequest, Weight weight)* - slúži na samotné odporúčanie služieb. Parametrami (v poradí deklarovanom) sú ID používateľa, požiadavka na doménu

do ktorej má služba spadať, požiadavka na cenu služby, adresa na ontológiu ktorá definuje sémantickú požiadavku na službu a váhy ktoré majú byť využité v odporúčaní.

b. *rateRecommendedServices(List<RecommendedService> recommendedServices)* – slúži na spracovanie zoznamu služieb ktorý bol používateľovi odporúčaný a z ktorého používateľ vybral a ohodnotil služby.

10.2. Používateľská príručka

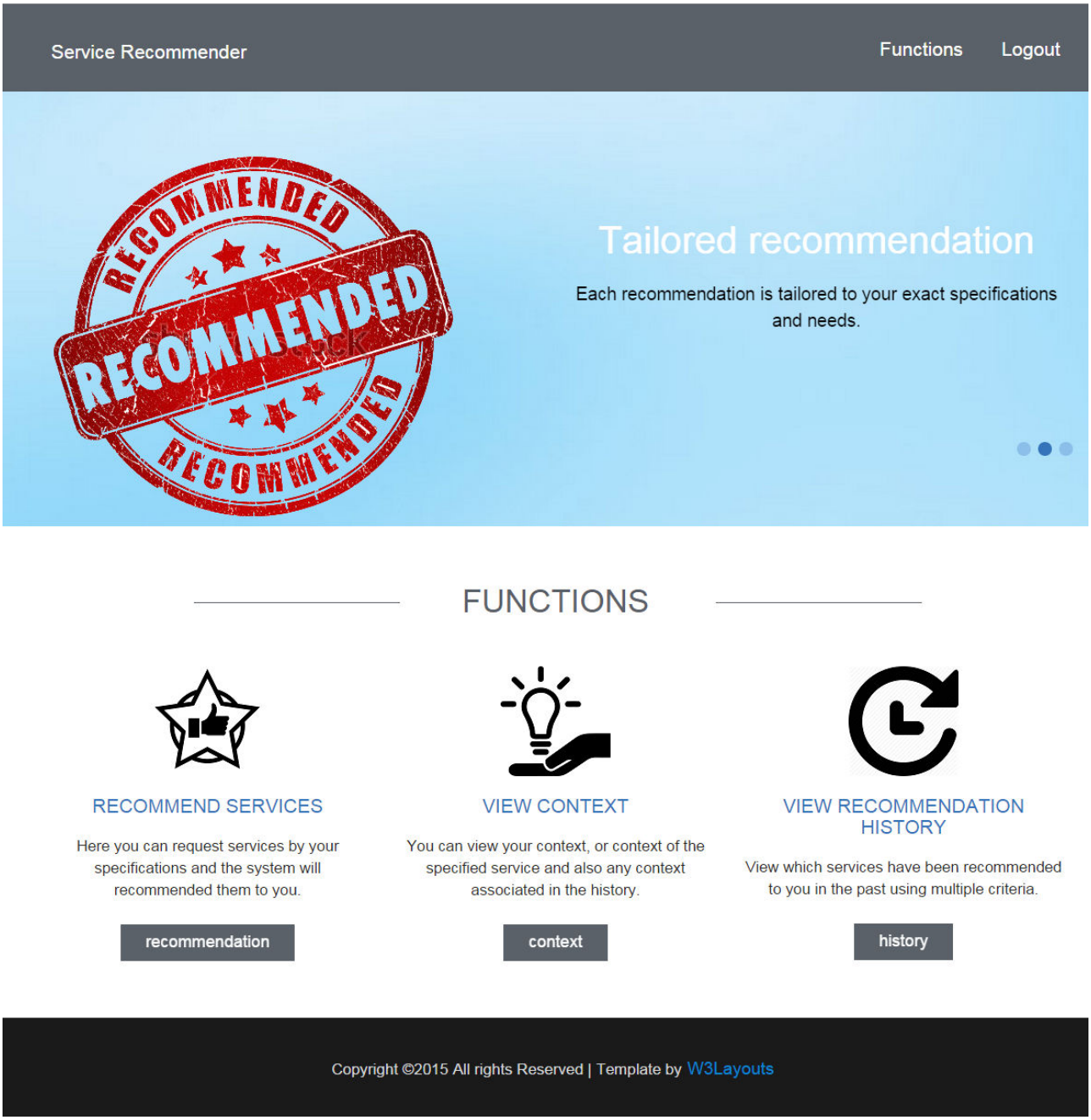
Pre využívanie funkcionality systému je nutné aby bol používateľ prihlásený. Funkcionalitu systému je taktiež možné využívať aj cez REST služby. Pokiaľ nie je používateľ prihlásený, tak je vždy presmerovaný na prihlasovaciu stránku (prípadne pri vyvolaní funkcionality cez REST mu je vrátená chybová hláška):



V pravom rohu môže používateľ zadať svoje prihlasovacie údaje a následne pre prihlásenie musí stlačiť tlačidlo „LOG IN“. Na tejto stránke sa používateľ môže taktiež zaregistrovať. Pri registrácii musí používateľ špecifikovať svoje prihlasovacie meno a heslo a môže taktiež zadať aj atribúty svojho kontextu a následne pre registráciu stlačiť tlačidlo „REGISTER“.

Po prihlásení je používateľ presmerovaný na hlavú stránku. V pravom hornom rohu všetkých stránok (okrem stránky pre prihlásenie) sa nachádzajú odkazy pre odhlásenie zo systému („Logout“) a pre navigáciu na funkcionalitu systému („Functions“). Pokiaľ je prihlásený používateľ Administrátor, má v hornom rohu taktiež možnosť navigácie na stránku importu

služby cez odkaz („Import Service“), iný používateľ ako Administrátor túto funkcionalitu prístupnú nemá.



Na hlavnej stránke je možné vidieť logo systému a v dolnej časti zoznam ponúkaných funkcionalít prístupných cez odkazy:

- odporúčanie služieb („recommendation)
- zobrazenie svojho kontextu a kontextu služby („context“)
- zobrazenie histórie odporúčaní („history“)

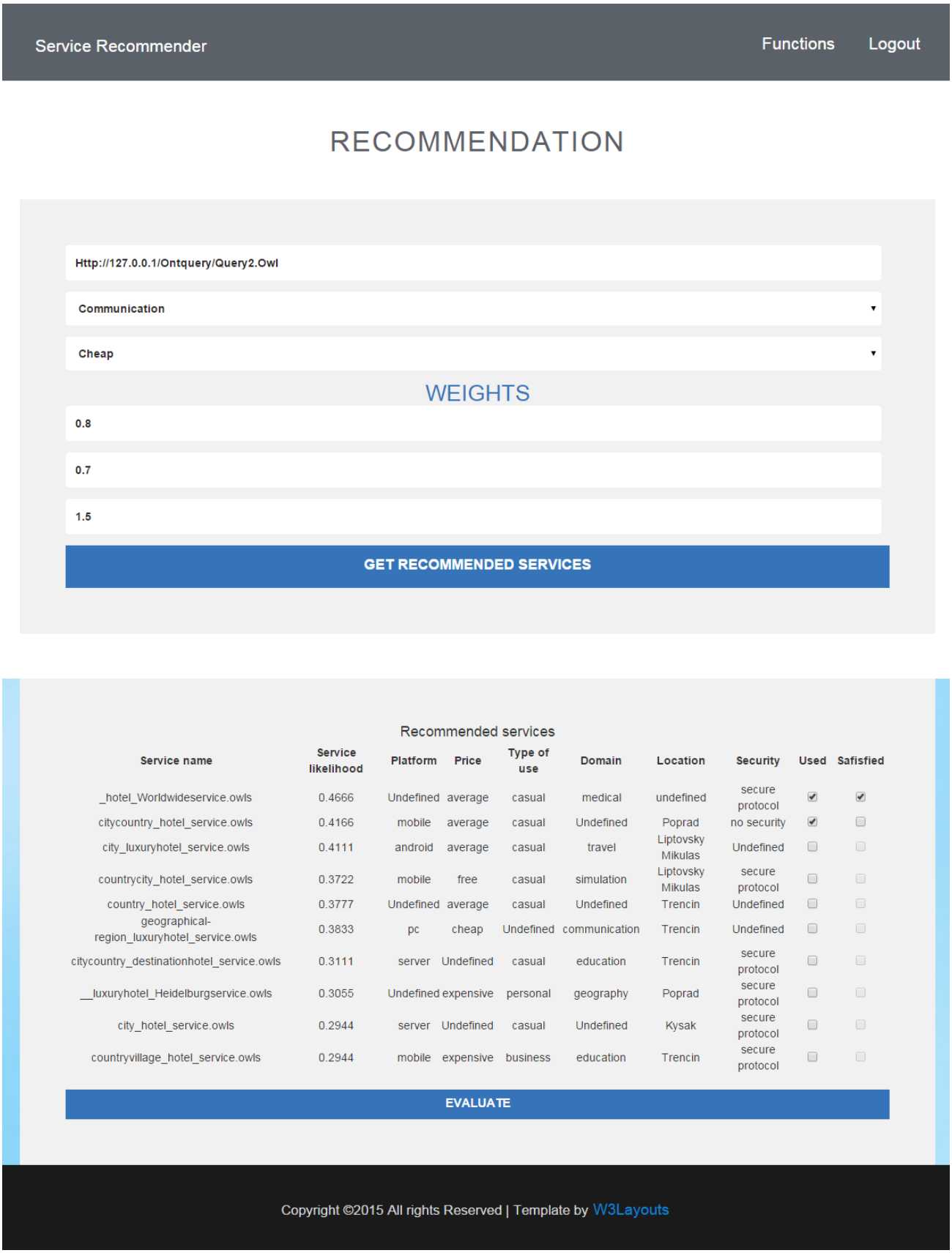
Na stránke pre odporúčanie má používateľ prístupnú funkcionalitu odporúčania služieb. Pre odporúčanie musí zadať odkaz na ontológiu ktorá obsahuje funkcionálnu požiadavku na službu a taktiež aj požadované atribúty kontextu služby – domain a price.
Pre vyvolanie funkcionality cez REST musí vykonať HTTP POST požiadavku na „/recommend“, pričom v tele požiadavky špecifikuje požadované parametre v JSON objekte:

```
{
  "action": "recommend",
  "price": 2,
  "domain": 2,
  "ont": "http://127.0.0.1/ontquery/query2.owl",
  "w_sem": 1,
  "w_clb": 1,
  "w_ctx": 1
}
```

Odpoveďou je taktiež JSON objekt ktorý obsahuje odporučené služby:

```
{
  "rec_services": [
    {
      "ser_name": "_hotel_Worldwideservice.owls",
      "context": {
        "platform": "Undefined",
        "price": "average",
        "created": "2015-05-10 20:44:00.0",
        "type_of_use": "casual",
        "domain": "medical",
        "security": "secure protocol"
      },
      "ser_likelihood": 0.3333333333333333,
      "rec_id": 361
    },....
  ]
}
```

Používateľ môže taktiež zadať aj jednotlivé váhy. Je nutné aby súčet váh bol 3, v opačnom prípade mu je zobrazená chyba. Pokiaľ váhy nezadá, tak hodnota každej váhy je 1. Pre odporučenie služieb musí stlačiť tlačidlo „GET RECOMMENDED SERVICES“.
Následne sú mu v tabuľke zobrazené odporučené služby. Pokým prebieha odporúčanie, tak kurzor je zmenený na stav „loading“, aby používateľ vedel že proces odporúčania prebieha. Odporučené služby sú zoradené podľa ich vhodnosti. Používateľ môže špecifikovať ktoré služby použil („Used“) a s ktorými bol spokojný („Satisfied“). To či bol so službou spokojný môže špecifikovať až po tom, ako ju označil za použitú. Následne môže odoslať jeho skúsenosť s odporučenými službami pomocou tlačidla „EVALUATE“.


Service Recommender
Functions
Logout
RECOMMENDATION
Http://127.0.0.1/Ontquery/Query2.Owl
Communication
Cheap
WEIGHTS
0.8
0.7
1.5
GET RECOMMENDED SERVICES
Recommended services
Service name
Service likelihood
Platform
Price
Type of use
Domain
Location
Security
Used
Safisfied
_hotel_Worldwideservice.owls 0.4666 Undefined average casual medical undefined secure protocol
citycountry_hotel_service.owls 0.4166 mobile average casual Undefined Poprad no security
city_luxuryhotel_service.owls 0.4111 android average casual travel Liptovsky Mikulas Undefined
countrycity_hotel_service.owls 0.3722 mobile free casual simulation Liptovsky Mikulas secure protocol
country_hotel_service.owls 0.3777 Undefined average casual Undefined Trencin Undefined
geographical-region_luxuryhotel_service.owls 0.3833 pc cheap Undefined communication Trencin Undefined
citycountry_destinationhotel_service.owls 0.3111 server Undefined casual education Trencin secure protocol
__luxuryhotel_Heidelburgservice.owls 0.3055 Undefined expensive personal geography Poprad secure protocol
city_hotel_service.owls 0.2944 server Undefined casual Undefined Kysak secure protocol
countryvillage_hotel_service.owls 0.2944 mobile expensive business education Trencin secure protocol
EVALUATE
Copyright ©2015 All rights Reserved | Template by W3Layouts

Pre vrátenie skúseností so službami cez REST je nutné vyvolať HTTP POST požiadavku na „/recommend“, pričom v tele požiadavky bude špecifikované v JSON objekte:

```
{
  "action": "rate",
  "used_serv": [
    {
      "rec_srvc_id": "381",
      "satisfied": true,
      "tried_out": true
    },
    {
      "rec_srvc_id": "382",
      "satisfied": false,
      "tried_out": true
    }
  ]
}
```

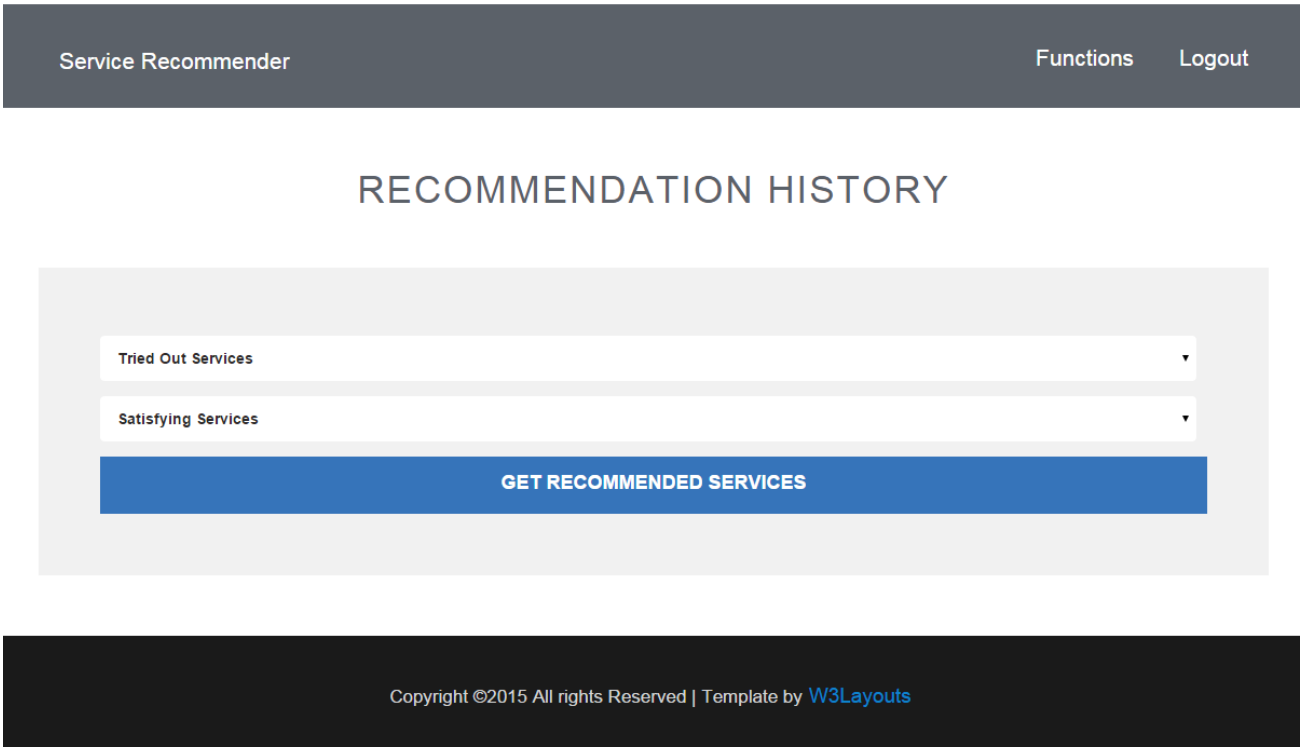
Používateľ môže taktiež prezerať služby ktoré mu boli v histórií odporučené. Toto okno je prístupné z hlavnej stránky. Používateľ môže špecifikovať či mu majú byť vrátené služby s ktorými bol spokojný alebo nespokojný a tie ktoré použil, alebo nepoužil. Pre zobrazenie odporučených služieb musí stlačiť tlačidlo „GET RECOMMENDED SERVICES“. História služieb mu je následne zobrazená v tabuľke pod formulárom v ktorom špecifikoval svoju požiadavku.

Pre získanie odporučených služieb je potrebné zaslať HTTP GET požiadavku na „/recommend_history“, pričom v požiadavke je nutné špecifikovať parametre „satisfied“ a „tried_out“

(napríklad „/recommend_history?satisfied=true&tried_out=true“)

V tele odpovede na požiadavku je JSON objekt obsahujúci nasledovné informácie:

```
{
  "rec_services": [
    {
      "satisfied": true,
      "ont": "http://127.0.0.1/ontquery/query2.owl",
      "tried_out": true,
      "ser_name": "_hotel_Worldwideservice.owls",
      "ser_likelihood": 0.3333333333333333
    },
    {
      "satisfied": true,
      "ont": "http://127.0.0.1/ontquery/query2.owl",
      "tried_out": true,
      "ser_name": "geopolitical-entity_hotel_service.owls",
      "ser_likelihood": 0.3333333333333333
    },...
  ],
  "success": true
}
```



Používateľom je taktiež prístupná funkcionalita zobrazenia ich kontextu a taktiež aj kontextu služby. Toto okno je prístupné z hlavnej stránky. Používateľ môže pre zobrazenie kontextu služby vybrať službu zo zoznamu všetkých služieb ktoré má systém prístupné a následne stlačiť tlačidlo „GET SERVICES CONTEXT“. Pre zobrazenie svojho kontextu musí stlačiť tlačidlo „GET MY CONTEXT“.

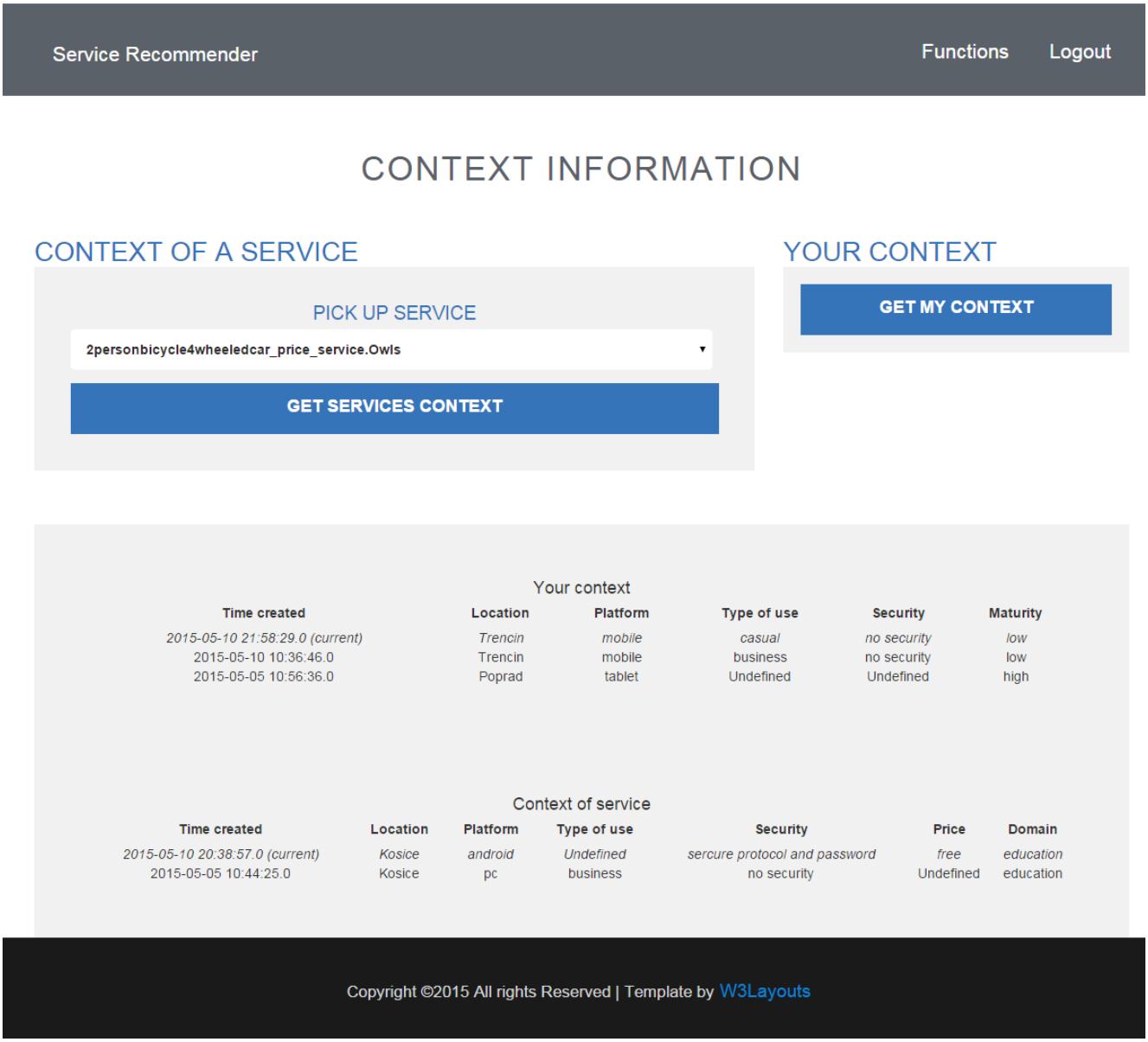
Pre získanie vlastného kontextu cez REST je nutné zaslať HTTP GET požiadavku na „/context?context=user“. V odpovedi je JSON objekt:

```
{
  "context": [
    {
      "platform": "mobile",
      "maturity": "low",
      "created": "2015-05-10 21:58:29.0",
      "type_of_use": "casual",
      "loc": "Trencin",
      "security": "no security"
    },
    {
      "platform": "mobile",
      "maturity": "low",
      "created": "2015-05-10 10:36:46.0",
      "type_of_use": "business",
      "loc": "Trencin",
      "security": "no security"
    }...
  ],
  "success": true
}
```

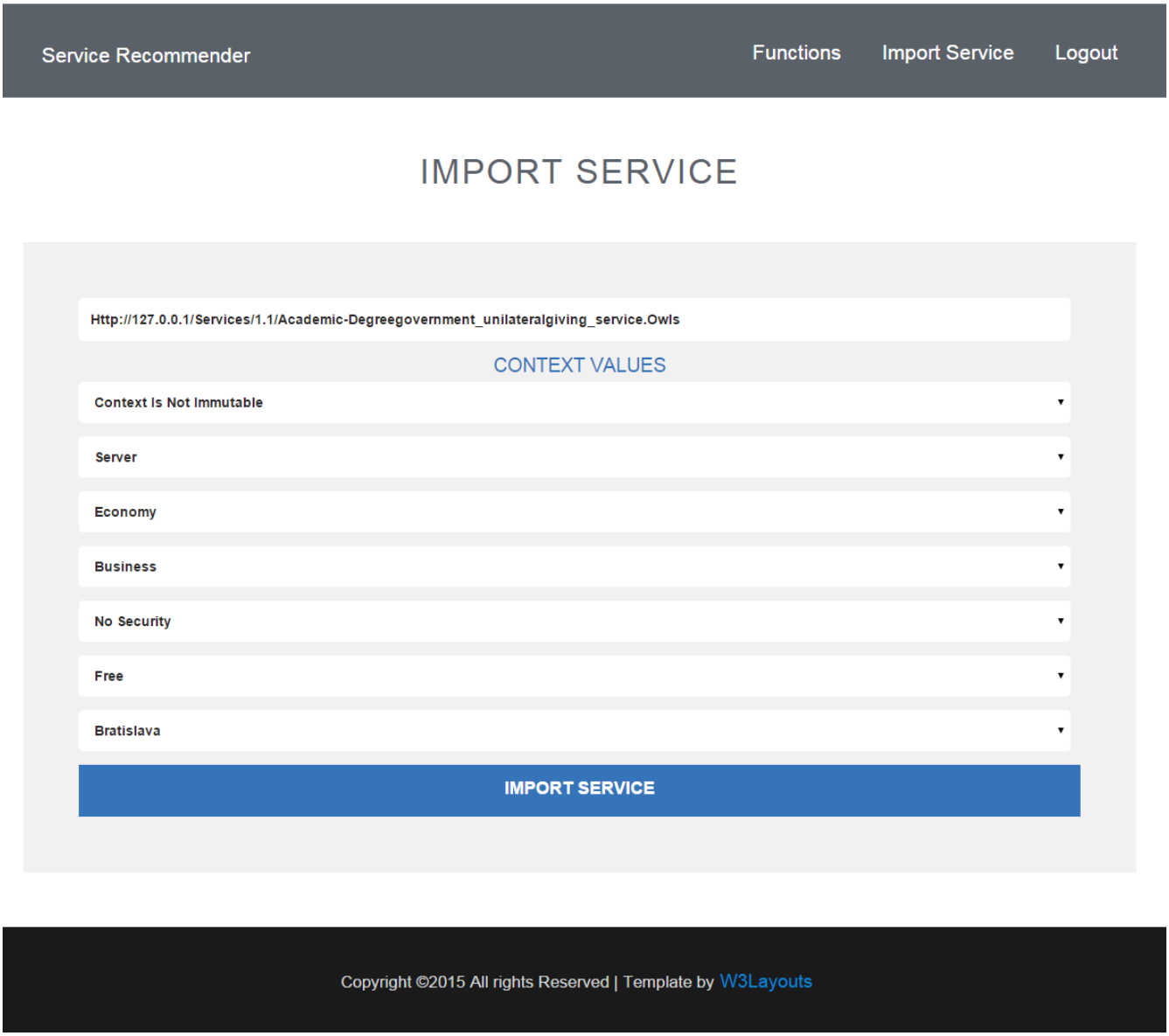

Pre získanie kontextu služby je požiadavka taktiež HTTP GET na „/context?context=service&obj_id=4“, pričom parameter „obj_id“ predstavuje ID služby. Odpoveďou je znova JSON objekt:


```
{
  "context": [
    {
      "platform": "windows",
      "price": "free",
      "created": "2015-05-10 20:43:59.0",
      "type_of_use": "casual",
      "domain": "communication",
      "loc": "Liptovsky Mikulas",
      "security": "secure protocol"
    },
    {
      "platform": "android",
      "price": "average",
      "created": "2015-05-10 20:38:57.0",
      "type_of_use": "personal",
      "domain": "Undefined",
      "security": "no security"
    }...
  ],
  "success": true
}
```


Nižšie v tabuľkách pod formulármi sú požadované kontexty zobrazené, pričom je zobrazená úplná história kontextu danej entity. Kontexty sú zoradené podľa času kedy boli vytvorené, pričom aktuálny kontext je zobrazený navrchu.



Administrátor môže pridať službu do systému cez odkaz ktorý má prístupný z hlavnej stránky. Musí špecifikovať URL služby ktorú chce importovať a môže špecifikovať aj atribúty kontextu importovanej služby. Taktiež môže zadať, či má byť kontext služby nemenný. Po tom ako zadá informácie o službe, môže ju importovať stlačením tlačidla „IMPORT SERVICE“.



Pri všetkých funkcionalitách ktoré systém ponúka, pokiaľ nastane chyba, je používateľovi zobrazené okno ktoré o nej informuje. Pokiaľ nastane chyba pri využívaní funkcionality cez REST, tak je vrátený JSON objekt s nasledovnou štruktúrou:

```
{
  "success": false,
  "errmsg" : "This contains error message describing the problem"
}
```

10.3. Inštalačná príručka

Je potrebné si nainštalovať aplikačný server JBoss (http://www.jboss.org/). Je potrebná aj inštalácia webového servera Apache a MySQL databázy, obe sú obsiahnuté v balíku XAMPP (https://www.apachefriends.org/index.html) .

Súčasťou priloženého média sú zdrojové kódy so všetkými potrebnými knižnicami. Implementácia prebehla vo vývojovom prostredí InteliJ IDEA (https://www.jetbrains.com/idea/download/).

Systém je nakonfigurovaný pre prácu s databázou „db_recommendation“, ktorej štruktúra je dostupná taktiež na priloženom médiu. Databáza má byť prístupná cez štandardný port 3306. Všetky zmeny v pripojení na databázu je potrebné vykonať v súbore *RecommendationEjb/ejbModule/META-INF/persistence.xml*.

Systém je nastavený pre prácu s JTA data-source „*java:/LocalSqlDS*“. Jednotlivé dataSources je možné modifikovať v nastavení servera v súbore *JBOSS_HOME/standalone/configuration/standalone.xml.*

V tomto súbore na ceste *<server>< profile> <subsystem xmlns="urn:jboss:domain:datasources:1.0"> <datasources>* je potrebné pridať nasledovný data-source:

```
<datasource jndi-name="java:/LocalSqlDS" pool-name="LocalSqlDS" enabled="true" use-java-context="true">
        <connection-url>jdbc:mysql://127.0.0.1:3306/db_recommendation</connection-url>
        <driver>mysqlDriver</driver>
        <transaction-isolation>TRANSACTION_READ_COMMITTED</transaction-isolation>
        <pool>
                <min-pool-size>3</min-pool-size>
                <max-pool-size>10</max-pool-size>
                <prefill>true</prefill>
        </pool>
        <security>
                <user-name>DB_USER_NAME</user-name>
                <password> DB_PASSWORD </password>
        </security>
        <statement>
        <prepared-statement-cache-size>32</prepared-statement-cache-size>
                <share-prepared-statements>true</share-prepared-statements>
        </statement>
</datasource>
```

Do zložky *APACHE_WEB_HOME/htdocs* je potrebné vyextrahovať sadu oanotovaných služieb OWL-S TC3 [22].

Po spustení DB servera, web servera a aplikačného servera je možné deploy-núť zdrojové kódy na server a využívať funkcionalitu systému. Pre urýchlené importovanie služieb zo sady OWL-S TC3, odporúčame využitie metód triedy com.recommendation.client.*TestClient* v java aplikácií *RecommendationClient.*

10.4. Obsah elektronického média

Priložené CD obsahuje súbory v nasledovnej štruktúre:

- doc – obsahuje PDF verziu tejto práce a taktiež aj PDF verziu podaného článku
- eval – obsahuje grafy a výsledky z overenia
- src – obsahuje zdrojové súbory
    - bank – obsahuje vyexportovanú banku ontológií a služieb OWL-S TC3
    - db – nachádza sa tu súbor „export.sql“ ktorý obsahuje export dát a štruktúry z MySQL databázy
    - src – obsahuje súbor „export.zip“ v ktorom sa nachádzajú vyexportované projekty so zdrojovým kódom z prostredia InteliJ IDEA

10.5. Výskumný článok (zaslaný na konferenciu ECBS-EERC 2015)

Software services recommendation using context

Ivan Martoš *(Author)*
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies, Slovak University of Technology
Bratislava, Slovakia
martos.ivan@gmail.com

Viera Rozinajová *(Author)*
Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technologies, Slovak University of Technology
Bratislava, Slovakia
viera.rozinajova@stuba.sk

***Abstract*— Nowadays a great number of different software services are available for usage. These services have different attributes and proper selection of software service for solving given problem is very difficult and non-trivial task. This paper focuses on recommendation of software services. We want to achieve selection of software services which fulfills needs of a user to the greatest possible extent. We focus on hybrid approach to recommendation, while using contextual information in the process. Context is used for representing information about each user and service separately. We also propose a method for context refinement with the expectation of retrieving more accurate results in future recommendations.**

Context; software service; semantics; recommendation

- INTRODUCTION

Service-oriented architecture (SOA) is one of the most popular techniques for creating Enterprise applications. SOA is based on using independent units of functionality called services. If a specific problem is too complex to be solved by a single service, it is possible to compose services into compositions so they can provide a solution for the problem, which couldn't be solved by a single service [1].

Nowadays a great number of different software services are available for usage. These services can be composed to create service compositions that fulfill needs of a user [2]. It is much easier and less time consuming to use existing service, than to create a new one. However, with a great number of services, proper selection of software service is very difficult and non-trivial task [3]. This problem can be solved using recommendation. In this paper we focus on solving the problem of service selection by recommendation of services. The proposed approach is based on using context as a parameter for recommendation. We focus on semantic web services, i.e. web services annotated with semantic description.

This paper is structured as follows: in chapter ¯ we explain current approaches for recommendation, in chapter ¯ we define context as it is used in this paper, in chapter ¯ proposed solution is explained and chapter ¯ contains information about implementation details. In chapter ¯ experiments and their evaluation are presented, chapter VII contains a comparison with other approaches. The conclusion and proposals for future work can be found in the last chapter.

- SERVICE RECOMMENDATION AND STATE OF THE ART

As it was mentioned above, there are many services available for usage and selection of proper service is very difficult task. This problem can be solved by recommendation.

There are many approaches for recommendation, generally they can be categorized into three groups [3]:

- *Collaborative filtering approach*
- *Content-based approach*
- *Hybrid approach*

Collaborative filtering approach assumes that similar users have similar demands. If models of users are similar, there is an assumption that their desires will be also similar 0. This approach however faces so called Cold-start problem – problem with insufficient data during the first recommendations. Since there are no data on which similarity could be calculated, it is not possible to perform recommendation [3]. Another disadvantage of this approach is that it does not take into account requests of the user, or attributes of the service, it is only based on requests of similar users. Also as this approach is based on the similar users, recommendation is in fact performed for a group of users, rather than for a specific user.

Content-based approach is based on description of a desired object (in this paper, service). The description of the service is compared to the description obtained from user's request. Usually this description is in form of semantic information about the service, which contains machine-readable information about the functionality of the service [3]. This description can be stored in various

formats. In our work we focus on semantic description in OWL-S and ontology in language OWL.

Hybrid approach is a combination of both approaches mentioned above. It is the most common approach for recommendation. Its goal is to use advantages of both approaches while eliminating the disadvantages [3].

In this paper, hybrid approach for recommendation of services is used, we also focus on recommendation based on context.

Various authors tend to choose different approaches for service recommendations. Recently approach using Quality of Services (QoS) has become more popular. In 0 authors propose an approach that uses QoS information and collaborative filtering. Their approach is based on an idea of users providing their individual QoS data of used services. In the first step they find similarities between users by comparing their QoS data using collaborative filtering and predict missing QoS values. Then they recommend web services based on predicted QoS values by comparing predicted values, values obtained from the user and values from similar users 0.

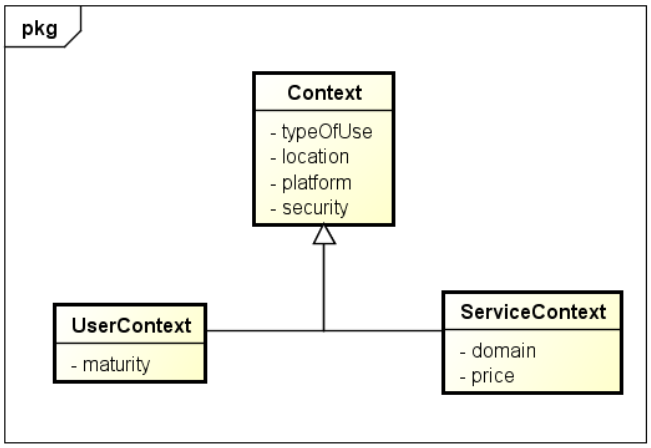
In [3] however authors propose an approach that uses context for recommendation. Their approach is based on comparing information about services and users stored inside context and semantical information about service and request of user. Authors propose a tool for description logics reasoning, and they perform content-based recommendation.

Authors in 0 selected different approach. They present a novel Web service recommendation framework. Their approach is based on QoS information, user's history and interests. They compare these data with descriptions of web services using content-based approach.

Each of these approaches that used collaborative filtering have faced Cold-start problem. It was unable for them to overcome problem with insufficient initial data. Authors who proposed an approach using context [3] did not take into consideration possibility of wrong contextual information. Also their approach did not have a solution for problem of missing contextual information. Thus in our approach we focused on these problems and tried to resolve them.

- CONTEXT

Although the term context is nowadays very popular, we have not found its standard and exact definition. Various authors define context differently, while often it is quite strongly related to content (so called context-content definitions). Reason for this is that it is not always possible to divide context from content [3].



Class diagram representing context definition

There are many various descriptions of context, perhaps the most widely one is: "Any information that can be used for characterization of entity or situation" [4]. The field of context is nowadays under intensive research. Researchers focus on its definition, modeling and usage. The context can include both functional and non-functional attributes of the service. Both attributes can be used during discovery and recommendation of services [3], [4].

In our approach we define the context of the user, and the context of the service. Class diagram describing this definition is in 0Both contexts have common base from which they inherit, and also each context has specific attributes. Using common base brings us possibility to compare these contexts and also modify them. Even if the model of context is same for each user and service, each user and each service have specific context. Context does not describe group of similar objects (users or services), but it is unique for an object it describes (user or service).

Interesting information which could be stored inside the context to characterize service or a user tends to be QoS. QoS represents non functional properties of the service 0. For web services, QoS are mainly composed of attributes that model its performance like reliability, throughput, and etc. This information can be highly dependent on network distance between service and its user [5]. These attributes can be modeled within the context of a service. However, since they are dependent on a location, it is more suitable to model location of the user and location of the service (further information below) [5]. Our interest was to find the answer to the question if QoS attributes could be used for modeling context of the user. In the context of a service they represent values describing the service, but they can't describe the user. However, it is possible to model the requirements of the user. As it was stated in [6], there are many attributes which can be included in QoS. These attributes describe services, however they can be used for describing the demands of users. Since context can include a information that characterizes the given entity, it is possible to include attributes from QoS into context.

In common context we wanted to model attributes which can describe attributes of the service and also requirements of the user. Therefore we selected those attributes which could represent both the service and the user. These attributes are:

- *typeOfUse* – represents the type of activity for which service was created (in case of service

context), or type of functionality that the user searches for (in case of user context)

- *location* – represents the location of user, or location (or region) for which service was created. There can be services that are created specifically for some region and thus it is appropriate to represent location as an attribute of context 0. Also as it was mentioned in [5], it is more appropriate to model location rather than attributes like reliability, throughput, etc. In terms of user context, if the user would have to specify his requirements and desires on reliability and throughput of services, most of the users would select the best option available, therefore modeling of these attributes for the user would not bring relevant information. It is more appropriate to model location of the user as well, so locations of the user and service can be compared
- *platform* – although services are designed to be platform independent, each platform has its own specifics. Thus it is appropriate to define an attribute that would represent targeted platform of a service, and platform that user uses for execution of recommended services
- *security* – this attribute originates in QoS. Every service has a specific level of security and also each user has a demand on security of services which he desires. This attribute models this property

Each context inherits these common attributes, but each context also has specific attribute(s). Service context has these specific attributes:

- *domain* – represents functional domain for which service was created (for example Communication, Economy, etc.). In this paper there are 9 domains defined. This attribute is defined only for context of a service, because service can be created for a specific domain, but user can only make a request for a service from specific domain. User as such does not belong into any domain
- *price* – represents value that must be paid for usage of this service. This attribute also origins in QoS and is modeled only for service, because price is specific for service. User may be willing to pay for one service more and other less, so price can't be generalized as an attribute of his context.

Context of the user has this specific attribute:

- *maturity* - represents computer skills of a user. Each user has different level of computer skills, so it is appropriate to define this attribute. Services are not affected by any maturity so this attribute is defined for context of the user only.

Since both contexts inherit from common context, it is possible to compare them, to use them in recommendations and also to utilize them for modification of other contexts.

Context can be created in two ways. Either it can be created during importing the service (service context) and user registration (user context), or it can be obtained or modified by usage of the system (this will be explained in chapter ¯). However, the provider of the service can state, that the context of his service can't be further modified – in this case he marks the context of the service as immutable. This is related to the fact that the service provider knows best if the particular context can be modified or it has to stay untouched.

- PROPOSED SOLUTION

In this paper, hybrid approach for recommendation of services is proposed. We propose an approach that is based on collaborative filtering, content-based recommendation and also context-based recommendation. Our solution is composed of two steps – recommendation of services and modification of context.

By recommendation of services we want to achieve that the selection of the service suites best to the needs of a user. User has to specify his requirements for the service. He submits his request in the form of OWL ontology, which contains two classes – one as a description of service inputs and other as a description of service outputs. This assures that he can provide request for functionality in the terms of semantics. We propose a way of recommendation using context in which the context of a service is compared to the context of a requesting user. Since the service context contains two more attributes than the context of a user, the user specifies desired values of these two attributes within his request.

In the first step of recommendation an initial service selection is performed. This initial selection is created because of computation time improvement. It would be very time consuming to execute recommendation upon every service that system has available, it is faster to recommend upon subset of these services. Using SPARQL-DL, query for retrieving all subclasses of the provided request ontology classes is created and performed. In this step ontologies which describe available services are used. Services which are described using any of these ontology classes are selected. These services represent initial selection. Every service inside initial selection at least partially fulfills functional request of a user. Upon this selection, recommendation itself is performed. For each service in this initial selection, its likelihood is computed. It represents assumed probability that the user will be satisfied with given service (in percentage). This value is computed for every service separately. The proposed formula is displayed in Figure 2.

$$likelihood_{total} = \frac{\begin{array}{l}(weight_{\text{semantic}} * likelihood_{\text{semantic}}) + \\ (weight_{\text{context}} * likelihood_{\text{contextual}}) + \\ (weight_{\text{collaborative}} * likelihood_{\text{collaborative}})\end{array}}{3}$$

Class diagram representing context definition

Total likelihood of service is computed as a weighted average of partial likelihoods. There are three partial likelihoods – semantic, contextual and collaborative. Each of them represents expected value of likelihood returned by appropriate recommendation approach.

Semantic partial likelihood is based on content-based approach. It represents similarity between semantic description of the service and semantic request of the user. Since OWL supports multiple inheritances, user can specify as many classes of ontology for specification of his request as he wants to. Value of semantic likelihood represents a portion of required ontology classes and ontology classes provided by the service (for input and output separately). During comparison of ontology classes, object-oriented approach of polymorphism is used – if object A extends object B, then object A can be referred to as object B. Thus during computation of semantic likelihood, service does not need to be described using exactly the same ontology classes as specified in request. It is suitable if it is described using ontology classes which extend ontology classes used inside request.

Collaborative partial likelihood is based on collaborative approach. It represents similarity between context of an evaluated service and average context of contexts of services which were positively recommended to users similar to current user. User can be marked as similar to current user, if his context differs from context of current user in maximum one attribute. Services which were previously successfully recommended to similar users are retrieved and average context of their contexts is created. Values of attributes of this average context represent those values, which are the most common inside contexts of individual services contexts. Resulting contextual likelihood represents percentual match of service context values and average context of services.

Contextual partial likelihood represents percentual match of values of context of current user and evaluated service.

As it was mentioned above, these partial likelihoods are weighted. Every user has different requirements on resulting service or different likelihood is more important to him. Using weights we enable him to specify importance of each approach and thus modify resulting likelihood of service.

After total likelihood is calculated for each service, services are ordered in descending order by this likelihood. Then a sub-list of services is created. Maximum number of services in this sub-list is specified inside settings of the recommendation system. Sub-list is created from top of original ordered list. After sub-list is created, this trimmed ordered list of services is recommended to the user.

We expect that the user tries out a few services of the recommended ones and then marks services which suit him and submits the result. In the next step clarification of context takes place. Situation may occur, when context of a service, or context of a user is not correctly specified, or does not have specified values of some attributes, or it hasn't any values at all. Users and also service providers may be lazy, or just skip the step of context creation. Therefore we propose an approach for context clarification. Using context clarification we want to achieve completion of missing context attributes, or correction of incorrect ones. Context clarification is based on previous recommendations. Clarification of the user context is based on contexts of services which suited him in the past. Clarification of the service context is based on contexts of users who were satisfied when using it.

After user submits his experience with recommended services, process of context clarification begins. At first user's context is clarified. Context of every service with which user was satisfied in the past is retrieved and average context is created (similarly as in the step of collaborative partial likelihood calculation). If values of user's context differ from values of average context and there was enough interaction of user in the past, different values of user context are modified to values from average context.

Clarification of service context is performed for each service that user marked as satisfying during last recommendation. Process of clarification of service context is similar to clarification of users context, only average context is created from contexts of users who were satisfied with given service in the past.

- IMPLEMENTATION DETAILS

System was implemented using standard and modern technologies for semantics and enterprise applications. Application layer was implemented using Java EJB. Framework Apache Jena and also SAWSDL4J and OWL API libraries were used for the work with semantic services. SPARQL-DL was used for reasoning upon OWL ontologies, and SPARQL from Apache Jena was used for reasoning upon semantic descriptions of services.

Services are stored inside local filesystem, or obtained from Internet. Information about services, users, contexts or recommendation history is stored

inside MySQL database, which is accessible from application layer using JPA.

An OWLS-TC4_SWRL service set was used for testing and evaluation of services. This set contains 1083 unique services described using 48 ontologies. Services inside this set belong to 9 distinct domains.

- EXPERIMENTS AND EVALUATION

Two questions were important for us when performing evaluation:

- Does hybrid recommendation using context provide better results than recommendation without contextual information? (evaluation of approach)
- Does context clarification have sense in terms of improvement of future recommendation? (overall evaluation)

Metrics used for evaluation were obtained from semantic matchmaker evaluator SME v2.2. Metrics used were precision, recall and F1.

Four different classes for possible recommended services were defined:

- True positive (TP) – service should be recommended and was recommended
- False positive (FP) - service shouldn't be recommended and was recommended
- True negative (TN) – service shouldn't be recommended and wasn't recommended
- False negative (FN) – service should be recommended and wasn't recommended

Equations used for calculations of metrics were:

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \cdot recall \cdot precision}{recall \cdot precision}$$

Equation used for calculation of metrics

- *Evaluation of approach*

This evaluation was performed by comparing results of recommendation with results from other recommendation systems. For evaluation of approach four experiments were conducted. OWLS-TC4_SWRL services set contains predefined queries. These queries have defined services upon which recommendation should be performed and also expected results are provided. Four different queries were selected. The reason for their selection was the fact that they had different requirements for required service and they were coming from different domains. The number of services for each query was 19, 112, 194 and 198.

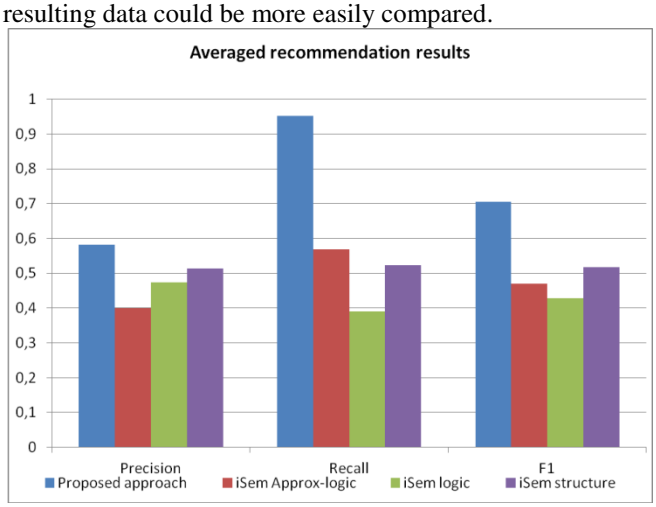
For performing the comparison a semantic matchmaker evaluator SME v2.2 tool was used. This tool is created for evaluation of semantic matchmakers using standard measures, and can be used for work with OWLS-TC4_SWRL services set. Matchmakers (recommendation systems) are provided as plugins into SME. Three different matchmaking methods were selected for comparison. These methods are from matchmaker iSeM v1.1:

- iSem approx. logic-based
- iSem logic-based
- iSem structure

Every query was submitted to every method.

Then the recommendation was performed by our system. At first context was created for services upon which recommendation had to take place. To simulate real situation, we assumed that services which had to be recommended should have values of the context more similar to the values of user's context, unlike the services which had not be recommended. After the context was created, recommendation was performed. The results were evaluated using the same metrics as in the SME. After each recommendation, database was erased and re-created for the next recommendation. This is due to the fact that we did not want to have previous data about recommendations in database.

After recommendation was performed by each method and for each query, results were evaluated. Evaluated values from queries were averaged, so resulting data could be more easily compared.



Results of evaluation of approach

As can be seen in 0in all aspects we have obtained better results than the other approaches. The main reason is that with using contextual information it was possible to overcome Cold-start problem. Other recommendation methods had difficulties with providing good results, because there was no information about previous recommendations, but context information is available from the beginning so the recommendation could be more effectively performed.

- *Overall evaluation*

To evaluate meaning of context clarification and possible overall improvement in recommendations, multiple recommendations had to be performed.

For evaluation two queries were selected from evaluation of approach (query A and query B). Contexts for these services were created like in evaluation of approach (more expected result, more common context values). Because of simplicity, contexts of services were marked as immutable and only context of the user could be altered. Three users were generated (Users A, B and C). The users B and C had set all the values of their context, but the user A had set all values except *typeOfUse* attribute. This was the value of the context that had to be clarified.

To simulate real-case scenario where user's demands for the recommended service increase, a simple algorithm was implemented. For the first recommendation, the user was satisfied if their likelihood was above 60%. For every next recommendation his demand for likelihood was increased by 12% (so during the second recommendation limit was 72% and so on). Only services, whose likelihood was above user's limit, were marked as satisfying.

Next, recommendations were performed by steps inside 0

Recommendations of user A were recorded. He was requesting the same query, so improvement in results could be evaluated. Also since he was missing one attribute, results could analyze whether results for his requests improved after clarification of this attribute. Since other similar users were also submitting queries (also different queries, so different services could be recommended to them), contextual likelihood did not have zero value anymore. Because of these two factors, overall evaluation of approach and context clarification could be performed.

Iteration	User	Query
1	A	A
2	B	A
3	C	A
4	A	A
5	B	B
6	C	B
7	A	A

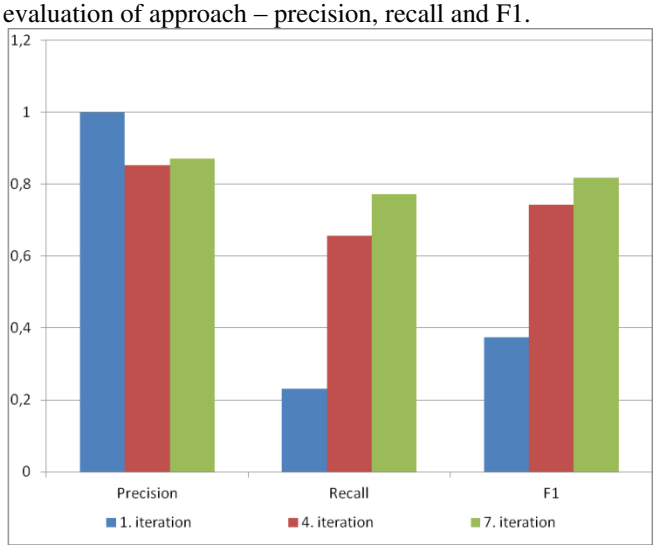
Steps in evaluation of clarification

Likelihood / Grade	3	2	1	0
<1;0,8)	TP	FP	FP	FP
<0,8 ; 0,5)	FN	TP		
<0,5 ; 0,3)		FN	TP	
<0,3 ; 0)			FN	TN
0				

Table for association of recommended results to metrics classes

As was mentioned above, every query had set of services which were expected as the result of recommendation. Each expected result was qualified with attribute *grade*, which informed about suitability of the service. Values of this attribute were 0, 1, 2 and 3. Higher the value, more suitable was the service. Depending on the grade of expected service and likelihood of recommended service it was associated with appropriate class (TP, TN, FP or FN) by pattern in 0

Results of recommendations for the user A were then evaluated. Metrics were the same as in the evaluation of approach – precision, recall and F1.



Results of overall evaluation

As can be seen in Figure 7, the results have improved during the recommendation. After the first iteration, there was no information about previous recommendations, so collaborative likelihood had value zero and also context of the user had missing attribute. After the fourth iteration it can be seen that the results have significantly improved (except of precision). Since this was the fourth recommendation, there were previous interactions and collaborative likelihood had no longer zero value. After the seventh iteration missing context value was set so system could recommend in its full potential. As can be seen from results - although demands of the user were higher after each recommendation, the results have still improved. It was proven that with clarification of context and using hybrid approach for recommendation better results can be obtained.

- COMPARISON WITH OTHER APPROACHES

In 0 authors propose recommendation system that is also based on context. Their recommendation is based

on content based approach. Authors in their work do not distinguish between context of the user and context of a service. In our work on the other hand we propose a system that uses hybrid approach for recommendation – we also use collaborative filtering. We believe that users and services have their specifics, so it is justified to distinguish their context into two separate contexts. Also in our approach we use QoS information within the context. Another difference is that authors in 0 treat context as final. We assume that attributes of contexts might not be filled, or they might be filled with wrong parameters so we propose an approach for clarification of contextual information.

• CONCLUSION AND FUTURE WORK

We have designed, implemented and evaluated service recommendation system based on hybrid approach and contextual information. We defined separate context for each user and service which also contains QoS information. Since we assume that values of context might not be filled or they might be filled with wrong values, we propose an approach for clarification of these values.

By evaluation of the system it was demonstrated that recommendation using context can overcome cold-start problem. Compared to other recommendation systems, our system provided better results, while the used metrics was not designed by us.

Also it was proven that clarification of the context has its value. In overall system evaluation experiment the simulated users had increasing demands, nevertheless the results were continuously improving.

In computation of semantic partial likelihood, semantic reasoner does not take into account every information that could be stored inside ontology. Therefore in the future work the process of comparison of semantic description of the service and semantic request from the user could be improved.

Also implemented approach for comparison of *location* attribute is fairly simple – locations are equal if contents of these attributes are equal. In 0 authors propose an approach for comparison of locations which describe entries in QoS manner. This approach is based on dividing entries into regions by their location and further comparing them. Using their approach it would be more accurate to compare *location* attribute of contexts and thus every attribute of QoS which depends on location.

• REFERENCES

T. Erl, "Service-Oriented Architecture: Concepts, Technology and Design", vol. 1. Prentice Hall Professional Technical Reference, (2005). 760s. ISBN 0-13-185858-0.

T. Erl, C. Utschig, B. Maier, H. Normann, B. Trops, "Next Generation SOA: A Real-World Guide to Modern Service-Oriented Computing", vol. 1. Prentice Hall Computer, (2014). 400s. ISBN 0133859045.

L. Lie, F. Lecue, N. Mehandjiev, "Semantic content-based recommendation of software services using context". In: ACM Transactions on the Web, (2013), vol. 12, issue 3.

Y. Na, L. Jian, Z. Yinliang, "A Context-based Framework for Web Services Discovery". In: Advanced Computer Control, Harbin, (2011), vol. 3, pp 238-341

M. Tang, Y. Jiang, J. Liu, X. Liu, "Location-Aware Collaborative Filtering for QoS-Based Service Recommendation". In: Web Services (ICWS), Honolulu, (2012), pp 202-209

A. Al-Moayed, B. Hollunder, "Quality of Service Attributes in Web services". In: Software Engineering Advances (ICSEA), Nice, (2010), pp 367 - 372

Z. Zibin, M. Hao, M.R. Lyu, I. King, "QoS-Aware Web Service Recommendation by Collaborative Filtering", In: Services Computing, IEEE Transactions, (2011), vol. 4, issue 2, pp 140 - 152

K. Guosheng, L. Jianxun, T. Mingdong, L. Xiaoqing, "AWSR: Active Web Service Recommendation Based on Usage History", In: Web Services (ICWS), Honolulu, (2012), pp 186 - 193