Универзитет „Св. Кирил и Методиј“

ФАКУЛТЕТ ЗА ЕЛЕКТРОТЕХНИКА И ИНФОРМАЦИСКИ ТЕХНОЛОГИИ

Институт за компјутерска техника и информатика

Милош Јовановиќ

АВТОМАТСКА КОМПОЗИЦИЈА НА СЕМАНТИЧКИ ВЕБ СЕРВИСИ

-магистерски труд-

Скопје, 2010

Ментор: **Доц. д-р Димитар Трајанов**
Факултет за електротехника и информациски технологии

Ментор: Доц. д-р Димитар Трајанов
Факултет за електротехника и информациски технологии

Комисија: Акад. Проф. д-р Љупчо Коцарев
Факултет за електротехника и информациски технологии

Доц. д-р Димитар Трајанов
Факултет за електротехника и информациски технологии

Доц. д-р Соња Филипоска
Факултет за електротехника и информациски технологии

Датум на одбрана:

Датум на промоција:

Магистерскиот труд е од областа на техничките науки

Содржина

Листа на слики

Листа на табели

Благодарност

Би сакал да му се заблагодарам на мојот ментор доц. д-р. Димитар Трајанов, како и на членовите на комисијата академик проф. д-р. Љупчо Коцарев и доц. д-р. Соња Филипоска, кои со своето искуство, знаење и совети успешно ме водеа и придонесоа за изработка на овој магистерски труд.

Изразувам голема благодарност и до моите колеги м-р Игор Мишковски, м-р Сашо Граматиков и Ристе Стојанов кои ми помагаа и ме охрабруваа во текот на изработката на магистерскиот труд.

Секако, сакам особено да му се заблагодарам на моето семејство и на моите најблиски, кои ми ја дадоа сета потребна поддршка и разбирање.

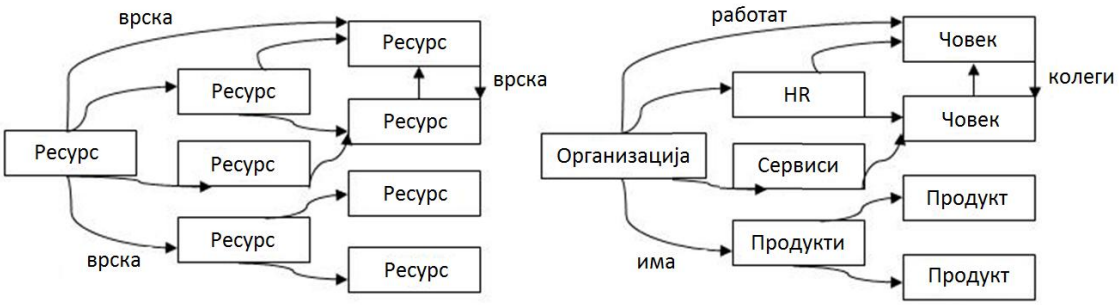
1. Вовед

Постоечкиот Веб претставува огромна мрежа на документи. Овие документи, а со тоа и податоците кои тие ги содржат, меѓусебно се поврзани на *механички* начин, со помош на меѓусебно референцирање со хиперлинкови. Она што во последните девет години е актуелно, е идејата на World Wide Web Consortium (W3C) и големиот број негови истражувачки и индустриски партнери: проширување на принципите на сегашниот Веб, од поврзување на документи, во поврзување на податоци. Ваквиот потфат е наречен *Семантички Веб* и има за цел податоците кои се наоѓаат на Вебот меѓусебно да ги поврзе според нивното значење, односно *семантика*. Идејата за Семантичкиот Веб првпат се појавува во статијата „The Semantic Web“, објавена во мај 2001 година во Scientific American, од страна на Тим Бернерс-Ли, Џејмс Хендлер и Ора Ласила [1]. Според нив, Семантичкиот Веб ќе ги структурира содржините на Вебот кои имаат значење, притоа создавајќи околина во која софтверските агенти, кои одат од веб страна на веб страна, ќе можат да извршуваат софистицирани задачи зададени од корисниците [1].

Семантичкиот Веб не е посебна, одвоена мрежа, туку проширување на постоечкиот Веб, во кој информацијата ќе има добро дефинирано значење, овозможувајќи им на компјутерите и луѓето полесно да соработуваат. Првите чекори за остварување на оваа идеја веќе се преземени. Во блиска иднина, со постојаниот развој на технологиите на Семантичкиот Веб, како што машините ќе стануваат поспособни да ги процесираат и „разберат“ податоците кои денеска

само ги прикажуваат, ќе се добијат нови значајни функционалности за крајниот корисник [1].

Семантичкиот Веб, главно, се концентрира на две работи. Прво, како да се креираат заеднички формати за интегрирање и комбинирање на податоците извлечени од различни извори, што е пристап различен од пристапот на постоечкиот Веб, кој се концентрира на размената на документи (HTML страни, на пример). Второ, креирањето на јазик во кој треба да се запишуваат релациите на податоците со објекти од реалниот свет. Тоа би му овозможило на корисникот (или машината) да започне да работи со податоци од една колекција на податоци, а потоа да продолжи да работи со податоци кои се наоѓаат во огромно множество на колекции на податоци, кои не се физички поврзани со почетната, туку се *однесуваат на истата работа* за која е заинтересиран корисникот. Тоа множество на колекции на податоци се наоѓа на Вебот, односно на Семантичкиот Веб [1]. Притоа, корисникот не мора експлицитно да знае со кое множество на податоци работи. На тој начин, Семантичкиот Веб им дава чувство на корисниците дека сите колекции на податоци кои се наоѓаат на Вебот, им стојат на располагање како една голема колекција на податоци [2][3].



Слика 1-1: Еволуција на Вебот кон Семантички Веб

Идејата е Семантичкиот Веб да се креира постепено, со инкрементални промени, преку собирање машински читливи описи на податоците и документите кои веќе се наоѓаат на Вебот. Како што знаеме, Вебот е огромно множество од статични веб страниците кои се поврзани заедно преку хиперлинкови. Во моментот Вебот е во фаза на еволуција кон Семантички Веб, како што е илустрирано со примерот на слика 1-1. Во рамките на самиот потфат на Семантичкиот Веб, постојат повеќе различни пристапи како да се додава семантичкиот опис на Веб ресурсите. На левата страна од слика 1-1, е прикажан граф кој го претставува денешниот, синтаксички базиран Веб, составен од механички поврзани ресурси. Ресурсите се поврзани заедно и формираат мрежа. За еден софтверски агент, не постои видлива разлика помеѓу ресурсите, односно помеѓу врските кои ги поврзуваат нив. За да им се даде значење на ресурсите и

врските, се испитуваат и развиваат нови стандарди и јазици. Правилата и описните информации кои се достапни од овие јазици дозволуваат индивидуално карактеризирање на типот на ресурси на веб и врските помеѓу ресурсите, како што е илустрирано на десната страна од слика 1-1.

Помеѓу ресурсите достапни на Вебот се наоѓаат и оние ресурси кои нудат одредени услуги, односно *сервиси* [3]. Под „сервиси“ се подразбираат не само оние веб локации кои нудат нестатички информации, туку и оние кои овозможуваат извршување одредени акции или правење одредени промени во „светот“, како што продажба на одреден продукт, или пак, контрола на одреден физички уред. Семантичкиот Веб би требало да им овозможи на корисниците автоматски да ги лоцираат, селектираат, искористуваат и контролираат сервисите кои се наоѓаат на Вебот.

За да може да искористи одреден Веб сервис, на софтверскиот агент му е потребен соодветен опис за сервисот, кој може да се интерпретира од страна на софтверски агент, опис во кој е назначен и начинот на кој може да се пристапи до сервисот. За таа цел, потребна е рамка според која ќе се креираат, разменуваат и искористуваат овие описи во рамките на Семантичкиот Веб. Постојат повеќе различни рамки и јазици кои се користат за оваа намена. Преку користењето на јазиците за анотација на сервисите кои постојат во рамките на Вебот, се добиваат т.н. *семантички веб сервиси*: веб сервиси чиј што опис и параметри се ставени во некаков контекст, односно се семантички анотирани.

Главниот проблем кај постоечките веб сервиси е тоа што нивниот опис е базиран на чист текст. Според тоа, сите пребарувања, откривања, композиции и извршувања се базирани на совпаѓања на синтакса. Како последица од тоа, искористувањето и интеграцијата на веб сервисите мора да се врши мануелно. Она што е главната идеја на семантичките веб сервиси, е семантичка анотација на веб сервисите, односно додавање семантика во нивниот опис, која ќе помогне да се автоматизира голем дел од функционалноста на самите веб сервиси. Семантичката анотација овозможува автоматско откривање на бараните сервиси во дадениот контекст, наместо откривање преку совпаѓање на текст.

Семантичките веб сервиси нудат една уникатна можност: автоматизација на процесот на композиција на самите веб сервиси, со цел да се обезбеди нова функционалност за крајниот корисник, функционалност која претходно не била достапна. Оваа автоматска композиција му овозможува на крајниот корисник, или неговата апликација, да поставува барања до системот за кои нема конкретен веб сервис, туку постои можност за автоматско препознавање на множество од сервиси, со чие правилно комбинирање би се извршило барањето на корисникот. Пронаоѓањето, селектирањето, композицијата и повикувањето на семантичките веб сервиси може да се врши автоматски, од страна на некој интелигентен систем.

Дополнително, експресивноста на онтологиите кои се користат во рамките на Семантичкиот Веб може да се збогати со додавање на дополнителна функционалност, односно со додавање на сервиси во рамките на самите онтологии и податоците кои се базираат на нив. Генерално, базите на знаење во рамките на апликациите изградени со технологиите на Семантичкиот Веб содржат податоци кои во текот на времето не се менуваат многу често. За ваквите податоци, можеме да сметаме дека се статички. До сега развиените технологии на Семантичкиот Веб сосема добро одговараат на потребите за развој на апликации кои целосно се темелат на нив [4]. Но, моменталниот развој на технологиите на Семантичкиот Веб ги ограничува апликациите во еден одреден поглед: работата со често променливи податоци и податоци кои на одредена апликација ѝ се потребни во реално време. Ваквите нестатички информации не е практично и исплатливо да се чуваат во базите на податоци.

Целта на овој магистерски труд е креирање на систем за интеграција и автоматска композиција на семантички веб сервиси во онтологиите. Преку проширување на онтологиите со функции, односно сервиси, се овозможува онтологиите да покријат поголем аспект од процесот на развој на апликации, со што нивната улога од технологија погодна за работа со податоци и знаење се проширува со функционалност. Внесувањето функционалност во семантичките апликации директно преку податочниот модел, придонесува да се намали потребата од дополнително програмирање во рамките на семантичките апликации. Со тоа се отвора можноста да се користат онтологиите и како податочен модел, но и како функционален дел од семантичките апликации. На тој начин, само со дефинирање на соодветна проширена онтологија, корисникот може да креира нова апликација, која се потпира единствено на технологиите на Семантичкиот Веб.

Магистерскиот труд е организиран во седум поглавја. Во ова поглавје е даден краток вовед во Семантичкиот Веб, како и предностите, предизвиците и пречките со кои се среќаваме при развојот на семантички апликации. Во продолжение, во второто поглавје се презентирани технологиите на Семантичкиот Веб, при што е направен преглед на технологиите кои се препорака на W3C конзорциумот: RDF, RDFS, онтологиите, OWL, SPARQL, RIF, итн., како и на останатите предложени нивоа кои ја сочинуваат архитектурата на Семантичкиот Веб. На крајот од второто поглавје е направена анализа на софтверските рамки за развој на семантички апликации, со особено внимание на Jena рамката, како најмоќна рамка за развој на семантички апликации. Во третото поглавје е презентиран концептот на семантички веб сервиси. Најнапред се претставени класичните веб сервиси, потребата од нив, нивното користење, технологиите кои постојат со цел да се овозможи нивното користење, постоењето на централен регистар на ниво на целиот Веб за објавување и лоцирање на веб сервисите, итн. Потоа е претставена потребата од додавање семантика во класичните веб сервиси,

за да и тие како елементи од постоечкиот Веб мигрираат и се приклучат кон идејата за семантичка анотација на ресурсите на Вебот, со цел добивање на Семантички Веб. Направен е преглед и на типовите на семантичка анотација, како и на апликациите кои се користат за семантичка анотација. Претставени се деталите за семантичката анотација на веб сервисите, со сите сличности и разлики од семантичката анотација на останатите ресурси од Вебот. На крајот од третото поглавје е направен преглед и анализа на постоечките рамки за семантичка анотација на класичните веб сервиси.

Во четвртото поглавје е презентирана рамката за проширување на онтологии со функции. Се разгледува потребата од проширување на онтологиите, со цел да се обезбеди механизам за работа со често променливи податоци во рамките на семантичките апликации, како и да се зголеми улогата и употребливоста на онтологиите во семантичките апликации. Претставена е онтологијата која го овозможува проширувањето и која е изработена како дел од овој магистерски труд. Потоа е направен преглед на карактеристиките на потребниот систем за работа со семантички веб сервиси, кој има за цел да го подржи проширувањето на онтологиите. Кон крајот од четвртото поглавје претставен е концептот на автоматска композиција на семантички веб сервиси. Во петтото поглавје е презентирана архитектурата и имплементацијата на целиот систем. Во рамките на ова поглавје, опишан е алгоритмот за наоѓање на соодветен семантички веб сервис, односно соодветна композиција од семантички веб сервиси. Даден е детален преглед на чекорите од алгоритмот: наоѓање на кандидат семантички веб сервиси, селекција на најпогодниот семантички веб сервис или композиција на семантички веб сервиси, повик на селектираниот сервис или композиција и добивање на резултатот. Потоа е презентирана веб апликација која се користи за семантичка анотација на класични веб сервиси, со цел да се добие потребниот репозиториум на семантички веб сервиси со кои работи имплементираниот систем. На крајот од поглавјето, даден е практичен пример за користење на проширена онтологија. Во шестото поглавје се донесени генерални заклучоци од предложеното проширување и неговата имплементација и врз основа на овие заклучоци, се предложени насоки за понатамошна работа. На крајот од овој магистерски труд, во седмото поглавје е даден преглед на трудови и книги користени при истражувањето и имплементирањето.

2. Технологии на Семантичкиот Веб

World Wide Web Consortium (W3C) развива интероперабилни технологии (спецификации, препораки, софтвер и алатки) кои имаат за цел да го искористат целосниот потенцијал на Вебот [5]. Конзорциумот е телото кое, помеѓу другото, ги надгледува и координира истражувањата и проектите во областа на Семантичкиот Веб.

Како дел од своите активности на полето на Семантичкиот Веб, W3C засега има објавено спецификации за следниве технологии:

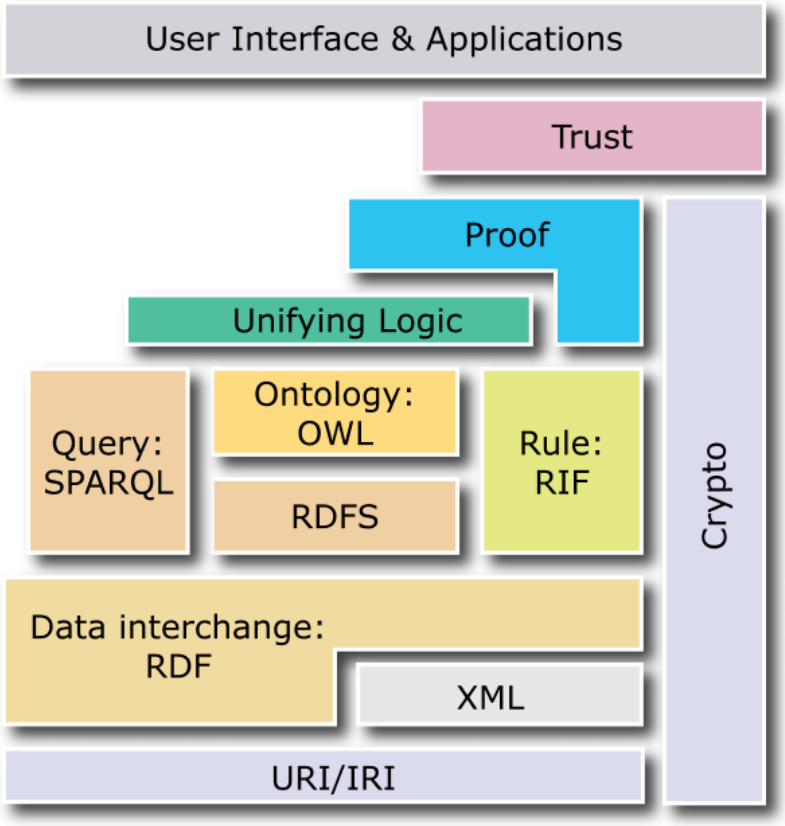
- Resource Description Framework (RDF),
- Gleaning Resource Descriptions from Dialects of Languages (GRDDL),
- RDFa in XHTML,
- SPARQL Query Language for RDF, и
- Web Ontology Language (OWL).

Овие технологии се наоѓаат во долниот, односно основниот дел од слоевитата архитектура на Семантичкиот Веб (слика 2-1) [2]. Развојот на останатите компоненти, стандарди и алатки е сé уште во тек.

Од слоевитиот приказ на Семантичкиот Веб, од страна на W3C до денес се стандардизирани и дефинирани следниве компоненти:

- URI – стандард за идентификување и лоцирање на ресурси на Веб,
- XML - ја обезбедува основната синтакса за структурата на содржините во документите [6],

- RDF - едноставен јазик за изразување на податочни модели, односно објекти и релациите помеѓу нив [7],
- RDF Schema – вокабулар за опис на својствата и класите на RDF-базираните ресурси [8],
- OWL – вокабулар за опис на својствата и класите, релациите помеѓу класите, нивната кардиналност, еднаквост, итн. [9], и
- SPARQL – јазик за поставување прашања во извори кои се дел од Семантичкиот Веб [10].



Слика 2-1: Слоевит приказ на архитектурата на Семантичкиот Веб

Resource Description Framework (RDF) претставува рамка која се користи за претставување на информациите на Вебот [11]. RDF е стандарден модел за размена на податоци на Вебот, кој овозможува обединување на податоци дури и кога шемите во кои тие се опишани се различни. На тој начин, RDF овозможува поврзување помеѓу апликациите, односно комбинирање на информациите кои се наоѓаат во повеќе апликации. Освен тоа, RDF овозможува и автоматска обработка на информациите на Вебот од страна на софтверски агенти. Со тоа, Вебот се „поместува“ од место на кое постојат информации кои може да ги прочита само човекот, кон светска мрежа на кооперативни процеси [11].

RDF Schema (RDFS) е јазик за репрезентација на знаење, кој ги обезбедува основните елементи за опис на онтологии, односно RDF вокабулари кои се користат за структурирање на RDF-опишани ресурси. Со помош на RDFS, се дефинира шемата со која ќе биде опишано одредено множество податоци, кое е опишано во RDF формат [8].

Јазикот за пишување на веб онтологии, OWL (Web Ontology Language) е јазик дизајниран за користење во апликации во кои содржините и информациите треба да се обработуваат, а не само да се презентираат на крајните корисници [9]. OWL се користи за експлицитно да се претстави значењето на термините во одредени вокабулари, како и релациите помеѓу нив. Ваквата репрезентација на термини и нивните релации се нарекува онтологија.

SPARQL (SPARQL Protocol and RDF Query Language) е јазик за поставување прашања врз податоци напишани во RDF формат. Овој јазик дозволува прашањата да се состојат од шеми за препознавање во RDF тројки, спојувања, разлики или опционални шеми [10].

Rule Interchange Format (RIF), како дел од слојот за Правила во слоевитата архитектура на Семантичкиот Веб, е рамка која е сé уште во фаза на развој [12]. Во блиска иднина се очекува да стане официјална препорака на W3C конзорциумот и да почне да добива реални имплементации.

Во последно време, интензивно се работи на подобрување и стандардизација на логиката и доказот во архитектурата на Семантичкиот Веб. Логиката и доказите претставуваат автоматизиран систем за заклучување, кој се наоѓа над онтологиите и служи за извлекување и креирање на ново знаење, односно нови релации помеѓу поимите и концептите од доменот.

Останатите аспекти, како што се доказот и довербата (Proof and Trust), сега за сега постојат само како теоретски аспекти [13].

2.1. Рамка за опис на ресурси (RDF)

Resource Description Framework (RDF) претставува рамка, односно платформа, за опишување на ресурси [7][11][14]. RDF е фамилија на спецификации, усвоени и препорачани од страна на W3C конзорциумот. Првично таа била дизајнирана како податочен модел за метаподатоци. Оваа платформа станува општ метод за опишување или моделирање на информациите кои се поставени како Веб ресурси.

RDF е платформа која успешно ги надминува ограничувањата на HTML (Hypertext Markup Language), кој претставува јазик со чија помош се дефинира визуелниот приказ на податоците кои се наоѓаат во некој веб документ. RDF

документите се основната градбена единка на Семантичкиот Веб. RDF за Семантичкиот Веб е она што е HTML за Вебот.

Од технички аспект, RDF е XML-базиран јазик за опишување информации кои се содржат во рамките на даден ресурс, или она што нам ни е од поголем интерес - веб ресурс. Под поимот веб ресурс се подразбира веб страница или цел веб сајт, или каков било ентитет за кој Вебот поседува информации во некаков облик. Со користење на овој едноставен модел се обезбедува мешање, изложување и споделување на структурирани и полуструктурирани податоци, помеѓу различни апликации.

RDF се користи и за да се опише кој било факт, односно ресурс, независно од доменот. Со тоа тој претставува основа за запишување, размена и повторно искористување на структурирани метаподатоци. Поради тоа што RDF е структурирана платформа, тој е разбирлив за машините. Машините можат да прават корисни операции со знаењето кое е изразено со помош на RDF.

Првиот клучен елемент во RDF е *ресурсот*. RDF е стандард за метаподатоци, т.е. тој нуди стандардизиран начин за наведување податоци за некој концепт. Тој концепт може да биде било што, а тоа во RDF се нарекува *ресурс*. Оттука, ресурсот е нешто што може да се опише со помош на RDF изрази. Може да биде веб страница, дел од веб страница (на пример, збор кој се наоѓа на страницата), цел веб сајт, или пак објект од реалниот свет, како на пример книга, човек, мачка – може да биде било што. Секој ресурс се идентификува со помош на единствен идентификатор за ресурси (Uniform Resource Identifier - URI, анг.). Овој идентификатор се користи како име на ресурсот. Причината зошто е потребно да се користи URI е изразена како:

Името на секој ресурс мора да биде глобално. Со други зборови, ако создавачот на знаењето не е сигурен дали некој друг веќе го користи истиот идентификатор, тогаш тој не би требало да го користи. [15]

Во Семантичкиот Веб е недозволиво да постојат нејасности и недореченост, па поради тоа се прават максимални напори таквите појави да се избегнат. Со оглед на тоа, потребно е URI да се сфати како општ локатор. URI се користи и сега во рамките на Вебот, во форма на URL (Uniform Resource Locator). Она што го прави URL единствен е доменот, за кој постои ограничување на уникатноста, т.е. единственоста.

Во RDF постои и нешто што се нарекува *својство.* Својството е ресурс кој има име и може да се искористи за да се опише некој специфичен аспект, карактеристика, атрибут или релација од дадениот ресурс. На пример, ако сакаме да опишеме фотоапарат (кој претставува ресурс), тогаш *тежина* би било негово својство.

Клучниот ентитет во RDF е *изразот*. Изразите во RDF се дефинираат на следниот начин:

Ресурс (Подмет) + Својство (Предикат) + Ресурс (Предмет)

Еден ваков израз уште се нарекува и RDF тројка. RDF конвенционално го прифаќа овој формат за опишување на ресурсите и истиот не може да се менува. Ваквата поврзувачка структура формира насочен, обележан граф, каде што врските ги претставуваат именуваните релации помеѓу два ресурса, кои пак се претставени со јазлите на графот. Графичкиот приказ, како модел на RDF, најмногу се користи за едноставно и разбирливо визуелно објаснување. Ваквиот граф од RDF тројки претставува основен податочен модел за Семантичкиот Веб и неговите апликации.

RDF исказите, односно RDF тројките, можат да се претстават во неколку различни формати: RDF/XML, Turtle, N-triples, итн. Генерално, форматот со кој се претставени ваквите податоци не е пресуден: она што е релевантно е дека тие се претставени во RDF тројки. Голем број RDF алатки можат да ги препознаат и евалуираат сите формати на RDF.

Поради постоењето на RDF/XML форматот за репрезентација на податоци опишани со RDF, често се појавува забуна за потребата од постоење на RDF, кога веќе постои стандард за опишување на податоци од/за различни извори и платформи: XML. Но, RDF не претставува верзија на XML. Фундаменталниот модел на RDF е независен од XML. RDF е модел кој ги опишува релациите помеѓу два (веб) ресурса, или помеѓу веб ресурс и конкретна стринг вредност. На фундаментално ниво, единствената сличност помеѓу RDF и XML е користењето на типови на податоци од XML шема, за опис на стринг вредностите во RDF.

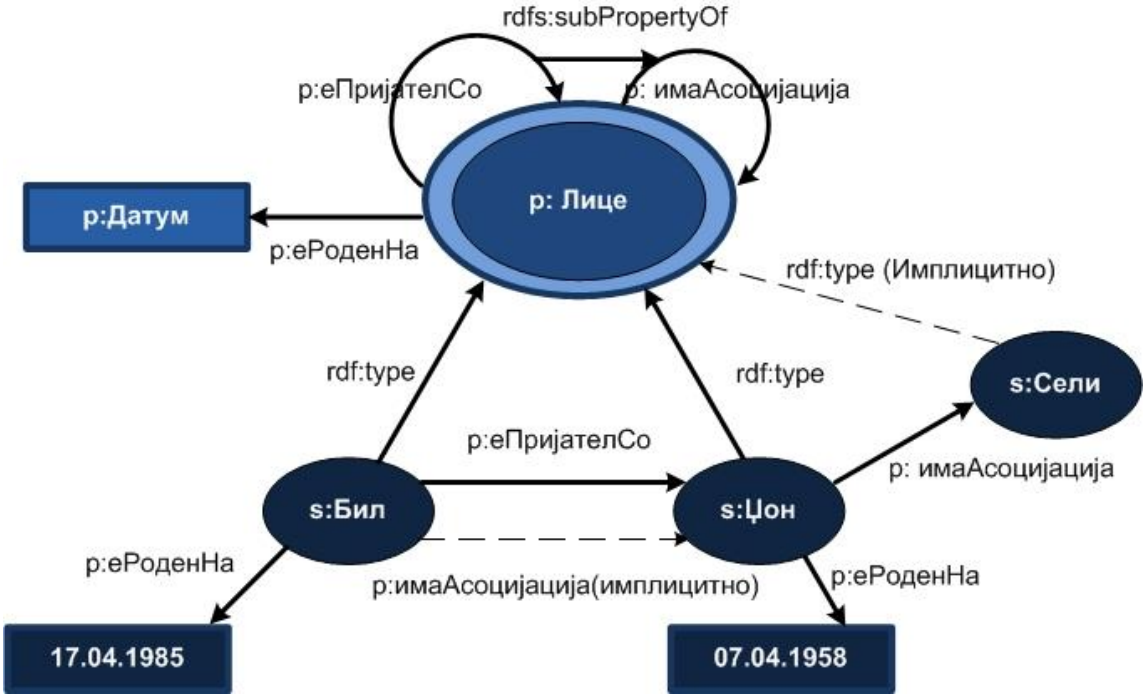
RDF платформата има и свои недостатоци. На пример, во неа не постои начин да се изразат класи, подкласи, релации кои можат да постојат помеѓу класите, симетрични својства, транзитивни својства, инверзни својства, кардиналност меѓу ресурсите, итн.

2.2. Шема за рамката за опис на ресурси (RDFS)

RDF претставува платформа за опишување на ресурси. Но таа не кажува ништо за тоа како се поврзани ресурсите меѓу себе. Со RDF им се овозможува на машините да го разберат значењето на ресурсите. Поради тоа, потребно е да се прошири RDF, така што ќе биде возможно да се поделат ресурсите во класи, подкласи и да се изразат релациите кои нив ги поврзуваат. Ова проширување на RDF се нарекува RDF шема (RDF Schema). Без RDFS, RDF документите би биле

разбирливи за машините, но само од нив никогаш нема да се постигне високо ниво на расудување. Ова е главната причина за усвојување на RDFS [8].

Со помош на RDFS, можеме да додадеме семантика помеѓу својствата и класите кои се дефинираат. Исто така, може да се специфицираат својствата и да се искажат типовите на објекти кои тие својства можат да ги примат. RDFS се пишува врз RDF, што значи дека исто така е XML базирана платформа.



Слика 2-2. Пример за репрезентација на знаење со помош на граф

Како заклучок за RDFS, би можело да се каже дека семантиката, односно значењето на еден поим, се одредува со помош на својства кои му се придружуваат и типовите на објекти кои тие својства би можеле да ги примат. Машините ја разбираат семантиката следејќи ја конвенцијата *ресурс - својство - вредност_на_својството.* На слика 2-2 е даден пример за репрезентацијата на знаење од доменот на пријатели. Во избраниот пример даден е опис на концептот *личност,* за кој се знае дека има *датум на раѓање*, како и дека *има асоцијација,* односно *има пријател.* Потоа е претставено дека Бил, личност родена на 17.04.1985 е пријател со Џон, личност родена на 07.04.1958. Од друга страна Џон е пријател со Сели, за која не е кажано експлицитно дека е од тип ентитет *Личност.* Меѓутоа системот може да донесе заклучок дека и Сели е ентитет од тип Личност, бидејќи тоа следува од дефиницијата на асоцијацијата *има асоцијација* [15]. Врските кои на сликата се обележани со испрекината линија се добиени по пат на *расудување*, процес за кој ќе стане збор подоцна.

2.3. Онтологии

Онтологиите претставуваат репрезентација на знаење која овозможува најголема флексибилност во изразувањето. Со помош на онтологиите, машините можат да „разберат, било што за било што“. За формално дефинирање на поимот *онтологија,* ќе ја искористиме дефиницијата на W3C:

„*Онтологиите се ентитети кои ги содржат поимите со чија помош може да се опише и претстави знаење од одредена област*“. [16][17]

Оваа дефиниција на онтологија се користи за да се укаже на фактот дека онтологиите се користат за да се опише и претстави некоја област на знаење. Со други зборови, онтологијата е зависна од доменот, таа не треба да се користи за да се претстави целото знаење, туку само одредена област од него, односно одреден домен. Доменот претставува една специфична област од сферата на знаењето, како на пример, фотографија, медицина, недвижен имот, образование, итн.

Онтологиите содржат *поими* и *врски* помеѓу поимите. Поимите најчесто се нарекуваат *класи,* а релациите помеѓу класите можат да се изразат преку хиерархиска структура: суперкласите претставуваат концепти од повисок ред – поопшти концепти, а подкласите претставуваат концепти од понизок ред – поконкретни концепти.

Покрај хиерархиската врска помеѓу класите, постои и уште едно ниво на релација кое се изразува со помош на *својства.* Овие својства можат да опишат различни способности и атрибути на концептите, а исто така можат да се искористат за да се асоцираат различните класи помеѓу себе. Поради тоа врските помеѓу класите не се само надкласи и подкласи, туку релациите се искажуваат со помош на т.н. својства.

Во онтологијата е кодирано знаењето од одреден домен на таков начин што тоа е лесно достапно и разбирливо за компјутерот. Ова е всушност и основната цел на онтологијата.

Предностите од користењето на онтологии можат да се изразат како:

- овозможуваат повторно искористување на знаењето од даден домен;
- претпоставките за доменот се експлицитни.

Заедно со јазиците за опишување онтологии, како што се OWL и RDFS, се овозможува кодирање на знаењето и семантиката така што машините можат да го разберат. На тој начин, тие овозможуваат автоматско процесирање на големо количество на податоци и негово интерпретирање во нови и проширени информации.

2.3.1. Предности на користењето онтологии за претставување на знаење

Од описот на концептот на онтологија како ентитет за претставување на знаење, може да се заклучи дека таа носи дополнителна комплексност при развивањето на апликации. Иако во денешно време постојат алатки за побрзо развивање на онтологии, сепак може да се потроши значително време на нивниот развој, во зависност од комплексноста на знаењето кое на апликацијата ѝ е потребно за да го користи. Поради тоа мора да постои добра причина за да се направи значајна промена во процесот на развивање на апликации во однос на претставување на знаењето. Имено, наместо традиционалниот пристап при развивање на бизнис апликација диктира користење на релациона база на податоци, се сугерира да се премине кон модел во кој ќе се репрезентира знаењето, а не податоците. На овој начин, ентитетот кој ќе се користи за перзистентност повторно останува релационата база, но во овој случај таа ќе служи како складиште за знаењето, бидејќи во неа ќе се чуваат врските меѓу концептите. Конечно, главната предност на овој пристап се состои во тоа што кога постои база на знаење, системот може имплицитно да носи нови заклучоци и да го проширува знаењето. Процесот во кој системот од некое почетно множество на знаење врши негово проширување се нарекува *расудување*. Токму ова е главната причина за мигрирање кон чување на база на знаење, наместо база на податоци. Со ова се оправдува потрошеното време за да се развијат онтологиите на знаење. Нивното користење остава можност за имплицитно проширување на знаењето.

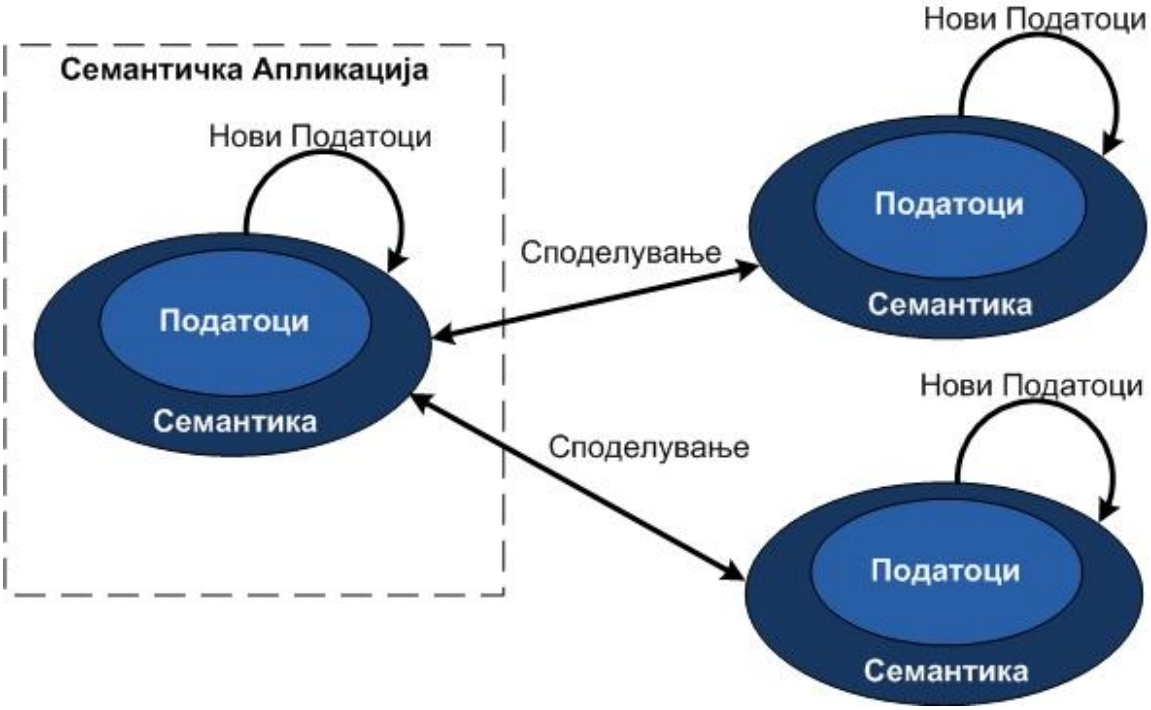
Табела 2-1: Споредба на релациона база на податоци со база на знаење

Структура	Шема	Онтологии
Податоци	Редици	Онтологии
Прашален јазик	SQL	SPARQL
Релации	Надворешен клуч	Повеќедимензионални
Уникатност	Примарен клуч	URI

Во Табела 2-1 е прикажана споредбата помеѓу релациона база на податоци и база на знаење. Како што може да се заклучи од табелата, базата на знаење нуди динамичко, проширливо складиште на податоци за разлика од релационата база. Структурата на базата на податоци зависи од нејзината шема, а структурата на базата на знаење зависи од исказите во онтологиите. Кај базите на податоци постои само еден тип на релација, а тоа е надворешниот клуч. Базата на знаење не е ограничена во тој аспект: кај неа може да постојат релации како *е-родител-на, е-*

дете-на, дел-од, е-поврзан-со, итн. Кога се поставува прашање до базата на податоци, одговорот доколку постои мора да се наоѓа во затвореното множество податоци за кое знае базата. Кога се поставува прашање до база на знаење (кај Семантичкиот Веб), знаењето може да се побара директно од различни извори и за тоа не се потребни никакви дополнителни технологии. На овој начин една апликација може во позадина да користи огромни бази на знаење распоредени на Вебот, без никаква пречка [15].

На слика 2-3 е илустриран фактот дека една апликација базирана на технологиите на Семантичкиот Веб може да излезе надвор од својот домен на знаење и да се послужи со податоци кои други податочни извори ѝ ги нудат преку некој медиум.



Слика 2-3. Споделување на податоци помеѓу апликации базирани на технологиите на Семантичкиот Веб

Врските кои покажуваат кон самата семантика означуваат дека врз основа на семантички анотираните податоци, апликацијата може да дојде до нови знаења. Тоа значи дека со тек на време ќе може да се формираат големи репозиториуми на знаење и да се создаваат навистина моќни семантички апликации врз база на тоа знаење. Со споделувањето на податоци и информации, како и со помош на расудувањето, семантичките апликации претставуваат многу помоќни алатки отколку кои било други апликации кои досега биле имплементирани.

Од таа причина се вели дека Семантичкиот Веб е *комплементарен* со постоечкиот Веб, тој нема да го надвладее, туку едноставно ќе ги пополни неговите недостатоци.

2.4. Јазик за веб онтологии: OWL

Web Ontology Langugage (OWL) е препорака од W3C конзорциумот и претставува јазик за создавање онтологии [9]. OWL е изграден врз RDFS, што значи дека исто така е XML базиран јазик. OWL дава слобода на изразување на врските помеѓу концептите кои се дефинирани на различни документи на Вебот. Тој исто така овозможува да се конструираат нови класи со помош на унии, пресеци, комплементи и други постоечки класи. Со OWL се додаваат ограничувања во бројноста, односно кардиналноста, и типот на својствата на класите, а може да се провери и дали сите членови на дадена класа имаат одредено својство, или пак само некои од нив. OWL им ја дава на машините максималната моќ за расудување, а сето тоа е овозможено благодарение на можноста да се дефинираат симетрични својства, асиметрични својства, инверзни својства, транзитивни својства, функционални својства, итн.

Голем проблем при дизајнирањето на онтологии е балансот помеѓу опциите за изразување и ефикасноста на расудувањето. Со други зборови, колку е побогат јазикот, толку расудувањето е покомплексно и понеефикасно. Поради тоа, целта е да се дизајнира јазик со кој може да се опишуваат големи онтологии, но истовремено доволно едноставен за да може со негова помош да се врши расудување. За таа цел, постојат три дефиниции на OWL: OWL Full, OWL Description Logic (OWL DL) и OWL Lite.

Првата варијанта, OWL Full, е онаа која е најбогата во поглед на можностите за изразување, но расудувањето е многу тешко да се изведе. OWL DL, пак, има одредени ограничувања: секој ресурс може да биде класа, инстанца, својство или вредност, т.е. не може во исто време да биде член на некоја друга класа. Потоа, функционалните и инверзно функционалните својства можат да бидат употребени само кај својствата од објектен тип, што не е случај кај OWL Full. Исто така, не може да се користи клучниот збор за изразување кардиналност кај транзитивните својства, а при развивањето на нови онтологии не може да се внесуваат други онтологии кои биле развиени со помош на OWL Full. Во OWL Lite се воведени уште поголеми ограничувања во контекст на можностите за изразување во однос на OWL DL. Во OWL Lite има ограничувања во изразите за еквивалентност, унија, различност, максималната кардиналност може да биде 0 или 1, а изразот за еквивалентност не може да поврзува анонимни класи, туку само идентификатори на класите.

На крај треба уште еднаш да се напомене дека предноста во користењето на поограничените варијанти на OWL носи големо подобрување на перформансите на расудувањето [16].

2.5. Софтверски рамки за развивање на семантички апликации

Постојат повеќе солидни софтверски рамки за развивање на семантички апликации кои се стабилни, со отворен код и одлична документација. Меѓу најпознатите вакви пакети се Jena [18] и Sesame [19]. И двете се имплементирани во Java програмскиот јазик и вклучуваат многубројни функции за програмско манипулирање со RDF и OWL. Sesame директно може да се постави на Tomcat сервер и на тој начин да се поврзе преку HTTP, додека пак за Jena постои посебен сервер наречен Joseki [20].

Табела 2-2: Краток опис на софтверски пакети и библиотеки кои се на располагање за развивање на семантички апликации

Пакет	Опис
4Suite	4Suite е библиотека за работа со XML и RDF имплементирана во Python програмскиот јазик [21].
Jena	Една од најпопуларните библиотеки за развивање на семантички апликации. Овозможува работа со OWL, RDF, SPARQL и расудување [18].
Sesame	Широко употребувана библиотека и сервер за работа со RDF. Има интерфејс за SPARQL и HTTP, а доаѓа и со солидни системи за расудување [19].
OWL API	OWL API е напишано во Java и содржи интерфејс за поддршка на различни системи за расудување [22].
RAP RDF API	RAP RDF API се користи за работа со RDF преку PHP [23].
Redland	Колекција од библиотеки за работа со RDF напишани во C [24].
LinqToRDF	.NET библиотека за работа со RDF преку Microsoft LINQ [25].

Во Табела 2-2 е илустриран краток опис на повеќето достапни библиотеки за работа со семантички апликации. Од сите нив, Jena е една од најкористените, најпопуларните и најдостапните.

2.5.1. Jena

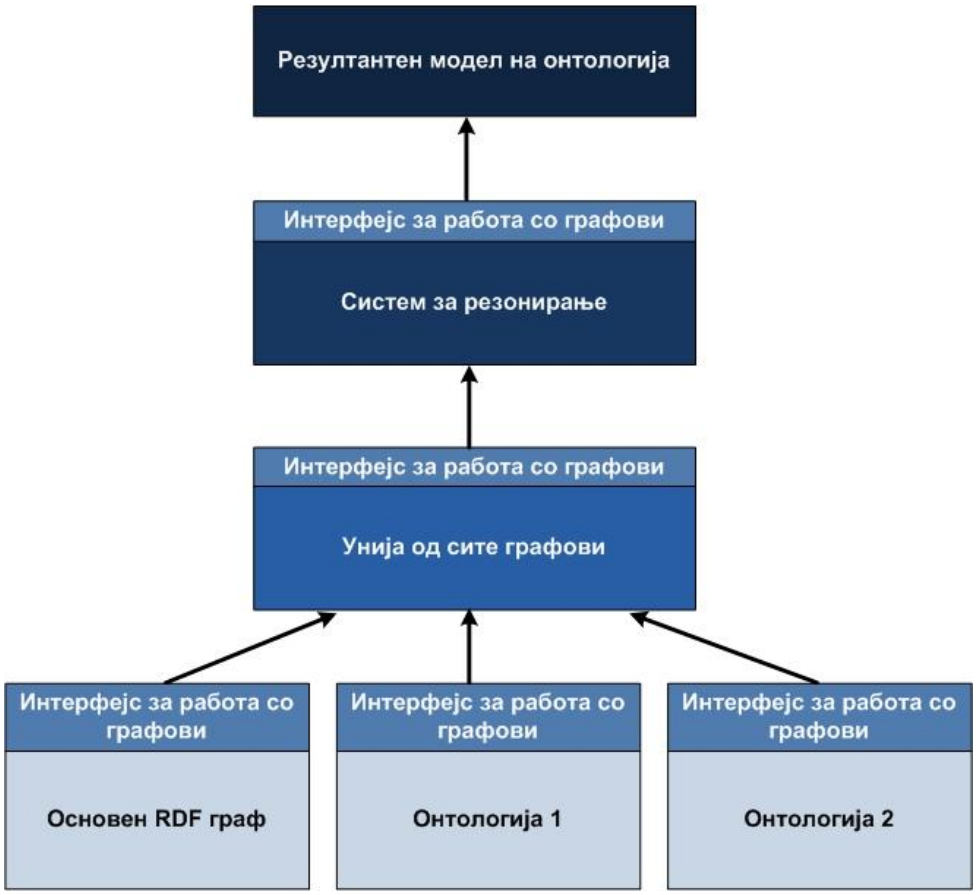
Jena е една од најмногу користените рамки кога се во прашање семантичките апликации. Развиена е во лабораториите на Hewlett Packard [15][18][26]. Во основа, Jena обезбедува рамка и околина за работа со RDF, RDFS, OWL, SPARQL и содржи системи за расудување. Jena библиотеките создаваат дополнително ниво на апстракција, кое ги пресликува изразите и концептите на Семантичкиот Веб во класи, објекти, методи и атрибути во Java програмскиот јазик. Овие објектно-ориентирани артефакти го намалуваат напорот кој е потребен за програмирање на семантички апликации. Една од најсилните страни на Jena е нејзината обемна и јасна документација. Изобилството на ресурси ги охрабрува програмерите да развиваат свои апликации со помош на Jena библиотеките.

Како дел од RDF способностите, Jena нуди управување со RDF ресурси, нивно запишување во RDF/XML, N3 и N-Triples формат. Jena исто така поддржува работа со RDFS, со помош на апликацискиот програмски интерфејсот (API) за сите проширувања кои ги нуди во однос на RDF. Уште повеќе, Jena нуди методи за работа со OWL и тоа во било која од трите варијанти: Full, DL и Lite. OWL интерфејсот за програмирање нуди можности за навигација низ графот на знаење, лоцирање на ресурси и нивно вчитување од графот [27].

Jena е способна за работа со повеќе онтологии одеднаш и тоа од различни извори. Интерфејсот за програмирање со кој доаѓа Jena многу го олеснува процесот на споделување информации. Jena исто така нуди можности за валидација на онтологија и за запишување на начинот на кој доаѓа до заклучоците, што му овозможува на развивачот да осознае на кој начин Jena доаѓа до одговорот на поставеното прашање.

Што се однесува до зачувувањето на перзистентни медиуми, Jena може да работи со OWL и RDF датотеки, но постои и можност да се поврзе и со база на податоци. Поради генеричкиот софтверски дизајн, Jena може лесно да се поврзе со различни релациони бази на податоци. Потребен е само соодветен двигател за соодветната SQL база на податоци.

Поставувањето на прашања до графот на знаење е важна тема кога се дискутира за библиотеки за развивање на семантички апликации. Кај Jena, на моделот може да се поставуваат прашања преку апликацискиот програмски интерфејс, или директно со конструирање на SPARQL прашања, со цел да се дојде до резултатите. Базата на знаење може да се постави на веб сервер, специјално дизајниран за Jena, наречен Joseki [20]. Joseki се поставува како медијатор помеѓу внесеното SPARQL прашање преку HTTP GET/POST методи, а враќа одговор во RDF/XML облик на резултатите, кои можат дополнително да се форматираат со помош на XSLT трансформации.



Слика 2-4: Организација на работата на Jena со повеќе онтологии

Најмоќната компонента на Jena е модулот за расудување. Тој нуди интерфејс за програмирање на неколку различни видови системи за расудување: RDF(S), OWL, Транзитивен и Генерички систем за расудување. Jena е компатибилна со надворешни системи за расудување, како што е Pellet [28]. Сите системи за расудување можат поединечно да се конфигурираат. На пример, OWL системот за расудување може да се конфигурира да користи DL, Full или Lite режим на работа.

На слика 2-4 е прикажано како Jena работи со повеќе онтологии врз кои се врши расудување. Најпрвин се собира целото знаење од повеќе онтологии и се врши операцијата унија врз нив. Врз таквиот граф се врши расудување и се додаваат нови врски во графот. Таквиот граф се преведува во модел на онтологија, за кој Jena има различни можности за манипулација. Важно е да се напомене дека притоа се користи ист интерфејс за пристап до графовите.

Еден од главните недостатоци на Jena е тоа што прво целиот граф на знаење се вчитува во главната меморија, а потоа се манипулира со него. Ова важи без разлика дали се работи со датотека или со релациона база на податоци. Од

овде произлегува дека има големи мемориски побарувања за големи множества на податоци, па често пати може да се случи меморијата која е доделена на Java виртуелната машина да не е доволна.

Друг недостаток е тоа што Jena нема предвидено механизми за конкурентна работа, па често пати се случува корисниците да работат со неконзистентно знаење. Постојат методи за регулирање на критични региони, но за нив треба експлицитно да се грижи програмерот.

Трет недостаток е превисоката цена на процесот на расудување во смисла на временски и мемориски ресурси. Ова ограничување добива сериозни димензии во случаи кога апликацијата треба често да биде во интеракција со корисникот т.е. базата на знаење фреквентно добива прашања од корисниците. Сепак, постојат обиди да се намали оваа цена со методи како што се затворање на графот на знаење и негова редукција.

И покрај сериозните недостатоци кои ги има оваа рамка, сепак Jena е еден од најнапредните и најмногу користените пакети за развивање на семантички апликации, па заради тоа истиот ќе биде искористен за имплементација на систем за проширување на онтологии со функционалности, кој ќе биде презентиран подоцна.

3. Семантички веб сервиси

Клиент - сервер архитектурата веќе долго време се користи како модел за градење на дистрибуирани апликации. Веб сервисите, како елементи од една таква архитектура, претставуваат апликации од серверска страна кои нудат механизам за пристап до нивните функционалности за да други апликации, т.е. клиенти, кои можат да ги повикуваат и искористуваат тие функционалности.

Сепак, и покрај големата распространетост на употребата на веб сервисите, процесот кој започнува со креирање на веб сервис, а трае до негово искористување од страна на потенцијален клиент, воопшто не е тривијален. Постојат голем број акции кои треба да се преземат, во рамките на кои клиентите треба да надминат и голем број пречки и ограничувања.

Појавата на технологиите на Семантичкиот Веб во последните години, нуди можност за олеснување и автоматизација на дел од акциите поврзани со веб сервисите. Иако првичната идеја и намена на овие технологии е додавање значење, односно семантика на ресурсите на Вебот, преку семантичка анотација, можностите за искористување на овој пристап за подобрување и поедноставување на текот на активности кај веб сервисите, се многу големи.

Дел од главните придобивки кои технологиите на Семантичкиот Веб можат да ги обезбедат во светот на веб сервисите се: автоматско откривање на веб сервисите, автоматско повикување на веб сервисите, автоматска композиција на веб сервисите, како и автоматско мониторирање на извршените веб сервиси.

3.1. Класични веб сервиси

Веб сервисите се елементи од клиент - сервер архитектурата и се користат за развој на дистрибуирани апликации. Тие, всушност, претставуваат апликации на серверска страна, кои нудат механизам за пристап до нивните функционалности за да други апликации, т.е. клиенти, кои можат да ги повикуваат и искористуваат тие функционалности.

Како комуникациски протокол кај веб сервисите се користи Hypertext Transfer Protocol (HTTP). Според тоа и множеството стандарди кои се користат за овозможување на комуникацијата со веб сервисите, претставува множество стандарди изградени над HTTP протоколот. Користејќи ги овие стандарди, апликациите можат да комуницираат една со друга, постигнувајќи интероперабилност во рамките на Вебот.

Интероперабилноста се постигнува со тоа што при комуникацијата и серверот и клиентот ги следат овие стандарди, остварувајќи меѓусебна комуникација преку HTTP, без разлика на платформата или програмскиот јазик во кој се развиени.

3.1.1. Опис на веб сервиси (WSDL)

Web Service Description Language (WSDL) претставува XML базиран јазик за опишување на веб сервиси [29]. Со WSDL описот се опфатени детали кои го идентификуваат и опишуваат веб сервисот, како што се: името на сервисот, неговите функции, влезните и излезните параметри. Според тоа, WSDL описот на еден сервис се смета за *оглас* за веб сервисот кој им се нуди на клиентите. Причината поради која се користи XML како основна синтакса, е фактот што XML е текстуален формат за претставување на податоци кој е целосно независен од платформа, како и формат кој лесно може да се обработи од секој програмски јазик.

Во процесот на креирање на веб сервис, за да бидеме сигурни дека имаме доволно информации за да го повикаме сервисот, треба да се вклучат следните ставки:

- локацијата на сервисот (обично сервисот е лоциран на некое URL);
- име на сервисот;
- типови на влезни и излезни параметри;
- протоколот кој се користи за повикување на веб сервисот и комуникација меѓу серверот и клиентот.

Пример за WSDL датотека, која претставува опис на веб сервисот getMegaPixel, е даден во продолжение. Веб сервисот getMegaPixel како влезен

параметар прима модел на камера, а како резултат ја враќа вредноста на мегапикселите со кои располага моделот на камера.

WSDL опис на getMegaPixel веб сервисот:

```
<?xml version="1.0" encoding="utf-8"?>
      <wsdl:definitions
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
      xmlns:tns="http://tempuri.org/"
      xmlns:s="http://www.w3.org/2001/XMLSchema"
      xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
      xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
      targetNamespace="http://tempuri.org/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

<wsdl:types>
    <s:schema elementFormDefault="qualified"
    targetNamespace="http://tempuri.org/">
      <s:element name="getMegaPixel">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1"
name="cameraModel" type="s:string" />
          </s:sequence>
        </s:complexType>
      </s:element>

      <s:element name="getMegaPixelResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1"
name="getMegaPixelResult" type="s:double" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </wsdl:types>

  <wsdl:message name="getMegaPixelSoapIn">
    <wsdl:part name="cameraModel" element="xsd:string" />
  </wsdl:message>
  <wsdl:message name="getMegaPixelSoapOut">
    <wsdl:part name="megaPixelValue" element="xsd:double" />
  </wsdl:message>

  <wsdl:portType name="Service1Soap">
    <wsdl:operation name="getMegaPixel">
      <wsdl:input message="tns:getMegaPixelSoapIn" />
      <wsdl:output message="tns:getMegaPixelSoapOut" />
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="Service1Soap" type="tns:Service1Soap">
```

```
    <soap:binding
transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="getMegaPixel">
      <soap:operation
soapAction="http://tempuri.org/getMegaPixel" style="document" />
      <wsdl:input>
        <soap:body use="literal" />
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal" />
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

  <wsdl:service name="Service1">
    <wsdl:port name="Service1Soap" binding="tns:Service1Soap">
      <soap:address
location="http://localhost/GetMegaPixelWS/Service1.asmx" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Генерално, еден веб сервис може да содржи неколку групи од методи (операции), при што секоја група од методи е наречена portType. Со еден повик на клиентот може да се повика еден метод. За повикување на метод, клиентот испраќа влезна порака и како резултат добива излезна порака. Секој податочен елемент од пораката е наречен wsdl:part. Протоколот кој се користи за повикување на сервисот и форматот на влезните и излезните пораки заедно е опишан во wsdl:binding тагот. Сервисот е отворен кон надворешниот свет преку една или повеќе порти, опишани со wsdl:port. Секоја порта ја специфицира единствената локација на ресурсот (URL), односно сервисот, како и wsdl:binding тагот кој се користи со портата.

Корен на секој WSDL документ е елементот <wsdl:definitions>, односно според стандардот сé би требало да биде дефинирано во рамките на овој елемент. Овој елемент ги содржи дефинициите за именските простори (namespaces, анг.).

Првиот елемент кој се наоѓа во коренот е елементот <wsdl:types>. Тој претставува XML шема, во која се наоѓаат дефинициите за кориснички дефинираните податочни типови, како и за вградените податочни типови. Во случај со претставениот веб сервис, не постојат кориснички дефинирани податочни типови, па сите параметри може да се претстават преку вградените XML Schema (XSD) податочни типови.

Наредни два елемента во WSDL описот се <wsdl:message>. Првиот <wsdl:message> елемент го претставува влезниот елемент и има единствено име getMegaPixelSoapIn, специфицирано преку name атрибутот. Во разгледуваниот пример имаме само еден влезен параметар, па затоа имаме дефинирано само еден <wsdl:part> елемент. Елементот <wsdl:part> користи два атрибути за дефинирање

на влезниот елемент. Првиот атрибут е name, кој се користи за специфицирање на името на влезниот параметар. Вториот атрибут е element атрибутот, чија вредност е tns:getMegaPixel. Вредноста всушност е покажувач кон елементот getMegaPixel, дефиниран во <wsdl:types> елементот.

Следен елемент е <wsdl:portType> елементот. Тој има единствено име, дефинирано преку name атрибутот. Во разгледуваниот случај името е Service1Soap. Овој елемент дефинира група од методи кои ги нуди веб сервисот и секој медот е дефиниран со <wsdl:operation> елемент, вгнезден во <wsdl:portType> елементот. Во разгледуваниот пример имаме само еден метод; во спротивно, во WSDL описот ќе имаше повеќе <wsdl:operation> елементи вгнездени во <wsdl:portType> елементот.

Во разгледуваниот пример, елементот <wsdl:operation> има име getMegaPixel. Внатре во овој елемент се дефинирани елементите <wsdl:input> и <wsdl:output>. Елементот <wsdl:input> е покажувач кон влезниот елемент, додека пак <wsdl:output> е соодветниот излезен елемент, т.е. резултатот од веб сервисот. Тие се поврзани користејќи ги единствените имиња од <wsdl:message> елементите.

Како заклучок, <wsdl:portType> елементот може да го дефинираме како елемент кој ја содржи групата од методи кои ги нуди веб сервисот, во нашиот случај наречена Service1Soap. Оваа група содржи еден метод и неговото име е getMegaPixel, неговиот влезен елемент е <wsdl:message> елементот со име getMegaPixelSoapIn и излезен елемент со име getMegaPixelSoapOut.

Следен во структурата на WSDL документот е <wsdl:binding> елементот. Намената на овој елемент е да покаже како да се повикуваат методите кои се наоѓаат во <wsdl:portType> елементот, користејќи посебен протокол. Овој елемент има единствено име дефинирано со name атрибутот. Во случајот кој го разгледуваме неговото име е Service1Soap. Вториот атрибут, type, го специфицира името на <wsdl:portType> елементот. Во овој случај неговото име е tns:Service1Soap. Иако <wsdl:binding> елементот специфицира како клиентот би требало да ги повикува методите кои се вклучени во tns:Service1Soap <wsdl:portType> елементот, овој дел од WSDL документот не е многу поврзан со семантичките веб сервиси.

Последниот елемент од WSDL документот е <wsdl:service> елементот. Тој има атрибут name, кој го специфицира името на сервисот. Во разгледуваниот случај овој веб сервис е наречен Service1. Овој елемент во себе го вклучува <wsdl:port> елементот, кој дава информации за тоа како овој сервис може да се повика. Бидејќи информациите за начинот на повикување на сервисот се веќе дадени во <wsdl:binding> елементот, овој <wsdl:port> елемент служи само како референца кон него. Во случајот тоа е tns:Service1Soap. Последниот елемент кој е

вгнезден во <wsdl:service> елементот е <soap:address> елементот, кој ја специфицира SOAP крајната точка. На овој начин се специфицита локацијата на која клиентот треба да го испрати SOAP барањето. Во примерот кој го разгледуваме, вредноста на URL за сервисот е http://localhost/GetMegaPixelWS/Service1.asmx, што означува дека сервисот е поставен на локалната машина.

3.1.2. Размена на податоци преку SOAP

SOAP претставува протокол кој ги стандардизира и специфицира комуникацијата и размената на податоци помеѓу дистрибуирани апликации [30]. Со користење на SOAP протоколот, односно SOAP пораките, две апликации кои се наоѓаат на различни машини, можат едноставно да воспостават комуникација и ефикасно да разменат пораки, кои ќе бидат прецизно интерпретирани од двете апликации, независно од платформата или програмскиот јазик во кој се развиени. Ваквата независност е овозможена од XML базираната синтакса на SOAP пораките.

За да се овозможи ваквата комуникација преку Веб, SOAP се базира на HTTP протоколот. Со тоа SOAP пораките кои се разменуваат помеѓу две апликации се праќаат и примаат како дел од стандардните HTTP барања и одговори.

Во светот на веб сервисите, SOAP се користи како стандард за размена на податоци помеѓу корисникот кој бара одредена услуга од веб сервис и серверот кој го нуди веб сервисот. SOAP пораките за барањето и одговорот се генерираат автоматски, врз основа на WSDL документот.

За веб сервисот презентиран во претходниот дел, пораките на SOAP барањето и SOAP одговорот се дадени во продолжение.

Изглед и дефиниција на SOAP барањето:

SOAP Request

```
POST  GetMegaPixelWS/Service1.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/getMegaPixel"
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMegaPixel xmlns="http://tempuri.org/">
      <cameraModel>string</cameraModel>
```

```
    </getMegaPixel>
  </soap:Body>
</soap:Envelope>
```

Изглед и дефиниција на SOAP одговорот:

SOAP Response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
```

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getMegaPixelResponse xmlns="http://tempuri.org/">
      <getMegaPixelResult>double</getMegaPixelResult>
    </getMegaPixelResponse>
  </soap:Body>
</soap:Envelope>
```

3.1.3. Тек на активности кај веб сервиси

Постои генерален тек на активности кога е во прашање креирањето, поставувањето и користењето на веб сервиси. Тој се состои од следниве точки:

1. Се креира веб сервисот, со користење на некој програмски јазик.
2. Веб сервисот се објавува преку неговиот WSDL документ, за да потенцијалните клиенти можат да пристапат до него. Генерално, WSDL документот се генерира од страна на некоја алатка за развој на веб сервиси или преку вграден механизам во некоја интегрирана развојна околина.
3. Веб сервисот се поставува на веб сервер.
4. Клиентската апликација (која може да биде напишана во друг програмски јазик) пристапува до WSDL документот, врз основа на кој се генерира порака за SOAP барање.
5. Веб серверот ја прима пораката за SOAP барање како дел од HTTP POST барањето. SOAP пораката се препраќа до веб сервисот.
6. Веб сервисот ја обработува SOAP пораката и креира SOAP одговор, кој го испраќа до веб серверот.

Веб серверот го формира HTTP одговорот, во кој го вклучува и SOAP одговорот, и го испраќа до клиентот.

3.1.4. UDDI: Регистар за веб сервиси

Процесот на креирање, објавување и повикување на веб сервис го објаснивме во претходниот дел. Но, во целата таа постапка се соочуваме со еден проблем: клиентот прво треба да го лоцира сервисот, за да може да го повика.

Да претпоставиме дека ни е потребен веб сервис кој ќе ни ја прикаже моменталната температурата на даден аеродром некаде во светот. Овој веб сервис како влезен елемент би требало да го добие кодот на аеродромот, а како резултат да ја врати температурата. Иако на прв поглед еден ваков веб сервис делува тривијално по својата функционалност, неговата важност расте со фактот што температурата постојано варира, па со тоа постои потреба од негово постојано повикување и искористување.

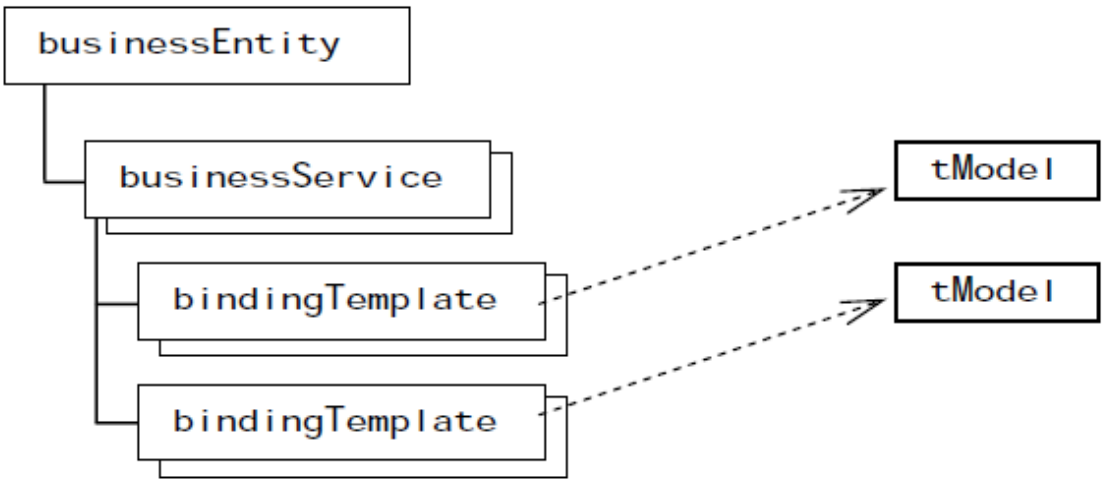
Проблемот со кој што веднаш се соочуваме е фактот дека ние не знаеме дали воопшто постои и каде е лоциран еден веб сервис со таква функционалност, па првиот чекор треба да биде негово лоцирање. За таа цел постои глобален регистар на веб сервиси, дизајниран за нивно откривање. Овој глобален регистар е наречен „Универзален опис, откривање и интеграција“ (Universal Description Discovery and Integration – UDDI, анг.) [31].

Како што кажува и самото име, главната функционалност на UDDI е обезбедување на поддршка за објавување и пронаоѓање на веб сервиси, кои се достапни на глобално ниво.

Пред да дискутираме околу деталите на UDDI, а имајќи го предвид постоењето на ваков глобален регистар на веб сервиси, потребно е да направиме извесни промени во текот на активностите поврзани со креирањето, поставувањето и користењето на веб сервиси:

1. Се креира веб сервисот, со користење на некој програмски јазик.
2. Авторот на веб сервисот го објавува веб сервисот во UDDI репозиториумот.
3. Веб сервисот се поставува на веб сервер.
4. Клиентската апликација (која може да биде напишана во друг програмски јазик) го пребарува UDDI регистарот и го пронаоѓа сервисот.
5. Клиентската апликација ја генерира пораката за SOAP барање, врз основа на WSDL документот преземен од UDDI.
6. Веб серверот го прима SOAP барањето како дел од HTTP POST барањето и истото го препраќа до веб сервисот.

7. Веб сервисот ја обработува SOAP пораката, креира SOAP одговор и го испраќа до веб серверот.
8. Веб серверот го формира HTTP одговорот, во кој го вклучува SOAP одговорот, и го испраќа до клиентот.



Слика 3-1: Основи UDDI податочни типови

Во UDDI регистарот постојат четири главни податочни типови: businessEntity, businessService, bindingTemplate и tModel. Всушност, постои уште еден дополнителен податочен тип наречен publisherAssertion, кој ретко се користи. Поради тоа, не се задржуваме на него. На слика 3-1 се прикажани овие четири главни податочни типови.

Табела 3-1: Атрибути и елементи на businessEntity

Елемент/Атрибут	Значење
businessKey	Атрибут; уникатен идентификатор за инстанцата
authorizedName	Атрибут; име на индивидуата која ја објавува оваа инстанца
Operator	Атрибут; име на UDDI регистарскиот оператор кој управува со главните копии на податоците од businessEntity
discoveryURLs	Елемент; листа на URL адреси кои покажуваат до други достапни механизми за лоцирање на сервиси
Name	Елемент; име на businessEntity (постои можност за претставување на различни јазици)
Description	Елемент; опис на businessEntity
Contacts	Елемент; листа на контакт информации

businessService	Елемент; листа на една или повеќе логички структури за опишување на бизнис сервиси
identifierBag	Елемент; листа од парови име – вредност, кои се користат за чување на идентификаторите на businessEntity
categoryBag	Елемент; листа од парови име – вредност, кои се користат за објавување специфични информации за businessEntity

Податочната структура businessEntity се користи за претставување на ентитетот (означен како бизнис) кој го нуди веб сервисот. За да се објави веб сервис на UDDI, потребно е прво да се објави нов businessEntity, за да може објавувачот на веб сервисот се претстави како бизнис ентитет, односно бизнис единка. Во продолжение е дадена XML шемата на businessEntity структурата. Во Табела 3-1 се прикажани сите атрибути и елементи кои може да ги има една инстанца на businessEntity податочната структура.

XML шема за businessEntity:

```
<element name="businessEntity" type="uddi:businessEntity" />
<complexType name="businessEntity">
  <sequence>
    <element ref="uddi:discoveryURLs" minOccurs="0" />
    <element ref="uddi:name" maxOccurs="unbounded" />
    <element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded"/>
    <element ref="uddi:contacts" minOccurs="0" />
    <element ref="uddi:businessServices" minOccurs="0" />
    <element ref="uddi:identifierBag" minOccurs="0" />
    <element ref="uddi:categoryBag" minOccurs="0" />
  </sequence>
  <attribute name="businessKey" type="uddi:businessKey"
use="required" />
  <attribute name="operator" type="string" use="optional" />
  <attribute name="authorizedName" type="string" use="optional"
/>
</complexType>
```

Атрибутот businessKey е клучен атрибут кој се користи за уникатно идентификување на бизнис ентитетот кој го објавува сервисот. Овој клуч е наречен универзално единствен идентификатор (Universally Unique Identifier – UUID, анг.) и постојат алатки за генерирање на вакви клучеви. Самиот UDDI репозиториум автоматски доделува вредност за клучот, со што нема потреба објавувачот на сервисот да се грижи за негово генерирање.

Една инстанца од businessEntity може да содржи повеќе discoveryURL елементи (кои се деца на discoveryURLs елементот) кои покажуваат кон т.н. документи за откривање. Меѓутоа, овие документи не се поврзани со техничкиот аспект на веб сервисите. Тие претставуваат документи кои нудат дополнителни

информации за самиот бизнис ентитет кој го објавува сервисот. На пример, едно такво URL може да биде веб страната на објавувачот на сервисот.

Елементите identifierBag и categoryBag главно се користат за додавање дополнителна идентификација и категоризација на информациите од businessEntity.

Една инстанца од businessEntity може да не содржи ниту една или да содржи повеќе businessService структури. Една businessService структура претставува еден специфичен сервис кој го нуди businessEntity ентитетот. Во продолжение е прикажана XML шемата за businessService структурата, а во Табела 3-2 се дадени сите атрибути и елементи на структурата.

XML шема за businessService:

```
<element name="businessService" type="uddi:businessService" />
<complexType name="businessService">
  <sequence>
    <element ref="uddi:name" minOccurs="0" maxOccurs="unbounded"
/>
    <element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded" />
    <element ref="uddi:bindingTemplates" minOccurs="0" />
    <element ref="uddi:categoryBag" minOccurs="0" />
  </sequence>
  <attribute name="serviceKey" type="uddi:serviceKey"
use="required" />
  <attribute name="businessKey" type="uddi:businessKey"
use="optional" />
</complexType>
```

Табела 3-2: Атрибути и елементи на businessService

Елемент/Атрибут	Значење
businessKey	Атрибут; референца кон UUID клучот на businessEntity инстанцата
serviceKey	Атрибут; единствен businessService клуч за идентификување на сервисот
name	Елемент; име на сервисот
description	Елемент; опис на сервисот
bindingTemplates	Елемент; референца кон техничките детали за сервисот
categoryBag	Елемент; парови име – вредност, кои служат за категоризирање на сервисот

Можеме да забележиме дека во XML шемата businessKey атрибутот е означен како опционален. Меѓутоа, во пракса е потребно овој клуч да покажува кон serviceEntity инстанцата која го нуди сервисот. Со други зборови, секоја

businessService инстанца треба да биде дете на една serviceEntity инстанца. Друг важен атрибут е serviceKey атрибутот, кој се користи за единствено идентификување на сервисот. Тој е задолжителен и автоматски се генерира и доделува од страна на UDDI регистарот, во форма на долг UUID број. Елементот categoryBag се користи за категоризирање на сервисот.

Елементот bindingTemplate го поврзува овој сервис со неговите технички описи. Секоја businessService инстанца или не содржи ниту една или, пак, содржи повеќе bindingTemplate структури. Една таква структура дефинира технички информации, како што се интерфејсот на сервисот и крајната точка, односно URL адресата на сервисот. Во продолжение е претставена XML шемата за bindingTemplate структурата, а во Табела 3-3 се прикажани сите атрибути и елементи на структурата.

XML шема за bindingTemplate:

```
<element name="bindingTemplate" type="uddi:bindingTemplate" />
<complexType name="bindingTemplate">
  <sequence>
    <element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded" />
    <choice>
      <element ref="uddi:accessPoint" />
      <element ref="uddi:hostingRedirector" />
    </choice>
    <element ref="uddi:tModelInstanceDetails" />
  </sequence>
  <attribute name="serviceKey" type="uddi:serviceKey"
use="optional" />
  <attribute name="bindingKey" type="uddi:bindingKey"
use="required" />
</complexType>
```

Табела 3-3: Атрибути и елементи на bindingTemplate

Елемент/Атрибут	Значење
bindingKey	Атрибут; уникатен клуч за bindingTemplate инстанцата
serviceKey	Атрибут; референца кон businessService инстанцата
description	Елемент; опис
accessPoint	Елемент; влезна адреса (URL) за сервисот
hostingRedirector	Елемент; кога не е наведен accessPoint елементот, овој елемент е задолжителен; покажува кон друг bindingTemplate
tModelInstanceDetails	Елемент; складиште кое ги содржи tModel деталите

Секој bindingTemplate е уникатно идентификуван преку системски генериран UUID, а неговата вредност е сместена во bindingKey атрибутот. Иако

serviceKey атрибутот е означен како опционален, вообичаено се користи и покажува кон родителот, т.е. businessService инстанцата.

Елементот <choice> од XML шемата специфицира дека bindingTemplate инстанцата мора да содржи или accessPoint елементи или hostingRedirector елемент. Елементот accessPoint содржи URL адреса преку која може да се пристапи до веб сервисот. Ако не е претставен accessPoint елементот во bindingTemplate инстанцата, тогаш мора да биде претставен hostingRedirector елементот, кој покажува кон друг bindingTemplate.

Елементот tModelInstanceDetails претставува референца кон складиште кој ги содржи сите технички информации (tModel) за даден веб сервис. Всушност, tModel е една од основните податочни структури во UDDI. Во продолжение е претставена XML шемата за tModel.

XML шема за tModel:

```
<element name="tModel" type="uddi:tModel" />
<complexType name="tModel">
  <sequence>
    <element ref="uddi:name" />
    <element ref="uddi:description" minOccurs="0"
maxOccurs="unbounded" />
    <element ref="uddi:overviewDoc" minOccurs="0" />
    <element ref="uddi:identifierBag" minOccurs="0" />
    <element ref="uddi:categoryBag" minOccurs="0" />
  </sequence>
  <attribute name="tModelKey" type="uddi:tModelKey"
use="required" />
  <attribute name="operator" type="string" use="optional" />
  <attribute name="authorizedName" type="string" use="optional"
/>
</complexType>
```

Како класна податочна структура во UDDI, tModel не може да биде дете на некој ентитет. Меѓутоа, може да покажува кон кој било друг ентитет. Секој tModel е единствено идентификуван преку tModelKey, кој се генерира автоматски и претставува UUID низа од карактери. Освен клуч, tModel има и име (name) и опционален елемент за опис (description). Елементот overviewURL најчесто покажува кон документ кој го опишува интерфејсот на сервисот.

Улогата и функцијата на tModel најдобро може да се увиди преку пример. Да се вратиме на веб сервис чиј што WSDL опис го дадовме во претходните делови. Тоа беше веб сервис кој за внесен модел на камера како резултат ја враќаше вредноста на мегапикселите на моделот на камерата. Од текот на активностите при работа со веб сервиси, исто така претставен во претходните делови, знаеме дека за да потенцијални клиенти може да го користат овој веб сервис, потребно е тој да се објави во UDDI.

Всушност, за да се објави во UDDI како веб сервис, прво треба да се креира сервисен тип. Сервисниот тип претставува едноставен интерфејс. Целта на овој интерфејс е да му покаже на светот дека е понуден веб сервис и дека може да биде откриен во UDDI регистарот. Овој сервисен тип е регистриран во UDDI репозиториумот со креирање на tModel податочна структура. Во продолжение е претставена tModel податочната структура за пример кој го разгледуваме.

```
<tModel tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">
  <name>getMegaPixel</name>
  <description>interface for camera Web service</description>
  <overviewDoc>
    <description xml:lang="en">URL of WSDL document</description>
    <overviewURL>
        http://localhost/GetMegaPixelWS/Service1.asmx
    </overviewURL>
  </overviewDoc>
</tModel>
```

Сега, откако е креиран сервисниот тип, односно интерфејсот, и откако е регистриран во UDDI, веб сервисот може да се објави во UDDI. Креираниот tModel ќе биде референциран во bindingTemplate инстанцата на веб сервисот. Во продолжение е претставена целосната businessEntity инстанца.

Инстанца на businessEntity за разгледуваниот пример:

```
<businessEntity
businessKey="uuid:AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAA"
operator="someOperatorName"
authorizedName="somePeronsName">
  <name>someCompanyName</name>
  <discoveryURLs>
     <discoveryURL useType="businessEntity">
           http://www.someWebSite.com/someDiscoveryLink
      </discoveryURL>
  </discoveryURLs>
  <businessServices>
    <businessService
    serviceKey="uuid:BBBBBBBB-BBBB-BBBB-BBBB-BBBBBBBBBBBB"
    businessKey="uuid:AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAA">
      <name>getMegaPixel</name>
      <description>returns the mega-pixel for a
model</description>
      <bindingTemplates>
        <bindingTemplate
          bindingKey="uuid:CCCCCCCC-CCCC-CCCC-CCCC-CCCCCCCCCCCC"
          serviceKey="uuid:BBBBBBBB-BBBB-BBBB-BBBB-BBBBBBBBBBBB">
          <accessPoint URLType="http">
            http://localhost/GetMegaPixelWS/Service1.asmx
          </accessPoint>
          <tModelInstanceDetails>
            <tModelInstanceInfo
               tModelKey="uuid:5DD52389-B1A4-4fe7-B131-
0F8EF73DD175">
              <instanceDetails>
```

```
                    <overviewDoc>
                      <overviewURL>
                        http://localhost/GetMegaPixelWS/Service1.asmx
                      </overviewURL>
                    </overviewDoc>
                  </instanceDetails>
                </tModelInstanceInfo>
              </tModelInstanceDetails>
            </bindingTemplate>
          </bindingTemplates>
        </businessService>
      </businessServices>
    </businessEntity>
```

Вака би изгледал веб сервисот во UDDI регистарот. Прво е опишан бизнисот, а потоа сервисот кој се нуди. Овој сервис има специфичен интерфејс опишан преку tModel со клуч uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175. За претставување на UUID, заради поголема прегледност на кодот користени се клучеви од типот uuid:AAAA..., uuid:BBBB..., и uuid:CCCC...

Како што може да се види од примерот, основна цел на UDDI репозиториумот е да им помогне на крајните корисници на Вебот да ги пронајдат потребните функционалности во форма на веб сервиси.

3.1.5. Користење на UDDI за лоцирање на веб сервиси

Да претпоставиме дека одреден клиент сака да пронајде веб сервис кој како влезен атрибут прима модел на камера, а како резултат добива вредност која ги претставува мегапикселите на дадениот модел. Без да знае дали и каде постои таков веб сервис, клиентот одлучува да побара во UDDI регистарот.

Меѓутоа, пребарувањето не е едноставен процес. Причината за тоа е фактот дека за да се пронајде веб сервисот, потребно е заинтересираниот клиент да знае барем една од следните информации:

- tModelKey клучот на интерфејсот кој го имплементира нашиот сервис;
- Името на сервисот, getMegaPixel;
- Еден од следниве клучеви: businessKey, serviceKey или bindingKey;
- Името на WSDL документот или неговата локација;
- Компанијата која го развила овој сервис;

Клиентот најчесто нема вакви информации, па оттука пребарувањето во UDDI регистарот за нови сервиси е многу тешко. Меѓутоа, кога клиентот знае дека даден сервис постои и има некои информации за него (на пример, која компанија го развивала и објавила), многу полесно се пронаоѓа веб сервисот.

Дизајнерите на UDDI се свесни за проблемите со пребарување, особено тешкотиите со кои се соочува клиентот кога сака да го користи UDDI за да пребара нови веб сервиси за кои нема никакви информации. За да се подобри пребарувањето, UDDI вклучува два типа на дополнителни информации за именување, идентификација и категоризирање на информациите, претставени преку identifierBag и categoryBag елементите.

3.1.6. Категоризирање на сервисниот тип (интерфејсот)

Прва идеја е додавање на дополнителни информации за категоризација во сервисниот тип (интерфејсот) tModel. Иако дознавањето на tModelKey клучот е невозможно (клиентот не знае дека овој интерфејс воопшто постои), сé уште е возможно да го пронајдеме извршувајќи пребарување базирано на информациите за категоризација. На пример, tModel интерфејсот од нашиот пример можеме да го класифицираме во категоријата “online information services”. Ако клиентот ја пребара оваа категорија, нашиот tModel интерфејс ќе се прикаже во резултатите.

Идејата секој ентитет кој објавува сервис да користи свој сопствен систем за категоризирање, не е баш добра. Тоа би го направило пребарувањето уште покомплицирано, поради тоа што клиентот треба да прави претпоставки за тоа каков систем за категоризирање користел ентитетот кој го објавил сервисот кој му е потребен. Добро решение на ова е користење на некаков вид на стандардизирана категоризација.

За таа цел, UDDI обезбедува неколку шеми за класифицирање. Една таква шема е шемата за класифицирање на tModel интерфејсите. Да се вратиме на разгледуваниот пример, односно на tModel-от кој претставува интерфејс за веб сервис опишан преку WSDL документ. Една возможна категоризација е да се класифицираат сите tModel интерфејси. Овој систем за класификација е наречен uddi-org:types и вклучува неколку различни типови на вредности, како што се wsdlSpec, xmlSpec и protocol.

Но, пред да се искористи оваа класификација за пронаоѓање на сервис, потребно е да се реши уште еден проблем: како да се претстави овој категоризирачки систем во UDDI? Одговорот е да се користи tModel. Поточно, за да ја претставиме оваа класификација во UDDI, се креира и повторно регистрира специјален tModel. Тој tModel ги има следниве карактеристики:

- поддржан е од секој UDDI регистар,
- секогаш го користи истиот клуч, со вредност UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4.

Во продолжение е претставено додавањето на оваа категоризација во tModel интерфејс на примерот кој го разгледуваме. Додавање wsdlSpec категоризација во tModel интерфејсот:

```
<tModel tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">
  <name>getMegaPixel</name>
  <description>interface for camera Web service</description>
  <overviewDoc>
    <description xml:lang="en">URL of WSDL document</description>
    <overviewURL>
              http://localhost/GetMegaPixelWS/Service1.asmx
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey="UUID:C1ACF26D-9672-4404-9D70-
39B756E62AB4"
      keyName="specification for a Web service described in WSDL"
      keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```

Овој интерфејс е категоризиран како wsdlSpec тип, што претставува спецификација за веб сервиси опишани преку WSDL, користејќи го uddi-org:types системот за категоризирање. Овој систем за категоризирање е претставен во tModel–от чиј tModelKey е UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4.

За додавање на категоризацијата го користиме keyedReference елементот, кој е дете на categoryBag елементот. Генерално, categoryBag елементот се користи за додавање категоризација или информации за класификација на инстанца од дадена UDDI податочна структура. Целта е да бидеме сигурни дека пребарувањето базирано на информациите од категоризација успешно ќе ја лоцира инстанцата.

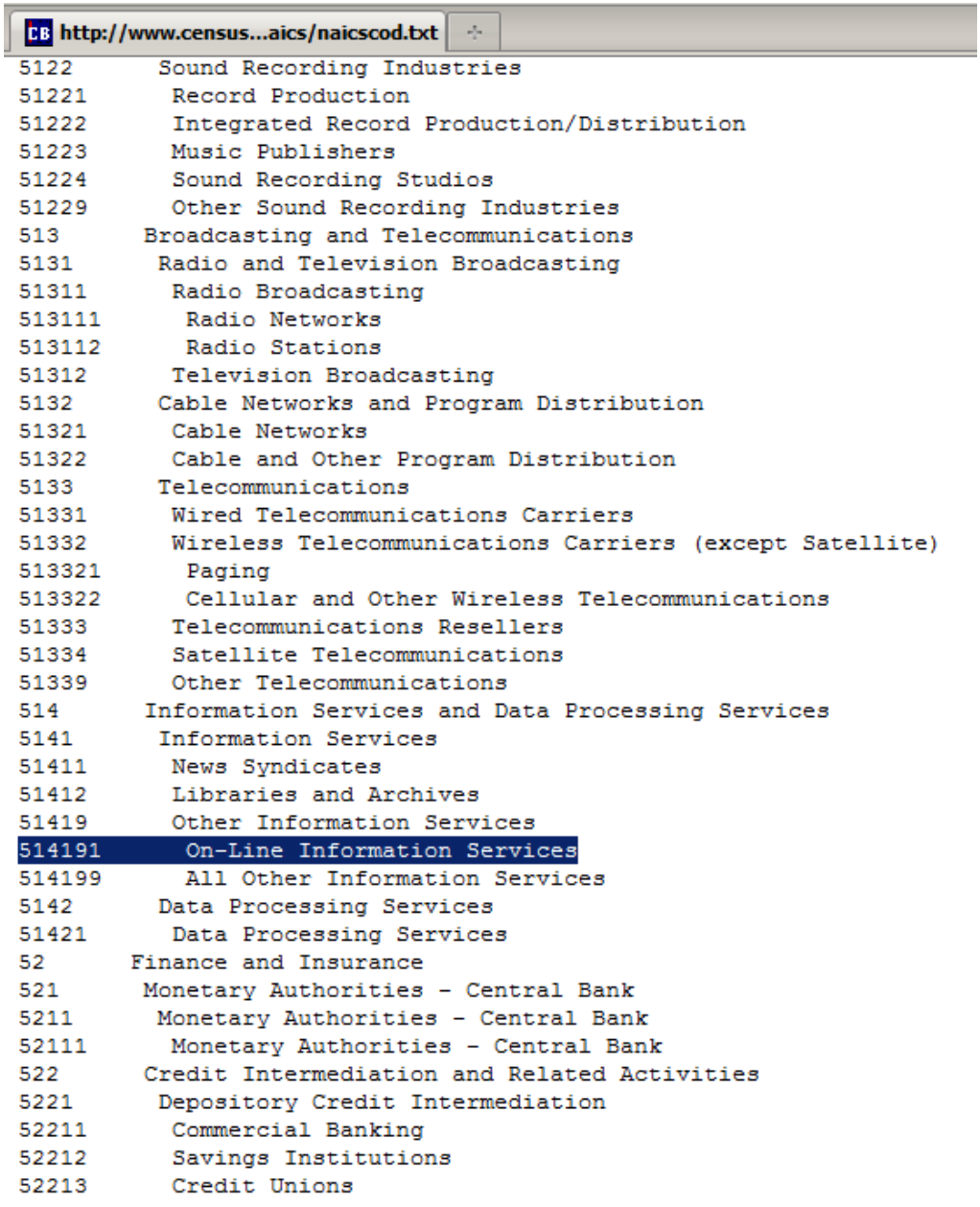
Ако ја погледнеме XML шемата за categoryBag податочната структура, ќе заклучиме дека можеме да додадеме неограничен број информации за категоризација во categoryBag структурата.

Во продолжение е прикажано користење на една популарна шема за категоризирање, наречена NAICS1997. Со неа ќе додадеме повеќе класифицирачки информации во нашиот tModel интерфејс.

NAICS1997 е акроним од North American Industry Classification System, систем реализиран во 1997 година. Тој обезбедува множество од кодови за класифицирање и идентификување на категорија за специфичен веб сервис. И оваа класификација е претставена преку tModel, чие име е ntisgov.naics:1997, кој има клуч tModelKey со вредност UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2.

За да можеме да го користиме NAICS1997 системот за класифицирање на нашиот tModel интерфејс, прво мора да го идентификуваме бизнис типот на веб сервисот кој го претставува интерфејсот. Еден начин да се пронајде добар бизнис

за класифицирање, е да се посети официјалната страна на NAICS [32]. На слика 3-2 се прикажани дел од кодовите кои можат да се користат.



Слика 3-2: NAICS кодови

Бидејќи “On-Line Information Services” одговара како класификација на веб сервисот кој го користиме како пример, го додаваме во нашиот tModel интерфејс.

Додавање на NAICS категоризација во tModel интерфејсот:

```
<tModel tModelKey="uuid:5DD52389-B1A4-4fe7-B131-0F8EF73DD175">
  <name>getMegaPixel</name>
```

```
  <description>interface for camera Web service</description>
  <overviewDoc>
    <description xml:lang="en">URL of WSDL document</description>
    <overviewURL>
      http://localhost/chap11/GetMegaPixelWS/Service1.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference
        tModelKey="UUID:C1ACF26D-9672-4404-9D70-39B756E62AB4"
        keyName="specification for a Web service described in
WSDL"
        keyValue="wsdlSpec"/>
    <keyedReference
        tModelKey="UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
        keyName="On-Line Information Services"
        keyValue="514191"/>
  </categoryBag>
</tModel>
```

На овој начин, во едноставен tModel интерфејс на разгледуваниот пример, сега имаме две различни класификации. Првата покажува дека tModel интерфејсот е од тип wsdlSpec, што означува дека веб сервисот е опишан преку WSDL документ, додека втората категоризација ја користи NACIS1997 шемата за категоризирање и го користи сервисот со вредност 514191, со кој веб сервисот се става во “On-Line Information Services” категоријата.

Крајниот резултат од категоризацијата е подобрен механизам за пребарување: доколку клиентот пребарува tModel интерфејси од тип wsdlSpec, во резултатите од пребарувањето ќе го добие и веб сервисот кој го разгледуваме како пример. Истото се случува и доколку пребарува низ категоријата “On-Line Information Services”. Без додавање информации за категоризација на tModel интерфејсот, не постои начин потенцијалниот клиент да го лоцира интерфејсот на овој, или кој било друг веб сервис.

Сепак, постои можност за уште поголемо олеснување на пребарувањето. Тоа се постигнува со додавање информации за категоризација во categoryBag елементот од секоја UDDI податочна структура, со должина колку што дозволува самата податочна структура. Структурата businessEntity е UDDI податочна структура кај која categoryBag елементот е опционален, па во него можеме да додадеме информации за категоризација, за да една инстанца од businessEntity биде полесно пронајдена од потенцијалните клиенти.

Според тоа, можеме да направиме промена на претходно прикажаниот businessEntity, со додавање на информации за категоризација како што е прикажано во продолжение.

Инстанца од businessEntity со информации за категоризација:

```
<businessEntity
businessKey="uuid:AAAAAAAA-AAAA-AAAA-AAAA-AAAAAAAAAAAA"
operator="someOperatorName" authorizedName="somePeronsName">
  <name>someCompanyName</name>
  <discoveryURLs>
    ... ...
  </discoveryURLs>
  <businessServices>
    ... ...
  </businessServices>
  <categoryBag>
    <keyedReference
            tModelKey="UUID:C0B9FE13-179F-413D-8A5B-5004DB8E5BB2"
            keyName="On-Line Information Services"
            keyValue="514191"/>
  </categoryBag>
</businessEntity>
```

3.1.7. Додавање информации за идентификација на сервисниот тип

Досега презентиравме како може да се додадат информации за категоризација, со цел да се подобри пребарувањето во рамките на UDDI регистарот. Покрај метод на категоризирање, пребарувањето на објавениот веб сервис во UDDI може да се подобри и со додавање на идентификатори на UDDI податочните структури. На пример, за уникатно идентификување на даден бизнис ентитет, може да се користи federal tax ID идентификатор. Овие дополнителни информации се информации за идентификација.

Додавањето на идентификатори на UDDI податочните структури се изведува преку користење на identifierBag елементот и се користи на сличен начин како и categoryBag елементот. Единствена разлика е што identifierBag елементот содржи информации за идентификација на ресурсот, додека пак categoryBag елементот содржи информации за категоризација.

Популарен систем кој се користи за идентификација е DUN-S Number идентификацискиот систем, претставен преку tModel наречен dnb-com:D-UN-S. Овој tModel има клуч tModelKey со вредност: UUID:8609C81E-EE1F-4D5A-B202-3EB13AD01823. Начинот на негово користење е даден во продолжение.

Користење на identifierBag елементот за додавање информации за идентификација:

```
<businessEntity businessKey="...">
...
  <identifierBag>
    <keyedReference keyName="IBM Corporation"
      keyValue="00-136-8083"
      tModelKey = "uuid:8609C81E-EE1F-4D5A-B202-3EB13AD01823"/>
    ...
```

```
    </identifierBag>
...
</businessEntity>
```

3.2. Од класични веб сервиси до семантички веб сервиси

Според досега кажаното, може да се заклучи дека процесот од креирање на веб сервис до негово искористување од страна на потенцијален клиент не е тривијален. Постојат голем број акции кои треба да се преземат, во рамките на кои авторот и корисникот на веб сервисот треба да се надминат и голем број пречки и ограничувања.

Со појавата на технологиите на Семантичкиот Веб се отвора можност дел од акциите поврзани со веб сервисите да се олеснување и автоматизираат. Иако првичната идеја и намена на овие технологии е додавање значење, односно семантика на ресурсите на Вебот, преку семантичка анотација, можностите за искористување на овој пристап за подобрување и поедноставување на текот на активности кај веб сервисите, се многу големи.

Некои од главните идеи и подобрувања кои технологиите на Семантичкиот Веб можат да ги обезбедат во светот на веб сервисите се:

- **Автоматско лоцирање на веб сервиси.** Пронаоѓањето на вистинскиот веб сервис може да биде тешко, особено кога корисникот на сервисот не знае за неговото постоење. За да биде веб сервисот вистински користен, потребно е да се обезбеди начин за негово лесно лоцирање. Потребно е и лоцирањето да биде што е можно повеќе автоматизирано, со голема точност и ефикасност.
- **Автоматско повикување на сервиси.** Откако ќе се лоцира бараниот сервис, потребно е одреден софтверски агент автоматски да го повика сервисот. Придобивките од ваквиот пристап се очигледни: без потребата од човечка интервенција, која меѓу другото внесува и доцнење во процесот, може да се зголеми ефикасноста на дистрибуираните системи и веб сервисите, како и ефикасноста на клиентите кои ги користат. Исто така, во многу случаи, автоматското повикување на сервиси е неопходно, со цел да се задоволат потребите на апликациите кои имаат потреба од податоци во реално време и треба непрекинато да работат.
- **Автоматска композиција на веб сервиси.** Многу често, потребно е повеќе веб сервиси да работат заедно, со цел да извршат специфично, покомплексно барање на клиентот. Потребно е одреден софтверски агент

да ги пронајде сите потребни сервиси и да ги повика во правилен редослед за да ја постигне посакуваната цел.

- **Автоматско мониторирање на извршените сервиси**. Доколку сите процеси се автоматизирани, тогаш се јавува прашањето: како ќе знаеме дали бараниот сервис е пронајден и успешно и точно извршен? За таа цел, потребен е механизам за автоматско детектирање и справување со возможни грешки и прекини во работата на целиот систем.

Во светот на веб сервисите, UDDI е единствен начин за пронаоѓање на бараниот сервис. Меѓутоа неговото користење за пронаоѓање на сервис е, во извесна смисла, проблематично, а во одредени случаи дури и невозможно, поради следните причини:

- UDDI е пребарувач според клучен збор, што значи дека при пребарувањето може да се случи корисникот да не пронајде ниту еден веб сервис, или пак, да пронајде огромен број резултати, што ја отежнува одлуката за тоа кој е најсоодветниот сервис.
- Кај различни податочни типови во UDDI се користат различни шеми за категоризирање и идентификување, со цел да се подобри процесот на откривање на веб сервисите. Меѓутоа, тоа бара авторот на веб сервисот да биде запознаен со сите шеми за класификација и идентификација. Дури и тоа да не е проблем, постои голема веројатност два автори на веб сервиси со иста функционалност, да ги категоризираат различно.
- UDDI не нуди друг начин за пребарување на веб сервисите.

Од сево ова може да се заклучи дека користењето на UDDI за пребарување на веб сервиси е релативно комплициран процес, кој е склон на грешки.

Сепак, овие забелешки не го прават UDDI неуспешен проект. Доколку, на пример, потенцијалниот клиент има некакви информации за даден веб сервис, како што е клучот на интерфејсот, или пак вредноста на клучот на businessEntity инстанцата, тој може да пронајде детални информации за сервисот, со користење на UDDI. На тој начин, со користење на добиениот резултат од пребарувањето од UDDI, клиентот може да дознае како може да му пристапи на соодветниот WSDL документ, кој нуди доволно информации за тоа како да се повика и искористи веб сервисот.

Без процес на автоматско лоцирање на веб сервиси, останатите цели – автоматско повикување, автоматска композиција и автоматско мониторирање – е невозможно да се реализираат.

Пронаоѓањето на бараниот веб сервис е една од главните мотивации за искористување на технологиите на Семантичкиот Веб во областа на веб сервисите. Со постојаниот и огромен раст на содржините на Вебот, пребарувањето на информации во негови рамки е навистина отежнато, со што се појавува и потребата од додавање значење, т.е. семантика на веб ресурсите, со цел да се направи порелевантен и поефикасен модел на пребарување. Истиот принцип може да се употреби и за додавање семантика кај веб сервисите, со цел да се овозможи нивно автоматско лоцирање. За да се додаде семантика кај веб сервисите, потребно е да се додаде семантички опис или на WSDL документите или во UDDI регистарот. Ваквото додавање на семантички опис кај веб сервисите, се нарекува *семантичка анотација на веб сервиси*.

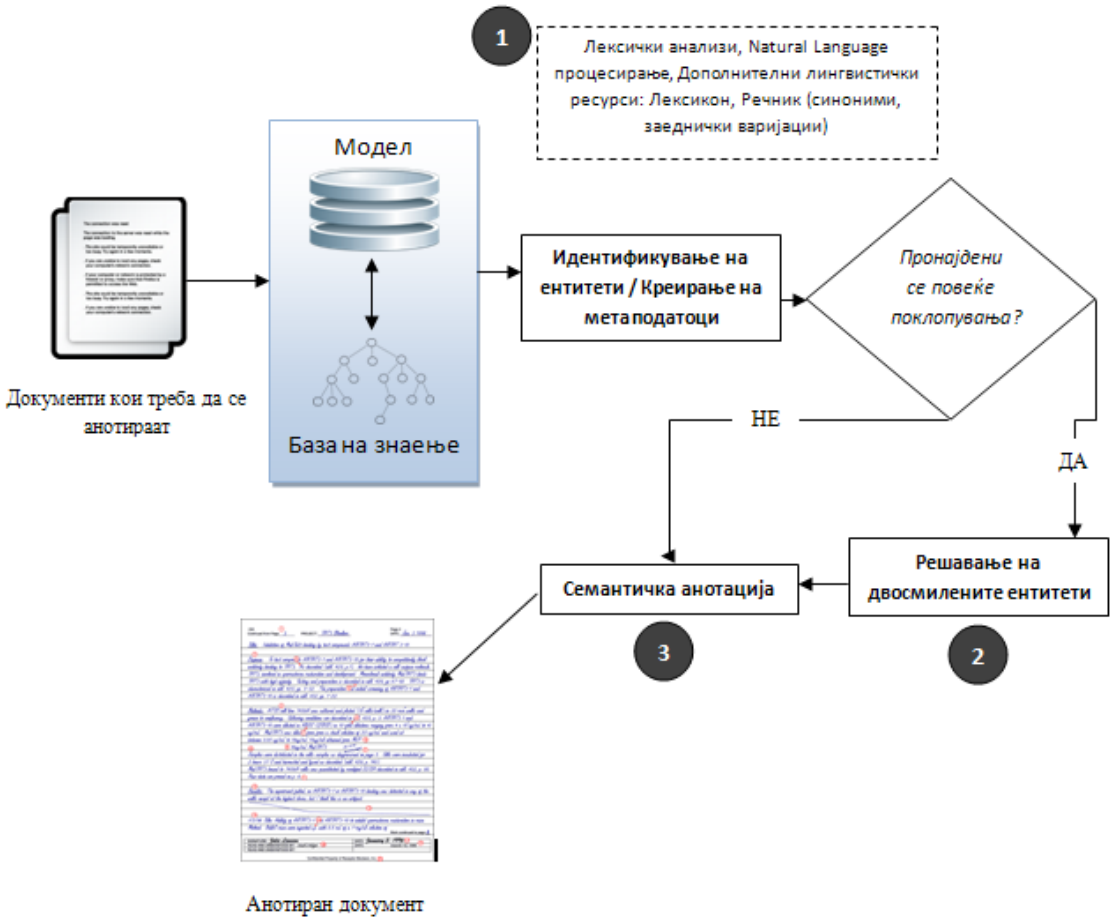
3.2.1. Семантичка анотација

Основниот концепт при реализацијата на визијата за Семантичкиот Веб е снабдување со метаподатоци и поврзување на метаподатоците со веб ресурсите. Процесот на поврзување на метаподатоците со ресурсите (аудио, видео, структуриран текст, неструктуриран текст, веб страна, слики, итн.) е наречено *анотација* и семантичката анотација, всушност, претставува процес на анотирање на ресурсите со *семантички метаподатоци*.

Семантичката анотација грубо може да се класифицира како формална или неформална. Формалната семантичка анотација, наспроти неформалната, следи механизми за репрезентација на концептуални модели, претставени преку користење на добро дефиниран јазик за репрезентација на знаењето. Таквата формална анотација на веб ресурсите лесно може да се процесира од машините и да резултира со големи подобрувања и можности за автоматизација на пребарувањето, јасно откривање на ресурси, анализа на информации, итн.

Семантичката анотација на ресурси поддржана од сегашниот модел, односно од онтолошката шема која обезбедува согласност и јасен модел за собирање податоци и метаподатоци, и подржана и од база на знаење, која содржи инстанци од онтологијата, се состои од три основни чекори: идентификација на ентитети, решавање на двосмислените ентитети и анотација. Овие три чекори, претставени на слика 3-3, се зависни од типот на документот кој се анотира.

На пример, процесот на идентификација на ентитети кои треба да бидат анотирани од текстуален документ е различен од процесот на идентификација на потенцијални ентитети од документ со експериментални податоци. Но, идејата и крајната цел е иста. Заради поедноставување, ресурсот во примерот кој ќе го разгледаме е текстуален документ. Притоа, се користи една онтологија, иако не постојат ограничувања на бројот на онтологии кои корисникот може да ги користи при семантичка анотација на еден ресурс.



Слика 3-3: Семантичка анотација на документи

3.2.1.1. Идентификација на ентитети

Процесот на идентификација на ентитет, претставен како чекор 1 од слика 3-3, вклучува извлекување на корисни информации од документот со помош на граматика базирана на правила, техники за процесирање на природни јазици, кориснички дефинирани шеми, итн. Дополнително, извлечените податоци за ентитетите користат и пополнета онтологија (информации на ниво на инстанци, наречени *база на знаење*, која се пополнува користејќи ја онтолошката шема) за да ги извлечат специфичните инстанци од различни класи. Пристапот на слика 3-3 користи комбинација од постоечки онтологии и база на знаења, лексикони и техники за процесирање на природни јазици.

Слика 3-4: Идентификација на ентитети во неструктуриран документ

Кога се идентификува ентитетот во документот, се извршува проверка за да се види дали тој ентитет веќе постои како инстанца во базата на знаење. Варијации од ентитети, како што е присуството на префикси или суфикси, кратенки (на пример: US за United States), итн., исто така се земени предвид при пребарување на соодветни инстанци во базата на знаење. На слика 3-4 се прикажани идентификуваните ентитети во еден CNN бизнис напис и соодветните класи од Stock онтологијата, која се користи во примерот. Ентитетите кои ни се од интерес,

се подвлечени со сина боја, а класите од онтологиите кои се поврзани со нив, се означени со сива боја. На пример, New York е инстанца од класата City; Microsoft е инстанца од класата Company, итн.

За да процесот на идентификација на ентитети се направи лесно проширлив, односно поскалабилен, како и специјализиран на одредена област, користењето на базата на знаење, проширена со новите идентификувани ентитети, дозволува корисниците да ги видат врските помеѓу овие нови идентификувани ентитети, кои не се присутни во самиот документ. На пример, фактот дека Microsoft и Oracle се конкуренти (слика 3-4), го нема во документот и е достапен само за корисникот, бидејќи е претставен во базата на знаење.

3.2.1.2. Решавање на двосмислени ентитети

При анотација на неструктурирани податоци, често се случува за еден ентитет, идентификуван во документот, да постојат повеќе референци во базата на знаење. На пример, за инстанцата John Smith, идентификувана во документот, може да има две инстанци на John Smith во базата на знаење, една за, на пример, финансиски аналитичар и друга за раководител на дадена компанија. Информациите кои се однесуваат на ентитетот John Smith во документот може да не се совпаѓаат со информациите достапни за истиот ентитет во базата на знаење. На пример, документот не мора експлицитно да го споменува John Smith како раководител во компанијата, но може да постои напис за стратегиите во компанијата, во која John Smith се споменува како раководител. Во таков случај, потребни се софистицирани методи за да се соберат информациите за текстот во кој лицето John Smith е споменато. Различни податочни извори имаат различни начини на претставување на еден ист ентитет. Варијациите при претставувањето вообичаено растат со погрешно пишување на поимите, користење на кратенки, различни конвенции за именување, варијации при именување со текот на времето, користење на различни јазици, итн. Решавањето на вакви двосмислени ентитети, претставено како чекор 2 на слика 3-4, е процес на идентификација кога различни референци покажуваат кон еден ист ентитет. Постојат повеќе пристапи за решавање на овој проблем, во зависност од природата на податочните извори, како и нивото на потребна точност кое треба да се постигне.

Во овој пример, даден на слика 3-4, потребно е во документот да се решат идентификуваните двосмислени ентитети и повеќето кандидати кои референцираат кон него, а кои се пронајдени во онтологијата. Широкото користење на информациите од текстот обезбедува најдобар доказ за усогласени одлуки. Текстот на ентитетот споменат во документот може да биде дефиниран во термини од текстот на документот, класификацијата на документот во хиерархијата од предмети, итн., за да се дознае за што се „зборува“ во документот.

Текстот на ентитетот во базата на знаење може да биде дефиниран во вид на вредности за атрибути од ентитетот, кои му припаѓаат нему и релациите во кои учествува. На пример, ако ентитетот "BEAS" кој се појавува во документот, има две инстанци во онтологијата, како "Bureau of Elder and Adult Services BEAS: организација" и "BEAS: симбол за BEA системи", собирањето на текстот во кој "BEAS" се појавува во документот е поврзано со BEA Systems и може да помогне при решавање на проблемот со двосмисленоста на двете референци кои се јавуваат во онтологијата.

3.2.1.3. Анотација

По решавањето на двосмислените ентитети, наредниот чекор е поврзување на семантичките метаподатоци со ентитетите во документот, преку процес на *анотација*. Анотацијата се прави со користење на W3C препорачани стандарди, како што се RDF (Resources Description Framework) и OWL (Web Ontology Language), кои служат за семантичко опишување на неструктурирани податоци и кои беа објаснети во претходните делови. Во продолжение се прикажани едноставни метаподатоци за неколку ентитети од примерот на слика 3-4. Од анотацијата можеме да видиме дека, на пример, ентитетот 'Hewlett-Packard' е инстанца од класата 'company', 'HPQ' е 'tickerSymbol', итн.:

```
<Entity id="494805" class="company">Hewlett-
Packard</Entity>(<Entity id= "875349"
class="tickerSymbol">HPQ</Entity>: up <Regexp
type="money">$0.33</Regexp> to <Regexp
type="money">$13.03</Regexp>, Research, Estimates) said a
report shows its share of the printer market grew in the
second quartet, although another report showed that its
share of the computer sender market declined in <Entity
id="7852" class="continentRegion">Europe</Entity>, the
<Entity id="7854" class="continentRegion">Middle
East</Entity'>
```

При процесот на идентификација на ентитети во документот, можно е да бидат пронајдени вредности за атрибути или релации кои претходно не биле присутни во базата на знаење. Во тој случај, може да се направи проширување на базата на знаење со нови податоци. Во зависност од типот на новите податоци, проширувањето на базата на знаење може да биде едноставно како внесување вредности за атрибути, при што процесот може да биде автоматизиран, или пак комплексно како менување на основната шема, при што се потребни дополнителни човечки интервенции.

3.2.2. Апликации за семантичка анотација

На полето на семантичка анотација, до сега се направени големи напори за градење на скалабилни, автоматизирани семантички платформи, кои служат за

анотација. Повеќето од овие системи главно се фокусирани на мануелна или полуавтоматска анотација и имаат за цел да им ја олеснат работата на луѓето. Меѓутоа, дури и со помош од страна на софтвер и компјутер, анотацијата на содржини е тешка задача, која бара многу време и која е подложна на грешки.

Покрај семантичкото анотирање на содржините, голем број на апликации обезбедуваат и складиште од анотации и онтологии, кориснички интерфејси, и програмски интерфејси за пристап до одредени функционалности кои обезбедуваат целосна поддршка за користење на анотации. Она што е карактеристично за овие апликации е користењето на различни техники за извлекување на информациите. Правила, шеми за откривање, машинско учење и почетно пополнета онтологија, се само дел од технологиите кои се користат.

Примерите за вакви апликации ги вклучуваат SemTag [33], The SHOE Knowledge Annotator [34], AeroDAML [35], Semantic Enchancement Engine (SEE) [36], OntoAnnotate [37], COHSE [38], CREAM [39][40], Annotea [41], KIM [42], итн. Во Табела 3-4 е дадена споредба на дел од алатките, врз основа на технологиите кои тие ги користат.

Табела 3-4: Платформи за семантичка анотација

Платформа	Метод	Машинско учење	Рачни правила	Пополнета онтологија
AeroDAML	Правила	Не	Да	WordNet
Armadillo	Шеми за откривање	Не	Да	User
KIM	Правила	Не	Да	KIMO
MnM	Код обвиткувачи	Да	Не	KMi
MUSE	Правила	Не	Да	User
Ont-O-Mat: Amilcare	Код обвиткувачи	Да	Не	User
Ont-O-Mat: PANKOW	Шеми за откривање	Не	Не	User
SemTag	Правила	Не	Не	TAP

SemTag е платформа за анализа на големи текстови [33]. Оваа платформа се користи за автоматско семантичко анотирање на податоците кај големи корпорации, со користење на TAP онтологијата [43]. Исто така, користи и алгоритам за отстранување на двосмисленост кај поими, специјализиран за

разрешување на двосмисленост во големи количества податоци. Анотациите се претставени со RDF Schema [11].

SHOE е еден од постарите системи за додавање на семантичка анотација на веб страните и им овозможува на корисниците да ги означуваат страните во SHOE со помош на онтологии достапни локално или преку URL [34]. Овие анотирани страници потоа може да бидат разбрани од SHOE алатки, како што е SHOE Search: Semantic Search – The SHOE Search Engine.

OntoAnnotate нуди детална поддршка за креирање на семантички поврзани метаподатоци од страна на луѓе [37]. При идентификација на ентитети на веб страните, користи комбинација од код обвиткувачи, шеми за поврзување и информации од онтологии извлечени со процесирање на текстот. Исто така, во оваа платформа е вклучен и систем за управување со документи, кој ги складира анотираните документи и нивните метаподатоци во RDF формат [7].

Knowledge and Information Management – KIM платформата обезбедува инфраструктура за управување со знаења и информации, како и сервиси за автоматска семантичка анотација, индексирање и извлекување на неструктурирани и полуструктурирани содржини [42]. Оваа платформа ги анализира текстовите, ги препознава референците на ентитетите и се обидува да ги поврзе со познати ентитети од онтологии. Референцата во документот се анотира со URI кон ентитетот. KIM користи онтологија од повисоко ниво, наречена PROTON и користи база на знаење наречена KIM. Освен автоматска семантичка анотација, KIM овозможува и извлекување на содржини, базирани на семантички ограничувања, поставување прашања кон основните онтологии и овозможува модифицирање на основните онтологии и самата база на знаење.

Креирањето на онтологии и на семантички анотации за веб ресурси, се основите на кои се гради Семантичкиот Веб. Покрај текстуалните и дигиталните содржини, најважни веб ресурси се и веб сервисите, кои претставуваат еден вид дистрибуирани ресурси кои на крајните корисници им обезбедуваат одредени *сервиси*. Веб сервисите се есенцијален дел од денешниот Веб, затоа што овозможуваат креирање на функционалности и сервиси кои можат да се искористуваат повторно, од најразлични корисници, при што им се овозможува поедноставен развој на дистрибуирани и сервисно ориентирани апликации.

Улогата на веб сервисите во Семантичкиот Веб е можеби уште позначајна: тие треба да им овозможи на корисниците на Семантичкиот Веб автоматски да наоѓаат одговори на своите барања, без експлицитно и мануелно да ги поминуваат чекорите на наоѓање на постоечки сервиси, нивно комбинирање и повикување во соодветен редослед, со цел да ја извршат посакуваната акција или да ја добијат бараната информација.

Меѓутоа, семантичката анотација на веб сервисите е комплетно различен концепт од концептот на анотација на останатите веб ресурси. Семантиката поврзана со веб сервисите треба да биде формулирана на начин што ќе го направи веб сервисот корисен за апликациите и корисниците кои го користат.

3.3. Семантичка анотација на веб сервиси

Концептот на извршување на бизнис процеси и интеграција на апликации преку веб сервиси, е во постојан раст. Иако веб сервисите се базираат на широко прифатени стандарди, она што во денешно време недостига е формалниот опис за значењето на нивната функционалност, а и податоците кои се разменуваат помеѓу веб сервисите претставуваат затворен пат за поедноставна интеграција помеѓу хетерогени апликации. Како што бројот на веб сервиси расте, важно е да постојат алатки за нивно автоматско откривање и нивна композиција. Под композиција на веб сервиси се мисли на правилно распоредување на редоследот на повикување на повеќе веб сервиси, со цел да се изврши посакуваната акција или да се добие посакуваната информација.

Во моментот, описот на веб сервисите е даден со WSDL датотеки, кои се креираат според WSDL стандардот. Но, самиот WSDL стандард го спречува процесот на автоматско откривање, автоматска композиција и повикување на веб сервисите. Причината за тоа е недоволната флексибилност на стандардот, кој во својата дефиниција не дозволува користење на концептите од Семантичкиот Веб како механизми за додавање значење на податоците со кои работат веб сервисите, како и нивната функционалност.

Еден од начините за да се реши овој проблем е развивање на *семантички јазик за веб сервисите*.

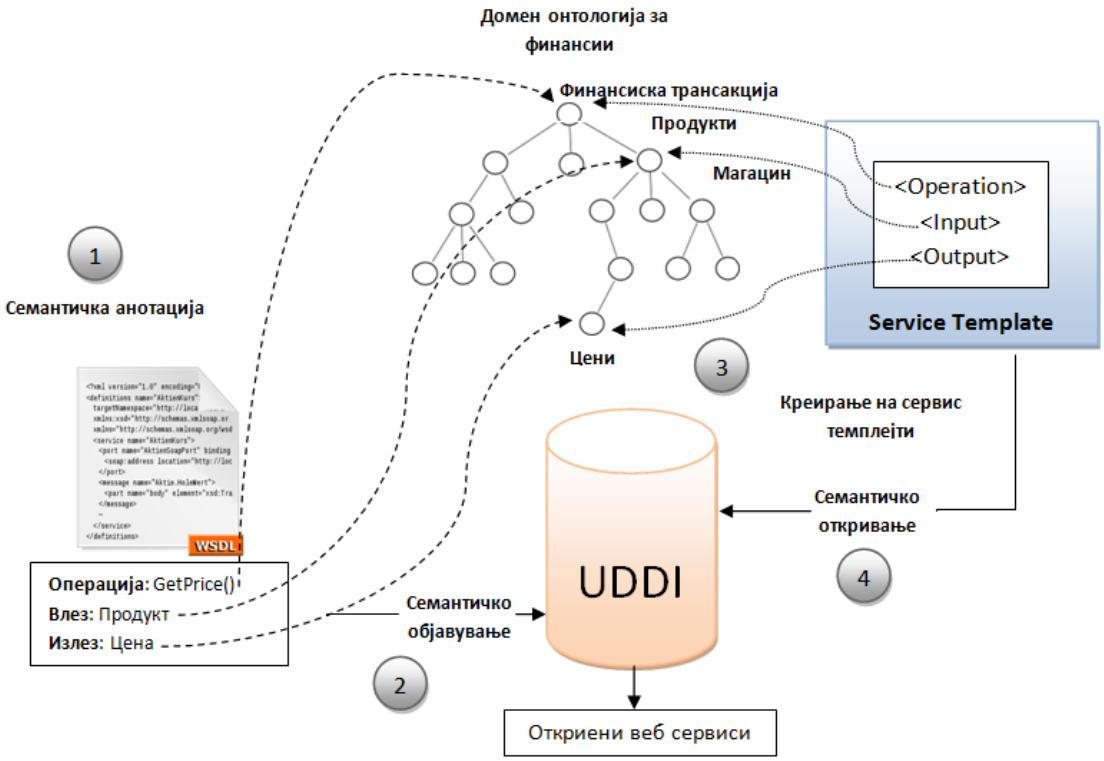
Генерално, постојат два начина за додавање на семантички опис кај веб сервисите. Првиот начин е креирање на семантички описи врз основа на некои универзално прифатени онтологии. Со користење на овие онтологии, може да се опишат најразлични аспекти на секој веб сервис. Софтверскиот агент, кој би се користел за да на крајниот корисник му обезбеди добивање некој сервис преку Веб, ќе има доволно информации за откривање, повикување, композиција и мониторирање на семантички опишаниот веб сервисот, односно *семантичкиот веб сервис*. OWL-S е еден пример за таква онтологија [44].

Вториот начин е да се додаде семантички опис директно во стандардите за веб сервиси, како што се UDDI или WSDL документите. Главната предност на овој пристап е повторно искористување на постоечките стандарди, кои се

прифатени од развивачите на апликации и кои, исто така, имаат широка поддршка од различни произведувачи и продукти.

3.3.1. Анотација на веб сервиси

Семантичката анотација на веб сервисите имплицира развивање на точни семантики за податоците и функционалноста на веб сервисите. Тоа се постигнува со анотирање на елементите на веб сервисот со концепти во одредена област, односно онтологија. Сé додека онтологиите претставуваат поглед за моделирана област, секакви двосмисленост при толкувањето на функционалноста или податоците на веб сервисот се елиминирани. Целта при анотирање на веб сервисите е да се овозможи нивно автоматско лоцирање и композиција. За да се разбере кои делови од веб сервисот треба да се анотираат, важно е да се разбере улогата на семантиката во животен циклус на сервисот и нејзиното користење во рамките на веб сервисот.



Слика 3-5: Семантика во животниот циклус на веб сервисот

Додека лоцираме веб сервис или правиме композиција од веб сервиси, корисникот ги опишува барањата во вид на функционалност, односно операции од веб сервисот, и податоците кои се користат, т.е. влезовите и излезите на сервисот. Дополнителните спецификации вклучуваат предуслови и ефекти врз операциите.

Предусловите се барања кои мора да се исполнети пред да се повика операција на веб сервисот, додека пак ефектите се резултати од повиканата операција. Семантичката анотација е поврзана со влезовите, излезите, предусловите и ефектите од операцискиот елемент во еден веб сервис.

Додавањето на семантика значително го збогатува животен циклус на еден веб сервис (слика 3-5). Развивачите на веб сервиси може да користат семантичка анотација за да ги објаснат можностите на нивните веб сервиси (чекор 1 од слика 3-5). Откако овие веб сервиси ќе се објават во UDDI регистарот (чекор 2 од слика 3-5), корисникот на веб сервисот може да ги формулира неговите барања во семантичкиот образец на сервисот (чекор 3 од слика 3-5) за да тој ги открие и компонира соодветните веб сервиси. Семантичкиот сервис е апстрактен опис за сервисот, каде текот на контрола е креиран рачно и бараната функционалност е опишана користејќи термини од онтологијата. Техниките за резонирање може да се искористат за да се споредат барањата во сервис образецот со можностите на веб сервисот достапни во UDDI регистарот. За време на композицијата, функционалниот аспект од анотацијата може да се користи за да се креираат корисни композиции од сервиси.

3.3.2. Типови на семантички опис кај веб сервисите

На Табела 3-5 се прикажани четирите типови на семантички опис кои можат да се сретнат кај веб сервисите: податочен, функционален, нефункционален и извршен семантички опис. Дадено е и објаснување како тие се поврзани со различните чекори прикажани на слика 3-5.

Табела 3-5: Четири типови на семантички описи кај веб сервисите

Семантички опис	Опис	Користење
Податочен	Формална дефиниција за податоците од влезните и излезните пораки од веб сервисот.	Откривање на сервиси и интероперабилност помеѓу сервисите.
Функционален	Формална дефиниција за можностите кои ги нуди веб сервисот.	Откривање и композиција на веб сервиси.
Нефункционален	Формална дефиниција за квантитативните и неквантитативните ограничувања, како што е QoS (Quality of service), барања како што се минимална цена и полиси како енкрипција на пораки.	Откривање, композиција и интероперабилност помеѓу сервисите.

Извршен	Формална дефиниција за извршувањето или текот на сервисите во процесот или на операциите во сервисот.	Процес на верификација и справување со грешки.

3.3.3. Креирање на семантички веб сервиси

Со зголемувањето на бројот на веб сервиси, нивната полуавтоматската анотација станува многу важна. Основната идеја при додавањето на семантички опис на елементите на веб сервисите, е наоѓањето најсоодветен семантички концепт за секој од WSDL елементите во онтологијата.

Поврзувањето на WSDL документ со OWL онтологија го воведува проблемот на поврзување на два хетерогени модели, секој со своите сопствени можности и ограничувања. Проблемот со поврзување на две различни онтологии го креира и проблемот на податочна интероперабилност во контекст на податочна шема. Во литературата најчесто зборовите *поврзување* и *мапирање* имаат исто значење. Тука зборот поврзување се однесува на процес на пронаоѓање на семантички совпаѓања помеѓу елементите од две шеми, додека пак зборот мапирање се однесува на физичката репрезентација на поврзувањата помеѓу елементите и правилата за трансформирање на елементите од една шема во друга.

3.3.4. Поврзување

Бидејќи проблемот со поврзување на шеми траел долго време, во заедницата за бази на податоци, во текот на ’80тите и раните ’90ти години, ја увиделе потребата од податочна интероперабилност, мапирање, спојување, трансформирање на шеми, семантичка хетерогеност и користење на онтологии и логички описи за семантичка интеграција.

Меѓутоа, поголем дел од претходните работи во заедницата за бази на податоци беа фокусирани на поврзување на хомогени модели. Секоја разлика во репрезентацијата на шемата беше решена со нормализирање на различните шеми пред истите тие да се поврзат. Во нашиов случај поврзувањето на WSDL со OWL е вистинско соочување со два различни модели. Трансформирањето на помалку експресивен модел (XML базиран WSDL) во поекспресивен модел (OWL) вообичаено бара човечка помош за да се додадат дополнителни семантички описи.

Моментално, во оваа област фокусот е на развивањето на генеричка инфраструктура која ќе изврши апстракција на операциите на моделот (шемата) и мапирањето помеѓу моделите како операции кои се генерички и независни од податочниот модел и конкретната апликацијата. Во областа на веб сервисите адресирањето на разликите помеѓу WSDL и OWL е извршено со нормализирање

на двете репрезентации во заеднички формат на граф. Резултатот од поврзувањето се семантички совпаѓања кои понатаму се претставуваат како анотации.

Полуавтоматски систем за анотирање на елементите од веб сервисот ќе биде возможен само доколку е можно да се поврзе WSDL документ со една или повеќе шеми од модел области, односно онтологии, и да се вратат семантичките совпаѓања со степен на сигурност при поврзувањето. Во случаи на двосмисленост, потребно е вклучување на корисник кој може да помогне да се разрешат двосмислените поврзувања продуцирани од системот. Иако потребата за поврзување на шеми е неопходна за генерирање на семантички анотации, потребата од мапирање заслужува поголемо внимание.

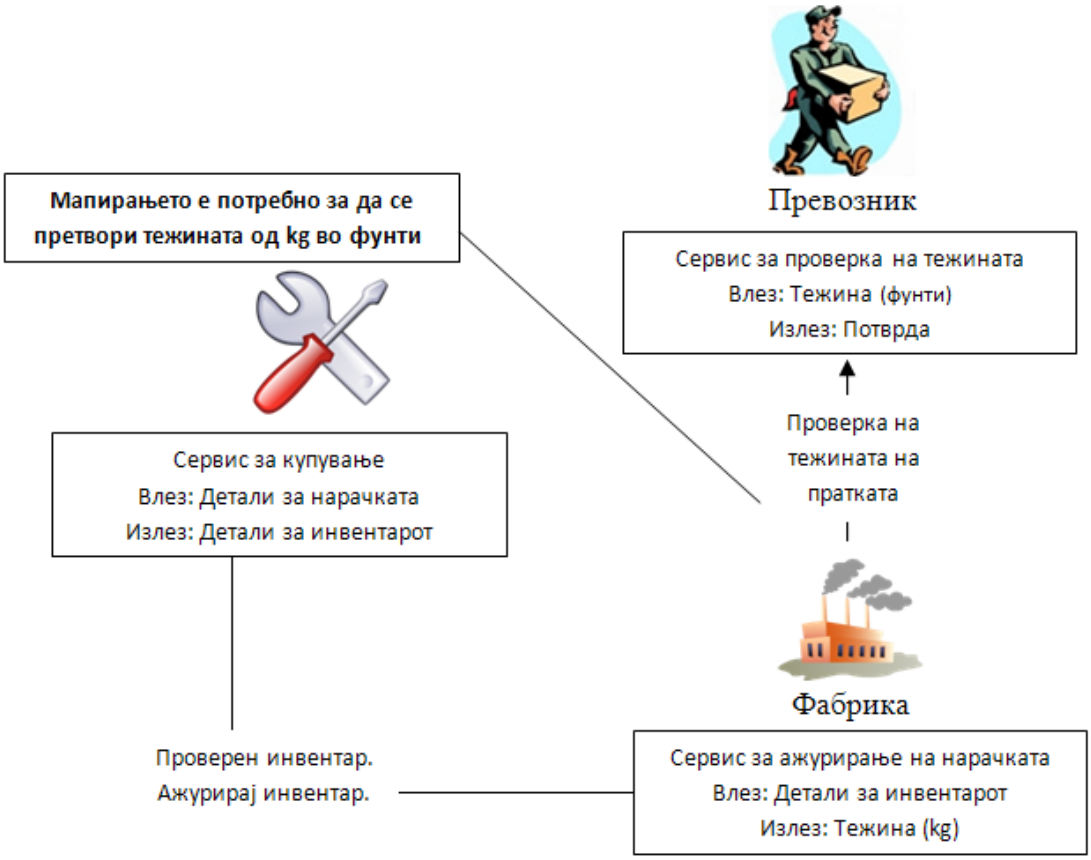
3.3.5. Мапирање

Како што видовме, семантичката анотација на елементите од веб сервисите го олеснува откривањето и композицијата на сервисите. Во контекст на композиција на сервисите, редоследот на сервисите обезбедува семантичка компатибилност помеѓу нивните влезови и излези, но не мора да значи дека обезбедува и интероперабилност.

На пример, веб сервисот од слика 3-6 извршува значаен процес во смисла на семантиката на неговата функционалност и податоците кои се разменуваат, но форматот на пораките кои ги разменуваат е некомпатибилен. Излезот од сервисот за ажурирање на инвентарот и влезот во сервисот за проверка на тежина и неговите елементи, се семантички компатибилни, но различни во нивните формати (килограми наспроти фунти), што ја прави композицијата бескорисна. Мапирањето помеѓу два елементи кои конвертираат еден формат на пораки во друг (од тежина во килограми, во тежина во фунти) е потребно за да се направи композицијата успешна.

Мапирањето како тоа на слика 3-6 е едноставно и може да биде автоматизирано. Меѓутоа, покомплексните мапирања се тешки за автоматизирање без човечка интервенција. Во Табела 3-6 се прикажани некои шема и податочни конфликти кои го прават мапирањето вистински предизвик.

Креирањето на мапирање помеѓу секои два веб сервиси кои треба да комуницираат, не е вистинско решение. Секогаш кога се креира нов сервис, целата интероперабилност која постои помеѓу веб сервисите треба да креира мапирања со новите пораки од новиот веб сервис, со цел да се зачува хетерогеноста. Како алтернатива, може да се креираат мапирања помеѓу елементите на веб сервисот и онтолошкиот концепт со кој елементот на веб сервисот е семантички поврзан. На тој начин, онтологиите стануваат алатка преку која веб сервисите го решаваат проблемот со структурна и синтаксична хетерогеност помеѓу пораките.



Слика 3-6: Илустрација на потребата од мапирање помеѓу пораките од веб сервисите

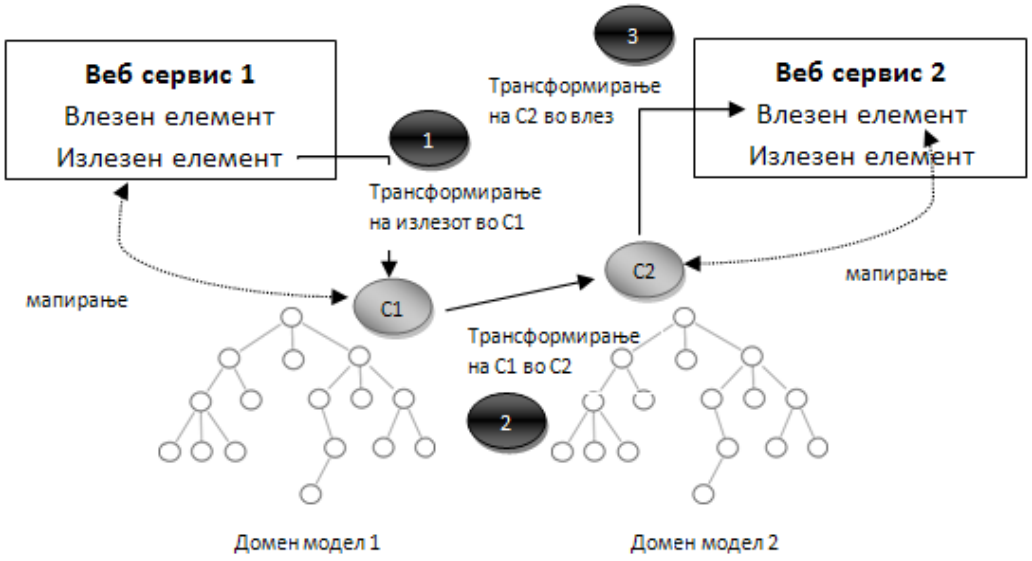
Табела 3-6: Можни шема/податочни конфликти помеѓу xml влезни/излезни пораки

Хетерогеност / Конфликти	**Примери – спротивставените елементи се обележани со боја**	
Некомпатибилност на областа – разликата во атрибутите се јавува поради користењето на различни описи за атрибути со исто значење (семантика)		
Конфликти во именувањето: два атрибути кои се семантички исти, може да имаат различни имиња (синоними); два атрибути кои не се семантички поврзани, може да имаат исти имиња (хомоними).	*Веб сервис 1* Student(#id, name) Student(#id, name)	*Веб сервис 2* Student(SSN,name) Kniga(#id, name)
Конфликти со претставувањето на податоците: два атрибути кои се семантички исти, може да имаат	*Веб сервис 1* Student(#id, name) #id е 4 цифрен број	*Веб сервис 2* Student(#id, name) #id е 9 цифрен број

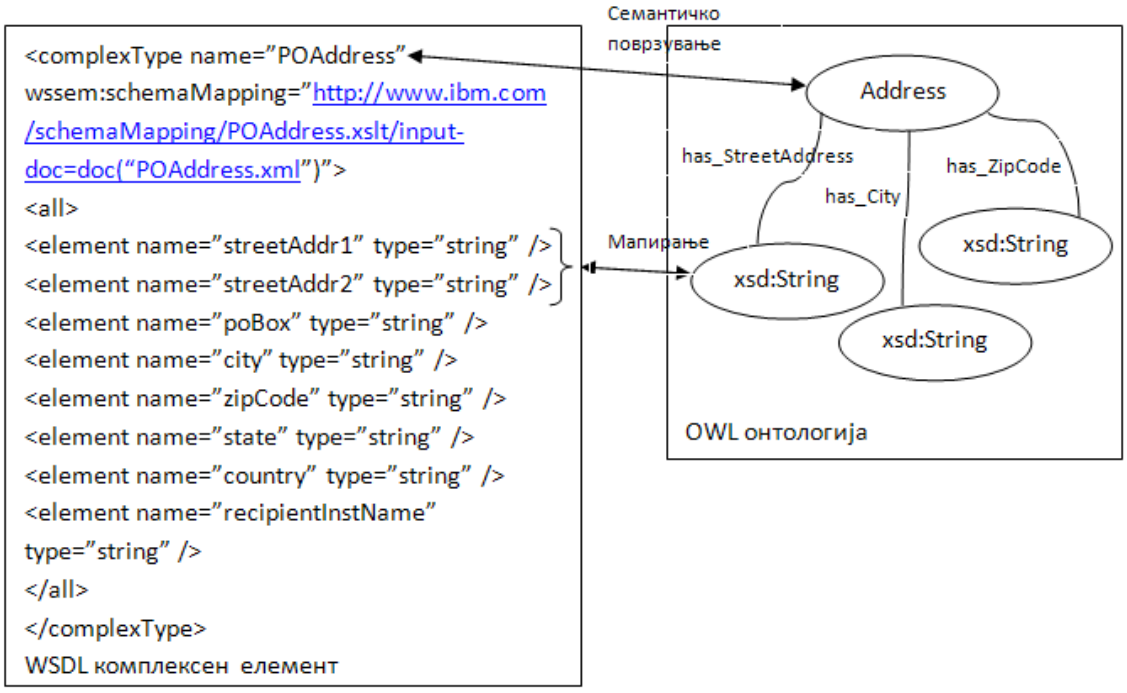
различни податочни типови или репрезентации.	
Конфликти со податочен опсег: два атрибути кои се семантички исти, може да се претстават со различна прецизност.	*Веб сервис 1* Marks 1-100 *Веб сервис 2* Grades A-F
Дефиниција на ентитети – разликата помеѓу ентитетите е последица на користењето на различни описи за семантички исти ентитети	
Конфликти во именувањето: Семантички исти ентитети може да имаат различни имиња (синоними). Два ентитети кои не се семантички поврзани, може да имаат исти имиња (хомоними).	*Веб сервис 1* Employee (#id, name) *Веб сервис 2* Worker(#id, name) *Веб сервис 1* Ticket(#idTicket, MovieName) *Веб сервис 2* Ticket(#FlightNo, Arr. Airport, Dep. Airport)
Конфликти со изоморфни шеми: Семантички исти ентитети може да имаат различен број на атрибути.	*Веб сервис 1* Person(name, address, WorkPhone, HomePhone) *Веб сервис 2* Person(name, address, phone)
Некомпатибилност на ниво на апстракција – разликите помеѓу ентитетите и атрибутите се последица на тоа што семантички исти ентитети и атрибути се претставено со различни нивоа на апстракција	
Конфликти при генерализација: Семантички исти ентитети се претставени со различни нивоа на генерализација.	*Веб сервис 1* GradStudent(#id, name, Major) *Веб сервис 2* Student(#id, name, Major, Type)
Конфликти со агрегација: Семантички исти ентитети се претставени со различни нивоа на агрегација.	*Веб сервис 1* Professor(#id, name, Dept) *Веб сервис 2* Faculty(#id, ProfID, Dept)
Конфликти со атрибути и ентитети: Семантички исти ентитети моделирани како атрибути во еден сервис и како ентитети во друг сервис.	*Веб сервис 1* Course(#id, name, Semester) *Веб сервис 2* Dept(Course, Semester, …, …)

Откако ќе се дефинира и претстави мапирањето, на слика 3-7 ни е прикажано како два веб сервиса ќе комуницираат користејќи ги овие мапирања од онтологиските концепти. Чекорите 1, 2 и 3 ја олеснуваат размената на пораки помеѓу двата веб сервиса. Во чекор 1, излезната порака од првиот веб сервис се трансформира во OWL концепт во кој е мапиран; OWL концептот потоа се трансформира во влезна порака за вториот веб сервис. Постои можност двата веб сервиса да не се анотирани или мапирани со иста онтологија. Во тој случај,

потребно е да се дефинира мапирање помеѓу онтологиските концепти (чекор 2). Сé додека мапирањето е од елемент во веб сервисот во онтолошки концепт, генерирањето на инверзно мапирањето (чекор 3) не може секогаш да биде автоматизирано и бара корисничка интервенција.



Слика 3-7: Онтологии како механизам за комуникација помеѓу веб сервиси



Слика 3-8: Мапирање со помош на XQuery и XSLT

Покрај ваквиот пристап, во процесот на автоматизација на мапирањето постојат и пристапи кои се користа во контекст на веб сервисите. Еден таков пристап е мапирањето со користење на XQuery и XSLT. И двата јазика работат со XPATH за трансформирање на xml објектите од еден во друг формат. На слика 3-8 е даден еден пример за мапирање помеѓу WSDL пораките и OWL концептите, со користење на XQuery и XSLT.

Мапирање со помош на XQuery:

```
for $a in doc("POAddress.xml")/POAddress
  return
    <POOntology:Address rdf:ID="Address1">
      <POOntology:has_StreetAddress rdf:datatype="xs:string">
        {fn:concat($a/streetAddr1, " ", $a/streetAddr2)}
      </POOntology:has_StreetAddress>
      <POOntology:has_City rdf:datatype="xs:string">
        {fn:string($a/city)}
      </POOntology:has_City>
      <POOntology:has_State rdf:datatype="xs:string">
        {fn:string($a/state)}
      </POOntology:has_State>
      <POOntology:has_Country rdf:datatype="xs:string">
        {fn:string($a/ country)}
      </POOntology:has_Country>
      <POOntology:has_ZipCode rdf:datatype="xs:string">
        {fn:string($a/zipCode)}
      </POOntology:has_ZipCode>
    </POOntology:Address>
```

Мапирање со помош на XSLT:

```
<xsl:template match="/">
  <POOntology:Address rdf:ID="Address1">
    <POOntology:has_StreetAddress rdf:datatype="xs:string">
      <xsl:value-of select="concat(POAddress/streetAddr1,
POAddress/streetAddr2) />
    </POOntology:has_StreetAddress>
    <POOntology:has_City rdf:datatype="xs:string">
      <xsl:value-of select="POAddress/city" />
    </POOntology:has_City>
    <POOntology:has_State rdf:datatype="xs:string">
      <xsl:value-of select="POAddress/state)" />
    </POOntology:has_State>
    <POOntology:has_Country rdf:datatype="xs:string">
      <xsl:value-of select="POAddress/ country" />
    </POOntology:has_Country>
    <POOntology:has_ZipCode rdf:datatype="xs:string">
       <xsl:value-of select="POAddress/zipCode" />
    </POOntology:has_ZipCode>
   </POOntology:Address>
```

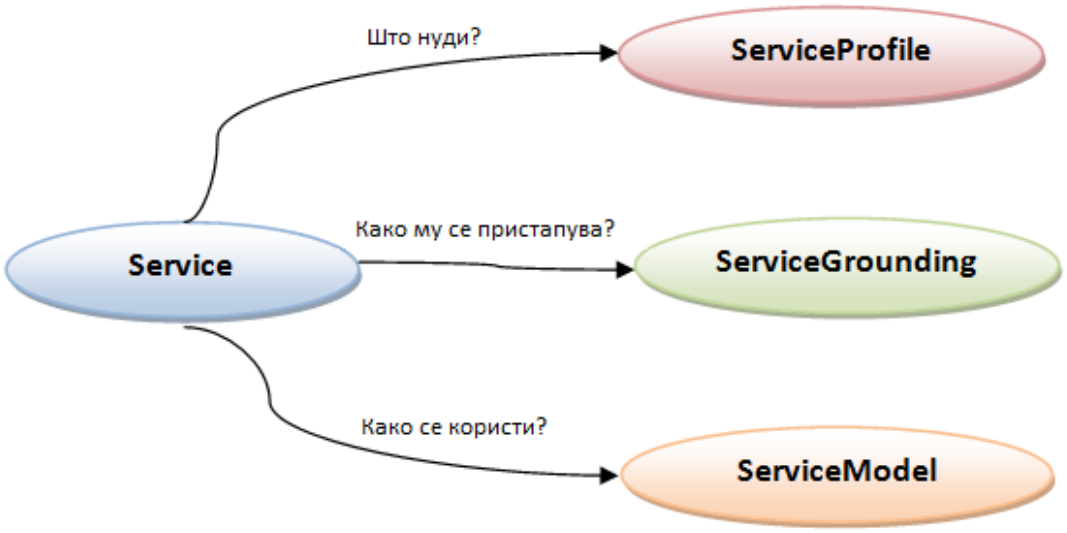
3.4. Рамки за семантичка анотација на веб сервиси

Најистакнати рамки за семантичка анотација на веб сервиси се OWL-S [44], WSMO [45], WSDL-S [46] и SAWSDL [47]. Додека WSMO и OWL-S дефинираат свои сопствени семантички модели за веб сервиси, WSDL-S и SAWSDL работат со задржување на информациите кои се веќе претставени во WSDL описот на веб сервисите.

3.4.1. OWL-S

Генерално гледано, постојат два типа на сервиси - атомични и композитни. Атомични сервиси се сервиси кои се состојат од само еден веб сервиси, додека пак композитни се оние сервиси кои се состојат од повеќе веб сервиси, кои заедно работат да извршат некоја задача. OWL–S, Semantic Markup for Web Services, обезбедува поддршка и за двете категории на сервиси [44].

OWL-S, како маркирачки јазик за веб сервиси, помага во автоматизацијата на дел од процесите кои се среќаваат при работа со веб сервиси, како што е откривање, извршување, интероперабилност, композиција и мониторирање на веб сервиси.



Слика 3-9: OWL-S онтологија

OWL-S користи своја онтологија од повисоко ниво за опишување на веб сервисите. Онтологијата се состои од четири основни класи. Класата Service

служи за да се поврзат останатите три класи: класата Service Profile, која го претставува делот од сервисот кој треба да даде одговор на прашањето „што нуди сервисот за потенцијалните корисници?“, класата Service Model, која го опфаќа делот од сервисот кој треба да одговори на прашањето: „како се користи веб сервисот?“ и класата Service Grounding, која го дава одговорот на прашањето „како му се пристапува на веб сервисот?“ (слика 3-9).

Секоја објавен веб сервис има инстанца од 'Service' класата. Својствата на Service класата, 'presents', 'describedBy' и 'supports' покажуваат кон соодветните 'ServiceProfile', 'ServiceModel' и 'ServiceGrounding' класи. Секоја инстанца од Service класата ќе има ServiceProfile, ServiceModel и ServiceGrounding описи. ServiceProfile обезбедува информации потребни за корисникот да може да го открие сервисот, додека пак ServiceModel и ServiceGrounding обезбедуваат информации за како корисникот да го користи веб сервисот.

3.4.2. WSMO

Web Service Modeling Ontology, или скратено WSMO, е концептуален модел за развивање на семантички веб сервиси [45]. Се состои од онтологија од елементи за семантички опис на веб сервиси, опишани во формален јазик за опишување, наречен Web Services Modeling Language (WSML) [48] и исто така околина за извршување на сервисите, наречена Web Service Execution Environment (WSMX) [49]. WSMO е базиран на Web Service Modeling Framework (WSMF) [50] рамката за развој на семантички веб сервиси.

Во WSMO, онтологиите обезбедуваат терминологија која се користи од останатите WSMO елементи за опишување на релевантните аспекти од областа. Преку Goal елементите се дефинираат барањата на корисниците кои треба да бидат извршени од страна на веб сервисите, а Mediator елементите ги решаваат проблемите со интероперабилност помеѓу различните WSMO елементи. WSMO се состои од три нивоа на посредување – DataLevel ниво, во рамките на кое се решаваат проблемите со интероперабилност помеѓу различни податочни извори, ProtocolLevel ниво, во рамките на кое се решаваат проблемите со интероперабилност помеѓу различни комуникациски протоколи и ProcessLevel ниво, во рамките на кое се решаваат проблемите со интероперабилност помеѓу различните бизнис процеси.

И WSMO и OWL-S користат ист пристап за градење на семантика за веб сервиси, користејќи свои сопствени онтологии за опис на елементите и поимите кои се среќаваат при работата со веб сервиси. OWL-S е базиран на OWL и поставува правила користејќи го Semantic Web Rule Language (SWRL) [51]. WSMO има своја сопствена фамилија на јазици, кои се базирани на описна логика и логичко програмирање [45].

3.4.3. WSDL-S

WSDL-S (Web Service Description Language Semantics) е развиен од страна на IBM и универзитетот во Џорџија [46]. Неговата основна цел е додавање семантика во рамките на стандардните WSDL документи. Главна предност на WSDL-S, во однос на останатите рамки за семантичка анотација на веб сервиси, е токму повторното искористување на WSDL описот.

Употребата на WSDL-S зависи од конкретните онтологии кои се користат при анотацијата. Со WSDL-S, семантичкиот опис се додава на различни делови од WSDL документот, со користење на конкретни онтологии од доменот за кој се наменети самите веб сервиси. Друга предност на WSDL-S е тоа што не го ограничува изборот на јазикот во кој се конструирани онтологиите кои се користат. Но, од друга страна, тоа значи дека софтверскиот агент кој е одговорен за процесирање на WSDL-S документите треба да биде попаметен, т.е. треба да биде способен за препознавање и разбирање на неколку различни јазици за опис на онтологии.

Важно е да се напомене дека WSDL-S главно е фокусиран на динамичкото лоцирање на веб сервисите. Дополнително, за семантичко анотирање на влезовите и излезите на сервисот, додава концепти на предуслови и ефекти. Меѓутоа, не обезбедува доволно семантички информации за автоматско повикување, композиција и мониторирање на даден сервис.

Основните конструктори кои се предложени од WSDL-S се прикажани во Табела 3-7. За да видиме како се изведува семантичката анотација на веб сервис, како пример ќе го земеме претходно разгледуваниот getMegaPixel веб сервис. Првиот чекор е анотација на operation елементот. Тоа се правени со додавање на атрибут кој покажува кон некој концепт во дадена онтологија.

Табела 3-7: WSDL-S конструктори за анотација

Конструктор	Значење
wssem:modelReference	Проширен атрибут кој дозволува еден-на-еден асоцијации на WSDL влезните и излезните елементи со концепти од специфична област.
wssem:schemaMapping	Проширен атрибут кој дозволува повеќе-на-повеќе асоцијации на WSDL влезните и излезни елементи со концепти од семантичкиот модел – најчесто поврзани со XML шема комплексни типови.
wssem:precondition и wssem:effect	Два елементи кои се деца на operation елементот и ја опишуваат неговата семантиката со специфицирање на предуслови и

	ефекти; се користи за откривање на сервиси.
wssem:category	Проширен атрибут кој се користи во portType елементот (во најновата верзија на WSDL portType елементот е заменет со interface елемент). Тој се состои од информации за категоризирање на сервисот кои може да се користат кога го објавуваме сервисот во регистар каков што е UDDI.

Во овој пример тоа е Cameral.owl онтологијата, во која не е дефинирана некоја операција, но за да се види како се прави анотацијата, претпоставуваме дека е дефинирана операцијата GetPixelOperation:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
      xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
      xmlns:s="http://www.w3.org/2001/XMLSchema"
      xmlns:s0="http://tempuri.org/"
      xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
      xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
      xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WS-
Semantics"

xmlns:camera="http://www.yuchen.net/photography/Cameral.owl"
      targetNamespace="http://tempuri.org/"
      xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    ...
  </wsdl:types>
  <wsdl:message name="getMegaPixelSoapIn">
      <wsdl:part name="parameters" element="s0:getMegaPixel" />
  </wsdl:message>
  <wsdl:message name="getMegaPixelSoapOut">
       <wsdl:part name="parameters"
element="s0:getMegaPixelResponse" />
  </wsdl:message>
  <wsdl:portType name="Service1Soap">
    <wsdl:operation name="getMegaPixel"
          wssem:modelReference="camera:GetPixelOperation">
            <wsdl:input message="s0:getMegaPixelSoapIn" />
            <wsdl:output message="s0:getMegaPixelSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Service1Soap" type="s0:Service1Soap">
    ...
  </wsdl:binding>
  <wsdl:service name="Service1">
    ...
  </wsdl:service>
</wsdl:definitions>
```

Можеме да видиме дека во примерот се додадени соодветни именски простори. Исто така, операцијата е анотирана со даден концепт од онтологијата.

Всушност, ова е и добро место за додавање предуслови и ефекти. Во разгледуваниот пример, нема предуслови и ефекти, но истите можат да се додадат, како илустрација на можностите на WSDL-S:

```
<wsdl:portType name="Service1Soap">
  <wsdl:operation name="getMegaPixel"
      wssem:modelReference="camera:GetPxielOperation">
    <wssem:precondition name="someNameForCondition"
                wssem:modelReference="camera:somePreconditionCo
          ncept">
    <wssem:effect name="someNameForEffect"

wssem:modelReference="camera:someEffectConcept">
    <wsdl:input message="s0:getMegaPixelSoapIn" />
    <wsdl:output message="s0:getMegaPixelSoapOut" />
  </wsdl:operation>
</wsdl:portType>
```

Следниот чекор е анотација на влезовите и излезите. Овие анотации може да се додадат во XML шема дефинициите на елементите:

```
<?xml version="1.0" encoding="utf-8"?>
<wsdl:definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:s="http://www.w3.org/2001/XMLSchema"
        xmlns:s0="http://tempuri.org/"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
        xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
        xmlns:wssem="http://www.ibm.com/xmlns/WebServices/WS-
Semantics"

xmlns:camera="http://www.yuchen.net/photography/Cameral.owl"
        targetNamespace="http://tempuri.org/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  <wsdl:types>
    <s:schema                    elementFormDefault="qualified"
targetNamespace="http://tempuri.org/">
      <s:element name="getMegaPixel">
        <s:complexType>
          <s:sequence>
            <s:element name="cameraModel"
                wssem:modelReference="camera:model"
type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="getMegaPixelResponse">
        <s:complexType>
          <s:sequence>
            <s:element name="getMegaPixelResult"
                wssem:modelReference="camera:pixel"
type="s:double"/>
          </s:sequence>
        </s:complexType>
      </s:element>
```

```
    </s:schema>
  </wsdl:types>
  <wsdl:message name="getMegaPixelSoapIn">
    <wsdl:part name="parameters" element="s0:getMegaPixel" />
  </wsdl:message>
  <wsdl:message name="getMegaPixelSoapOut">
    <wsdl:part                                    name="parameters"
element="s0:getMegaPixelResponse" />
  </wsdl:message>
  <wsdl:portType name="Service1Soap">
    <wsdl:operation name="getMegaPixel"
          wssem:modelReference="camera:GetPxielOperation">
      <wssem:precondition name="someNameForCondition"

wssem:modelReference="camera:somePreconditionConcept">
        <wssem:effect name="someNameForEffect"
              wssem:modelReference="camera:someEffectConcept">
      <wsdl:input message="s0:getMegaPixelSoapIn" />
      <wsdl:output message="s0:getMegaPixelSoapOut" />
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="Service1Soap" type="s0:Service1Soap">
    ...
  </wsdl:binding>
  <wsdl:service name="Service1">
    ...
  </wsdl:service>
</wsdl:definitions>
```

Меѓутоа, тука постојат повеќе опции. Прво, ако WSDL type елементот користи едноставен тип (како што е string или integer), тој може директно да се мапира со концепт од онтологијата, како што е прикажано на претходниот пример. Доколку, пак, елементот е од комплексен тип, може или да се мапира целиот елемент со даден концепт од онтологијата или да биде посебно анотиран секој елемент од комплексниот тип:

- Мапирање на целиот комплексен тип:

```
<complexType wssem:modelReference="camera:someConcept">
  <all>
    <element name="leaf-node-1" type="xsd:integer"/>
    <element name="leaf-node-2" type="xsd:string"/>
    <element name="leaf-node-3" type="xsd:integer"/>
  </all>
</complexType>
```

- Мапирање на секој елемент од комплексниот тип:

```
<complexType>
  <all>
      <element name="leaf-node-1" type="xsd:integer"
            wssem:modelReference="camera:someConcept1">
      <element name="leaf-node-2" type="xsd:string"
            wssem:modelReference="camera:someConcept2">
      <element name="leaf-node-3" type="xsd:integer"
            wssem:modelReference="camera:someConcept2">
  </all>
```

```
</complexType>
```

На крај, можеме да се додаде и семантика за категоризирање на сервисот. Основна цел на додавањето на категоризацијата е лесно откривање на сервисот:

```
<wsdl:portType name="Service1Soap">
  <wssem:category name="On-Line Information Service"
            taxonomyURI="http://www.naics.com/"
        taxonomyCode="514191" />
  <wsdl:operation name="getMegaPixel">
    ...
  </wsdl:operation>
</wsdl:portType>
```

Предноста на додавање на семантика во WSDL е повторното искористување на достапните стандарди и алатки. Попрецизно кажано, алатките кои се развиени врз основа на стандарден WSDL опис, продолжат да се користат.

За да го решиме проблемот со автоматско откривање, јасно е дека сервисот прво мора да биде објавен во некој регистар. Добар избор е UDDI регистарот.

Организацијата OASIS (Organization for the Advancement of Structured Information Standards) има објавено шеми за мапирање од WSDL-S во UDDI податочни структури, но и понатаму семантиката треба посебно да се додаде во UDDI регистарот [52]. Според тоа, сé уште не постои унифициран метод за објавување на WSDL-S документ во рамките на UDDI, ниту пак унифициран метод за негово откривање.

3.4.4. SAWSDL

Semantic Annotations for WSDL and XML Schema (SAWSDL) е препорака на W3C конзорциумот, од 2007 година, што значи дека самиот конзорциум го фаворизира користењето на оваа рамка за семантичка анотација на веб сервиси. SAWSDL не специфицира јазик за претставување на семантички модели. Тој нуди механизам со кој концепти од семантичките модели, вообичаено дефинирани надвор од WSDL документот, се референцираат со WSDL и XML шема компоненти [47].

Оваа рамка дефинира начин како да се додаде семантичка анотација на различни делови од WSDL документот, како што се излезните и влезните пораки, интерфејсите и операциите. На пример, спецификацијата дефинира начин за анотирање на WSDL интерфејси и операции категоризирајќи ги информации кои може да се користат за објавување на веб сервисот во регистар. Анотацијата на шема типовите може да се користи за време на откривање и композиција на веб сервисите. За да се изврши семантичка анотација, SAWSDL дефинира дополнителни атрибути кои може да се додадат на WSDL и XML шема

елементите. Семантичката анотација референцира концепт од онтологијата и е независна од јазикот во кој е напишана онтологијата.

SAWSDL е проширување на WSDL, реализирано со користење на проширени елементи. Притоа, постојат два основни типови на анотација, model reference и schema mapping. Model reference анотацијата, се користи за поврзување на интерфејси, операции, влез, излез и xml шема елементи и атрибути со семантички концепти. Model reference анотацијата се користи од METEOR-S рамката за лесно откривање на веб сервиси и динамичко конфигурирање на веб процеси [55]. Schema mapping анотацијата, исто така се користи од METEOR-S рамката и за решавање на проблемите со хетерогени податоци, особено при трансформирање на една репрезентација на податоци во друга, за да може да се користи од друг веб сервис.

4. Рамка за проширување на онтологии со функции

Јазикот за веб онтологии (Web Ontology Language, OWL), како дел од архитектурата на Семантичкиот Веб, обезбедува механизам за дефинирање на структурата на податоците од Семантичкиот Веб (репрезентирани во RDF) преку онтологии [9]. Преку стриктна дефиниција на структурата на податоците, може да се контролира содржината на базата на знаење, изградена од податоци претставени во RDF, односно може да се валидираат податоците и врз основа на базата на знаење да се расудува, односно да се извлекуваат ново, неексплицитно знаење. Со тоа се овозможува проширување на базата на знаење со нови факти, односно нови RDF тројки, преку користење на постоечките информации и дефинициите за елементите од доменот, зададени преку OWL онтологија.

Сепак, иако OWL има за цел да ја дефинира структурата и релациите помеѓу ентитетите во даден домен, а со тоа и ограничувањата кои мора да ги почитуваат тие, неговата експресивност е ограничена. Имено, потребата за дефинирање на одредени ограничувања и функционалности кои се вообичаени за реалните апликации за кои се користи процедурално програмирање, не може да биде задоволена со користење само на онтологии. Иако според архитектурата на Семантичкиот Веб дел од оваа функција треба да ја преземат јазиците за дефинирање правила, како што е форматот за размена на правила (Rule Interchange Format - RIF), нивната стандардизација и имплементација се сé уште во тек [12].

Дури и кога RIF ќе стане стандард и ќе почнат да се користат правилата во сите системи изградени со технологиите на Семантичкиот Веб, повторно остануваат нерешени дел од проблемите со кои се среќаваме поради декларативниот пристап на RDF, RDFS и OWL. Еден од тие проблеми е добивањето на нестатички информации, како и добивањето информации во реално време.

Проблем со кој се среќаваат апликациите, односно системите изградени со технологиите на Семантичкиот Веб, е проблемот со добивање на информации во реално време. Тоа може да се увиди од следното корисничко сценарио: да претпоставиме дека сакаме да креираме апликација која прикажува корисни информации за одредени географски предели. Апликацијата се базира на технологиите на Семантичкиот Веб. Сакаме нашата апликација, покрај техничките информации за географскиот предел, да прикажува и податок за температурата на таа локација во реално време. Потпирајќи се исклучиво на технологиите на Семантичкиот Веб, тешко би го извеле тоа, повторно поради потребата од многу честа и многу голема промена на информации во базата на знаење. Ова е последица на тоа што податоците кои се читаат во реално време, се *нестатички*.

Исто така, постои и проблемот со добивање на нестатички информации. Тоа може да се увиди со следното корисничко сценарио: да претпоставиме дека сакаме да креираме апликација за управување со состаноци, која целосно се потпира на технологиите на Семантичкиот Веб и притоа користи само декларативно програмирање. Едно сценарио кое треба да се имплементира во таков систем е креирањето на нов состанок. Покрај внесување на учесниците во состанокот, одбирање на датум, почеток и времетраење на состанокот, корисниците на ваквата апликација би требало да одберат и просторија во која сакаат да биде одржан состанокот. Она што тука треба да внимаваме, е кои простории ќе му ги понудиме на корисникот да може да ги одбере. Логично е да му се понудат оние простории кои на тој датум, во таа временска рамка кога се закажува состанокот, се слободни. Со користење само на постоечките технологии на Семантичкиот Веб, па дури и со користењето на правила, решението на ваквиот проблем е нетривијално. Ова е последица на фактот што информацијата за тоа дали одредена просторија, на даден датум, во дадено време е слободна или зафатена, е *нестатичка информација*.

Идејата претставена во овој магистерски труд опфаќа дефинирање на рамка за проширување на онтологиите, со цел да се постигне поголема функционална моќ на онтологиите. На тој начин, онтологиите ќе можат да се искористат за развој на апликации во кои тие нема да го дефинираат само податочниот модел и базата на знаење, туку ќе внесат и одредена функционалност во самите апликации. Така ќе се избегне потребата од пишување дополнителен код во некој програмски јазик

за дефинирање на функционалностите, динамиката и ограничувањата на семантичките апликации; самите технологии на Семантичкиот Веб ќе бидат доволни за дефинирање на целосно функционална семантичка апликација.

На овој начин се овозможува технологиите на Семантичкиот Веб да покријат поголем аспект од процесот на развој на апликации, а нивната улога од технологии погодни за работа со податоци и знаење се проширува со функционалност, со што се избегнува потребата од дополнително програмирање во рамките на семантичките апликации. Со тоа се зголемуваат и можностите на самото декларативно програмирање, врз кое се потпираат технологиите на Семантичкиот Веб.

Проширувањето се постигнува со дефинирање на онтологија, која ја моделира рамката на проширувањето. Во онтологијата се дефинирани класи и релации со чија помош може на одреден ресурс, од одредена база на знаење, да му се придружи *нестатичка вредност*, вредност која се добива со повик на одреден веб сервис. Со тоа, преку оваа онтологија се обезбедува потребната поддршка за додавање на надворешни функционалности кон базата на знаење, со што суштински се менува карактерот на информациите кои постојат во базата на знаење, односно карактерот на информациите кои можат да се користат во рамките на апликациите кои целосно се потпираат на технологиите на Семантичкиот Веб.

4.1. Претходни истражувања во областа

Постојат повеќе идеи за проширување на онтологиите и на јазикот за веб онтологии (OWL), но повеќето од нив се однесуваат на проширување на типовите на податоци кои можат онтологиите да ги подржат [53], односно на ставањето на онтологиите во одреден контекст [54].

Проширувањето на форматите кои се користат за опис и претставување на податоци, како што се XML и XML-базираните RDF и OWL, со функции и сервиси, е идеја која е веќе разгледана и предложена во проширувањето наречено функционален XML (Functional XML - fXML) [56]. Спојувањето на податочните типови од Семантичкиот Веб, претставени преку концептите од онтологиите, семантичките веб сервиси, како дистрибуирани функционалности кои се дел од Семантичкиот Веб и функционалното проширување на XML, веќе се предложени како идеја која има за цел да ги надмине разликите помеѓу класичните програми и податоците, со што XML јазикот станува вистински *само-опишувачки* [57][58]. Ваквото проширување на XML е наречено семантички функционален XML - Semantic fXML. Идејата за Semantic fXML нема конкретна имплементација до денес.

Идејата за проширување на онтологиите, чија примарна цел е репрезентација на податоци, со функции е делумно инспирирана од овие идеи. Но, проширувањето на онтологиите како градбена единка на Семантичкиот Веб е од поголема корист за самите технологии на Семантичкиот Веб, за разлика од проширувањето на XML, кој не се користи директно во апликациите базирани на технологиите на Семантичкиот Веб.

Идејата, пак, за автоматска композиција на веб сервиси не е нова. Со појавата на сервисно ориентираната архитектура, интересот во оваа област многу порасна. Појавата на технологиите на Семантичкиот Веб уште повеќе ја актуелизираше оваа проблематика, затоа што овие технологии овозможуваат типовите на влезните и излезните параметри да бидат многу детално прецизирани. Користењето на онтологии за анотација на веб сервисите, овозможува крајниот корисник, или пак некој систем кој работи со веб сервиси, точно да знае кои се семантички опишаните влезни параметри на даден семантички анотиран веб сервис, а кој е семантички опишаниот излезен резултат. Ако се земе предвид дека пред појавата на технологиите на Семантичкиот Веб ваквото совпаѓање на влезните и излезните параметри се правело врз основа само на тогаш расположливите типови на податоци, како што се цел број, низа од знаци, итн., јасно е колку појавата на овие технологии ја подобрува и зајакнува целата идеја.

На полето на системи кои подржуваат работа со семантички веб сервиси веќе постојат интензивни истражувања, готови рамки за развој на апликации, практични прототип изведби на примери од проблематиката, како и два завршени проекти кои како финални продукти ги имаат претставено Internet Reasoning Service III [59][60][61] и WSMO Studio [62][63][64]. Овие два продукти нудат широк спектар на можности кои се главната предност на семантичките веб сервиси, како што се семантички опис на постоечки или нови веб сервиси, нивно објавување, повикување, полуавтоматска и автоматска композиција, итн.

Сепак, главен заклучок е дека со нивна помош не може да се креира интелигентен систем, кој би можел да прима одредени барања од крајните корисници или нивните семантички апликации, врз база на семантика автоматски да расудува за барањето, да пронаоѓа постоечки семантички веб сервиси од локален репозиториум или од Вебот, да направи нивна композиција, да ги повика и крајниот резултат да му го врати назад на испраќачот на барањето.

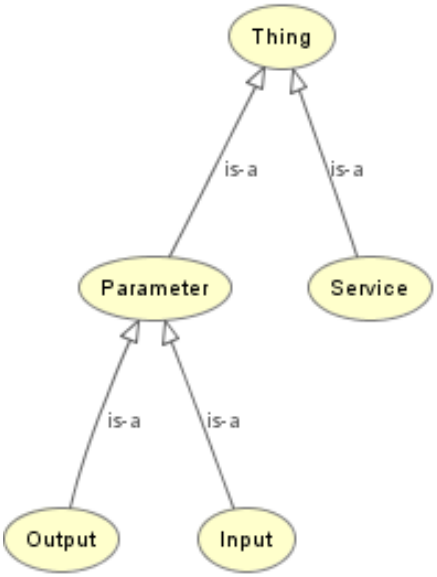
Токму предизвикот за креирање на еден ваков интелигентен и автоматизиран систем е главната мотивација зад идејата за проширување на онтологиите со функции и збогатување на семантичките апликации со нестатички податоци.

4.2. Онтологија за функционално проширување

Функционалното проширување, со кое во рамките на онтологиите се внесува механизам за пристап до нестатички податоци, односно до податоци кои често ја менуваат својата вредност и кои е непрактично да се чуваат во базата на знаење, го постигнуваме преку дефинирање на нова онтологија, наречена functionalOWL. Со новата онтологија, всушност, се моделира рамката на предложеното проширување.

Во онтологијата се дефинирани класи и релации со чија помош на одреден ресурс, од која било база на знаење, може да му се придружи *нестатичка вредност*, односно вредност која се добива како резултат од повик на одреден веб сервис.

Онтологијата functionalOWL ги содржи класите Service, Parameter, Input и Output (слика 4-1), како и релациите hasNonstaticValue, hasParameter, hasInput, hasOutput, hasValue, parameterType и hasName. Со помош на релацијата hasNonstaticValue се овозможува секој ресурс, од одредена база на знаење, да може својата вредност да ја добие со поврзување со одреден веб сервис, односно резултатот од веб сервисот да стане негова вредност. Опсегот на релацијата hasNonstaticValue се инстанци од класата Service, класа која ги моделира веб сервисите со кои можат да се поврзат податоците од базата на знаење.



Слика 4-1: Класи во functionalOWL онтологијата

Структурата на functionalOWL онтологијата е направена врз основа на искуството добиено од онтологиите кои ги користи OWL-S, па и самата OWL-S онтологија [44]. Сепак, потребата од functionalOWL и начинот на користење на functionalOWL не се тесно поврзани со OWL-S онтологијата. OWL-S онтологијата се користи за семантичка анотација на класични веб сервиси, со цел да им се даде недвосмислено, семантичко значење на елементите од кои е изграден еден класичен веб сервис. Од друга страна пак, functionalOWL онтологијата се користи за дефинирање на вредноста на даден ентитет од одредена база на знаење како *нестатичен*, односно дефинирање дека вредноста на ентитетот треба да се добие како резултат од некој семантички веб сервис (или композиција) која како влезни и излезни аргументи ги има аргументите кои по тип одговараат со оние дефинирани во инстанцата на класата Service, кои се наоѓаат во самата база на знаење.

Во продолжение поединечно ги презентираме класите и релациите од предложената functionalOWL онтологија.

4.2.1. Service класа

Класата Service од functionalOWL онтологијата служи за моделирање на веб сервисите со кои располага системот, односно веб сервисите кои можат да се постават како извор на вредности за ресурсите од дадена база на знаење. Оваа класа не се поврзува со конкретен веб сервис дефиниран во системот, туку само дефинира скелет на кандидат веб сервиси, преку дефинирањето на влезните параметри и излезниот параметар. Врз база на овие податоци, системот ќе биде во можност да пронајде кандидат сервиси кои можат да го вратат бараниот излезен параметар, ако на располагање ги имаат назначените влезни параметри. Пронаоѓањето на кандидат сервисите, селекцијата на еден конкретен веб сервис или на множество веб сервиси кои со правилна композиција можат да одговорат на барањето, како и повикувањето на сервисите е задача на системот, и ќе биде дискутирана во продолжение.

Service класата е дефинирана како опсег на својството hasNonstaticValue, преку кое останатите ресурси, без разлика на нивниот тип (види 4.1.4), можат како вредност да го искористат резултатот кој се добива со повик на веб сервис.

Дефиниција на Service класата:

```
<owl:Class rdf:ID="Service">
  <rdfs:comment>
The class for a Service.
  </rdfs:comment>
</owl:Class>
```

Оваа класа е дефинирана како домен на својствата hasName, hasParameter, како и на својствата изведени од hasParameter: hasInput и hasOutput. Со помош на овие четири својства, на инстанца од класата Service ѝ се доделува име, односно влезни и излезни параметри.

4.2.2. Parameter класа

Класата Parameter од functionalOWL онтологијата се користи како еден вид на апстрактна класа, која ги моделира параметрите кои се користат за веб сервисите од системот. Според тоа, не е предвидено да се креираат инстанци од оваа класа, затоа што нејзината функција е да послужи како класа од која ќе бидат изведени класите за влезните и излезните параметри, а со тоа ќе се запази хиерархијата во моделот кој functionalOWL онтологијата го претставува.

Оваа класа е дефинирана како опсег на својството hasParameter, преку кое на инстанците од класата Service може да им се додаде параметар.

Во исто време, оваа класа е дефинирана и како домен на својствата parameterType и hasValue. Преку нив, на инстанци од класата Parameter, како и на инстанци од Input и Output класите (кои се наследени од Parameter), може да им се постави тип и вредност.

Дефиниција на класата Parameter:

```
<owl:Class rdf:ID="Parameter">
 <rdfs:subClassOf>
   <owl:Restriction owl:minCardinality="1">
     <owl:onProperty rdf:resource="#parameterType" />
   </owl:Restriction>
 </rdfs:subClassOf>
</owl:Class>
```

Од дефиницијата на класата може да се види дека за неа има дефинирано едно ограничување: инстанците од класата мора да имаат дефинирано тип на параметар, односно да го имаат својството parameterType. Ова ограничување има за цел да не дозволи во базата на знаење да се инстанцира влезен или излезен параметар кој нема дефинирано тип. Типот на параметарот е неопходен за да системот може да побара кандидат сервиси за обезбедување на потребниот податок.

4.2.3. Input и Output класи

Класите Input и Output се класи наследени од класата Parameter. Со наследувањето, тие го наследуваат и ограничувањето кое дефинира дека мора да постои дефиниран тип на параметарот, а истовремено се и домени на својствата

parameterType и hasValue. Со помош на овие својства, инстанците на влезни излезни параметри можат да дефинираат тип и вредност.

Дефиниција на класите Input и Output:

```
<owl:Class rdf:ID="Input">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
</owl:Class>

<owl:Class rdf:ID="Output">
  <rdfs:subClassOf rdf:resource="#Parameter"/>
  </owl:Class>
```

Класата Input е опсег на својството hasInput, со кое на инстанца од класата Service ѝ се доделува влезен параметар.

Класата Output, пак, е опсег на својството hasOutput, со кое на инстанца од класата Service ѝ се доделува излезен параметар.

4.2.4. Релации во онтологијата

Во рамките на functionalOWL онтологијата се дефинирани релациите hasNonstaticValue, hasParameter, hasInput, hasOutput, hasValue, parameterType и hasName.

Релацијата hasNonstaticValue како домен ја има дефинирано класата owl:Thing, која е најопштата класа во една онтологија. Опсегот на hasNonstaticValue е класата Service, со што се овозможува на инстанца од која било класа (која истовремено е инстанца и од owl:Thing, во секоја онтологија) да ѝ се даде нестатичка вредност, односно нејзината вредност да биде еднаква со излезот кој го произведува сервисот кој ќе биде пронајден според дефиницијата на инстанцата од класата Service.

Бидејќи релацијата hasNonstaticValue поврзува две класи, таа претставува објектно својство.

Дефиниција на релацијата hasNonstaticValue:

```
<owl:ObjectProperty rdf:about="#hasNonstaticValue">
  <rdfs:range rdf:resource="#Service"/>
  <rdfs:domain rdf:resource="&owl;Thing"/>
</owl:ObjectProperty>
```

Релацијата hasParameter како домен ја има класата Service, а како опсег ја има класата Parameter. На тој начин се овозможува поврзување на инстанца од класата Service со инстанца од класата Parameter. Но, во functionalOWL онтологијата не е предвидено да се креираат инстанци од класата Parameter, затоа што во хиерархијата на онтологијата неа ја користиме како заедничка родител-класа за класите Input и Output.

Според тоа, релацијата hasParameter се користи за поврзување на инстанци од класата Service со инстанци од класите Input и Output, со цел да се дефинира скелет за потенцијалните сервиси кои системот треба да ги пронајде, а треба да ја извршат соодветната задача, односно да ја вратат потребната информација. Поврзувањето со инстанци од класите Input и Output е возможно затоа што со наследување од класата Parameter, овие две класи влегуваат во опсегот на hasParameter релацијата.

Дефиниција на релацијата hasParameter:

```
<owl:ObjectProperty rdf:ID="hasParameter">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="#Parameter"/>
</owl:ObjectProperty>
```

Релациите hasInput и hasOutput се дефинирани како релации наследени од hasParameter. Тие се користат за додавање влезни и излезни параметри на инстанци од класата Service.

Дефиниции на релациите hasInput и hasOutput:

```
<owl:ObjectProperty rdf:ID="hasInput">
  <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
  <rdfs:range rdf:resource="#Input"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="hasOutput">
  <rdfs:subPropertyOf rdf:resource="#hasParameter"/>
  <rdfs:range rdf:resource="#Output"/>
</owl:ObjectProperty>
```

Релацијата parameterType се користи за поврзување на инстанци од класите Parameter, Input и Output со вредност која го опишува нивниот тип. Како домен на релацијата е дефинирана класата Parameter, додека пак опсегот е недефиниран. Недефинираноста на опсегот дозволува како тип на параметар да се дефинира инстанца од која било класа во дадена онтологија, или пак типот да се дефинира како XML Schema, односно XSD податочен тип.

Дефиниција на релацијата parameterType:

```
<rdf:Property rdf:ID="parameterType">
  <rdfs:domain rdf:resource="#Parameter"/>
  <rdfs:comment>
      Range is left unspecified, to allow for both OWL classes
and XSD datatypes.
  </rdfs:comment>
</rdf:Property>
```

Релацијата hasValue е наменета за поврзување на инстанца од класата Parameter, односно инстанци од класите Input и Output, со конкретна вредност, која може да биде произволна. Тоа значи дека опсегот на оваа релација, слично како и опсегот на релацијата parameterType, е недефиниран.

Дефиниција на релацијата hasValue:

```
<rdf:Property rdf:ID="hasValue">
  <rdfs:domain rdf:resource="#Parameter"/>
  <rdfs:comment>
      Range is left unspecified, to allow for both OWL classes
and XSD datatypes.
      release.)
  </rdfs:comment>
</rdf:Property>
```

Релацијата hasName се користи за поврзување на инстанца од класата Service со конкретно име, кое има вредност од типот литерал, односно низа од знаци.

Дефиниција на hasName:

```
<rdf:Property rdf:ID="hasName">
  <rdfs:domain rdf:resource="#Service"/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Literal"/>
</rdf:Property>
```

Според дефиницијата, доменот на релацијата hasName е класата Service, додека опсегот е литерал, односно низа од знаци, дефинирана во RDF Schema именскиот простор.

4.3. Систем за работа со семантички веб сервиси

Проширувањето на онтологиите со функции има за цел да ја зголеми експресивната и функционалната моќ на технологиите на Семантичкиот Веб, со тоа што на едноставен начин ќе се овозможи развој на покомплексни системи, во кои податоците нема да бидат само статични, сместени во база на знаење, туку ќе можат да бидат и често променливи, комплексни податоци, како и податоци кои на системите им се потребни во реално време.

Откако ја презентиравме онтологија која овозможува искористување на функционални вредности во рамките на еден систем базиран само на технологиите на Семантичкиот Веб, сега ќе претставиме систем кој ќе го разбере она што оваа онтологија за проширување го означува, систем кој ќе знае како да ги поврзе дефинициите на сервисите со конкретни, семантички анотирани веб сервиси и систем кој ќе ги повика истите, со цел да се добие бараниот податок.

Овој систем треба да обезбеди интеграција на едноставни, односно *атомични* сервиси, како и на комплексни, односно *композитни* сервиси. Атомични сервиси се сервиси кај кои се повикува само еден веб сервис, се праќа

порака со барање (кое најчесто содржи одредени податоци) и притоа може да се добие и одговор. Кај атомичните сервиси карактеристично е што по завршувањето на повикот и добивањето на евентуален одговор, прекинува интеракцијата помеѓу сервисот и барателот на сервисот, што во случајот е нашиот систем. Во оваа категорија на сервиси спаѓа, на пример, сервис кој за дадена адреса го одредува поштенскиот број или можеби географската ширина и должина. Системот до овој сервис испраќа барање, заедно со податокот (во овој случај, адресата) и назад добива одговор од сервисот, заедно со бараните информации. По ова, нивната интеракција престанува.

За разлика од атомичните сервиси, композитните сервиси се составени од повеќе поедноставни сервиси, при што е потребна поголема интеракција помеѓу системот како барател на сервисот и множеството сервиси. Купувањето на книги на Интернет има ваква природа; корисникот пребарува книги според одреден критериум, по можност чита критики, се одлучува дали ќе купи некоја од книгите или не, и остава податоци од кредитната картичка и контакт информации.

Системот, кој го подржува проширувањето на онтологиите со функции, всушност, има за цел да ги овозможи следните три сценарија:

- **Автоматско наоѓање на веб сервиси**. Автоматското наоѓање на веб сервиси претставува автоматски процес за лоцирање на веб сервисите кои можат да ја извршат задачата која е специфицирана во онтологијата, преку едноставно споредување на влезните и излезните параметри со кои се бара извршување на задачата, со влезните и излезните параметри на веб сервисите кои му се на располагање на системот. На пример, во онтологијата може да постои дефиниција на ресурс која кажува дека моменталната температура (како сервисно својство) во одреден град (како онтолошки ресурс) се добива со повик на некој сервис, кој како влезен параметар добива име на град, а како излезен параметар враќа моментална температура за тој град. Корисникот кој работи со податоци кои се моделирани со онтологијата, доволно е да инстанцира запис за одреден град, без да мора да се грижи за наоѓање на соодветен сервис и без да специфицира на кој начин ќе ја добива информацијата за моменталната температура. Системот, при обработка на податоците на корисникот, ќе го препознае овој дел во кој корисникот бара моментална температура за градот за кој поставил запис. Задача на системот е да најде еден или повеќе кандидат сервиси кои можат да ја извршат задачата специфицирана во онтологијата.
- **Автоматско повикување на веб сервиси.** Автоматско повикување на веб сервиси се изведува со повик на веб сервиси преку системот. Идејата е самиот систем, откако ќе најде соодветен веб сервис, да го повика сервисот,

при што ќе му ги пренесе потребните аргументи, ќе го добие одговорот од сервисот, ќе го обработи и ќе го прикаже.

- **Автоматска композиција на веб сервиси**. Оваа акција вклучува автоматска селекција и композиција на веб сервисите, со цел да се изврши одредена покомплексна задача, за чие решавање не постои еден конкретен веб сервис во системот. Во класичниот пристап, потребно е клиентот да ги пронајде сите веб сервиси кои би му помогнале да ја заврши конкретната задача што ја има, да одреди определен тек на податоците и да ги повикува веб сервисите еден по еден, според определениот тек. Целта на овој систем е да го ослободи клиентот од ваквата „мануелна“ работа, со тоа што од него би се барало само дефинирање на задачата која е потребно да се изврши (со дефинирање на влезните параметри и бараниот излез). Системот е одговорен за автоматско наоѓање, селекција, композиција и повик на сервисите, со цел да се изврши зададената комплексна задача.

Целата изведба на проширувањето на онтологиите со функционалност и на системот, има за цел да им ја олесни задачата на корисниците на сервиси, со тоа што дефиницијата на потребните функционалности во рамките на онтологиите се сведува на користење на класата Service, за која треба да дефинираат еден или повеќе влезни параметри и еден излезен параметар, и која треба да ја поврзат со друг ресурс, преку својството hasNonstaticValue.

4.3.1. Автоматска композиција на семантички веб сервиси

Автоматска композиција на семантички веб сервиси претставува акција која вклучува автоматска селекција и композиција на семантички анотирани веб сервиси, со цел да се изврши одредена покомплексна задача, за чие решавање не постои еден конкретен, семантички опишан веб сервис во системот.

Да го разгледаме следното сценарио: развивач на одредена апликација сака на дното од почетната страна на апликацијата да се прикажува моменталната температура во градот во кој што се наоѓа седиштето на компанијата која ја користи неговата апликација. Да претпоставиме дека во базата на знаење постои информација само за името на компанијата која ја користи апликацијата, но не и за градот во кој се наоѓа нејзиното седиште. Имајќи го предвид ова, развивачот на апликацијата може со functionalOWL онтологијата во апликацијата да дефинира дека податокот за моменталната температура ќе се зема преку сервис кој како влез прима аргумент од типот компанија, а на излез враќа моментална температура во градот во кој што се наоѓа нејзиното седиште. Во ваков случај, задача на системот е да најде сервис кој ќе го врати бараниот резултат.

Доколку во системот не постои еден ваков семантички анотиран веб сервис, кој може директно да одговори на барањето, системот ќе се обиде да најде *множество* на веб сервиси, со чија композиција би можело да се даде одговор на барањето. Едно решение на проблемот е системот да пронајде, на пример, три веб сервиси, од кои: првиот како влезен аргумент прима податок од типот компанија, а како излез го враќа градот во која е основана фирмата; вториот веб сервис прима податок од тип град, а ја враќа географската позиција на градот; третиот веб сервис врз основа на влезот од тип географска позиција, ја одредува и враќа моменталната температура, која што е и бараниот податок.

На тој начин, со автоматска композиција на овие три семантички веб сервиси, системот го наоѓа бараниот податок.

Ваквиот систем не може да се изведе само со технологиите на Семантичкиот Веб, поради нивната декларативна природа. Поради тоа, потребно е овој систем да биде реализиран во програмски јазик и рамка за развој кои имаат доволно голема моќ да се справат со барањата на самиот систем.

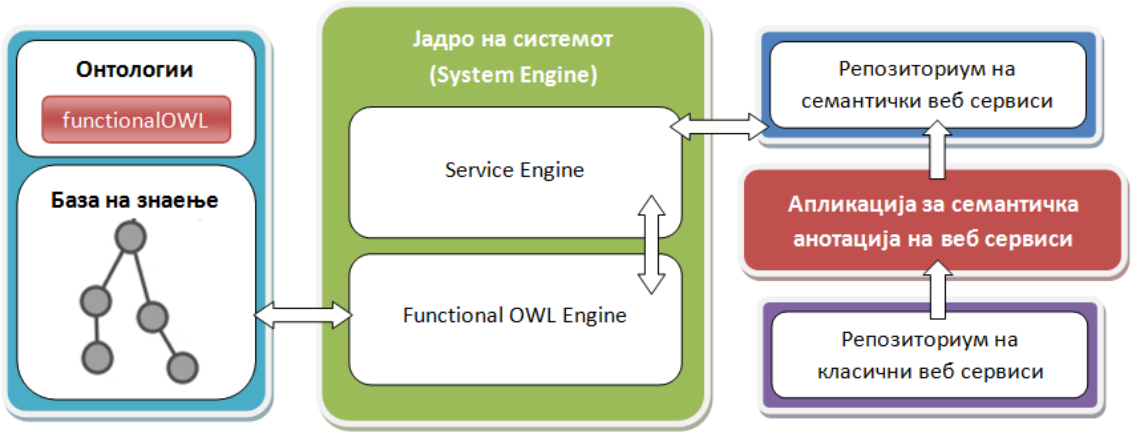
5. Имплементација на системот

5.1. Архитектура

Проширувањето на онтологиите со функции и системот за работа со семантички веб сервиси, опишани во четвртото поглавје, има за цел да обезбедат поголема функционална моќ на онтологиите, а со тоа и на технологиите на Семантичкиот Веб, генерално. Како што веќе покажавме, проширувањето на онтологиите е реализирано со развој на functionalOWL онтологијата, која ја моделира рамката на планираното проширување и која овозможува на ресурси од која било онтологија да им бидат придружени нестатички вредности, кои се добиваат со повик на семантички веб сервиси.

Архитектурата на целиот систем за проширување на онтологиите е прикажана на слика 5-1. Тој се состои од неколку дела: база на знаење изградена врз основа на множество од онтологии помеѓу кои е вклучена и functionalOWL онтологијата, јадрото на системот кое е составено од два дела, Service Engine и Functional OWL Engine, репозиториуми на класични и семантички веб сервиси, како и апликација за семантичка анотација на веб сервиси.

Во базата на знаење е содржан податочниот слој на одредена апликација, во која со помош на functionalOWL онтологијата се дефинирани ентитети чии што вредности се добиваат преку повик на сервис со одреден потпис (број и тип на влезни параметри и тип на излезен параметар).



Слика 5-1: Архитектура на системот

Јадрото на системот, преку својот Functional OWL Engine е задолжено за детектирање на сите инстанци од класата Service во рамките на базата на знаење, идентификување на влезните параметри и нивните семантички типови и одредување на семантичкиот тип на излезниот параметар. Овие податоци се препраќаат до делот Service Engine од јадрото на системот, кој е задолжен за пребарување на репозиториумот на семантички веб сервиси и наоѓање на соодветни веб сервиси, односно композиции од веб сервиси кои можат да го исполнат барањето од базата на знаење, односно да го вратат бараниот податок.

Откако ќе се пронајде соодветниот семантички веб сервис, односно композиција од семантички веб сервиси, делот Service Engine е одговорен за повик на овој сервис, односно композиција, и враќање на резултатот до Functional OWL Engine. Понатаму, Functional OWL Engine го поставува резултатот на соодветното место назад во базата на знаење.

Апликацијата за семантичка анотација на веб сервиси се користи за пополнување на репозиториумот на семантички веб сервиси, преку семантичка анотација на класичните веб сервиси кои се наоѓаат во одреден репозиториум.

Имплементацијата на секој од деловите на системот во детали е дадена во продолжение.

5.2. Имплементација на јадрото

Делот од системот кој треба да го обезбеди наоѓањето на кандидат сервиси, да ја направи селекцијата на најсоодветните сервиси и кој треба да ги повика, го имплементиравме со помош на Jena развојната рамка за семантички апликации [18]. Поконкретно, спецификациите за тоа што овој систем треба да обезбеди, кои беа дадени во делот 4, ги имплементиравме во програмскиот јазик Java, со

користење на Jena развојната рамка, преку имплементација на алгоритмот кој обезбедува автоматска композиција на семантички веб сервиси.

Алгоритмот се состои од неколку чекори:

- Чекор 1: детектирање на нестатичка вредност на инстанци во базата на знаење;
- Чекор 2: наоѓање на кандидат семантички веб сервиси;
- Чекор 3: селекција на најпогодниот сервис, или на најпогодната композиција сервиси;
- Чекор 4: повикување на семантичкиот веб сервис, или на композицијата сервиси;
- Чекор 5: враќање на резултатот назад во онтологијата, односно во базата на знаење.

За извршување на наведените чекори од алгоритмот, креиравме два Јава пакети, mk.edu.ukim.feit.etnc.jena.functionalOWL кој ги содржи методите за извршување на првиот чекор од алгоритмот, и mk.edu.ukim.feit.etnc.jena.serviceEngine кој ги содржи методите за извршување на останатите чекори.

5.2.1. Детекција на нестатичка вредност на инстанци

Првиот чекор кој имплементираниот систем го извршува е детектирање на дефиниција за нестатичка вредност на инстанца од дадена база на знаење. Постапката се состои во идентификување на инстанците од одредена база на знаење кои ја користат релацијата hasNonstaticValue од нашата functionalOWL онтологија, за спецификација на нивната вредност, односно инстанците чија вредност се добива преку повик на одреден сервис.

За оваа функционалност, во нашиот систем го креиравме Јава пакетот mk.edu.ukim.feit.etnc.jena.functionalOWL, во кој се наоѓаат Јава класите FunctionalOWLEngine и FunctionalServiceUtils.

FunctionalOWLEngine класата имплементира обвиткувачи околу функционалностите на Jena рамката за развој на семантички апликации, со кои се овозможува едноставна работа со онтологии, OWL и RDF податоци. Функциите од оваа класа се користат за вчитување на онтологиите и базите на знаење кои треба да се обработат, со цел да се детектираат потребните инстанци.

FunctionalServiceUtils класата се состои од методи кои пребаруваат низ селектираните онтологии и бази на знаење, со цел да ги соберат потребните информации и дефиниции кои му се потребни на системот за да може да продолжи со понатамошните чекори.

Методите дефинирани во FunctionalServiceUtils класата кои ни се од интерес се:

- getAllServices(OntModel model);
- getInputParameters(OntModel model, OntResource service);
- getOutputParameter(OntModel model, OntResource service);
- getInputType(OntModel model, OntResource input);
- getOutputType(OntModel model, OntResource output);
- findSemanticWebServices(OntModel model, OntResource service).

Методот getAllServices(OntModel model) пребарува низ онтолошкиот модел, односно базата на знаење, со цел да ги детектира инстанците кои преку релацијата hasNonstaticValue земаат вредност преку инстанца од класата Service од functionalOWL онтологијата.

Методот getInputParameters(OntModel model, OntResource service) за одреден сервис дефиниран во базата на знаење како инстанца од класата Service од functionalOWL онтологијата, ги наоѓа дефинициите на влезните параметри, како инстанци од класата Input, поврзани со инстанцата од Service со релацијата hasInput.

Методот getOutputParameter(OntModel model, OntResource service), пак, за одреден сервис дефиниран исто така во базата на знаење како инстанца од класата Service од functionalOWL онтологијата, ја наоѓа дефиницијата на излезниот параметар, како инстанца од класата Output, поврзана со инстанцата од Service со релацијата hasOutput.

Методот getInputType(OntModel model, OntResource input) се користи за да од инстанца на класата Input го најде типот на влезниот параметар, како вредност на релацијата parameterType.

Методот getOutputType(OntModel model, OntResource output) претставува метод кој за инстанца од класата Output го наоѓа типот на излезниот параметар, како вредност на релацијата parameterType.

Методот findSemanticWebServices(OntModel model, OntResource service) е метод кој за одредена инстанца на сервис од базата на знаење, со користење на претходно дефинираните методи, ги наоѓа типовите на сите влезни параметри и типот на излезниот параметар и преку повик на методот findCompatibleService од класата ServiceEngineUtils (дефинирана во пакетот mk.edu.ukim.feit.etnc.jena.serviceEngine) го наоѓа најсоодветниот семантички веб сервис, односно најсоодветната композиција од семантички веб сервиси.

5.2.2. Наоѓање на кандидат семантички веб сервиси

Вториот чекор во алгоритмот е наоѓање на кандидат семантички веб сервиси, кои би можеле да дадат решение на поставеното барање. Како што веќе прецизиравме, барањето се поставува преку functionalOWL онтологијата, односно со инстанцирање на класата Service, на која ѝ се придружуваат еден или повеќе влезни параметри и еден излезен параметар.

При обработка на податоците од базата на знаење, системот го зема Service елементот, од кој ги чита неговите влезни параметри, како и излезниот параметар, преку методите дефинирани во класата FunctionalServiceUtils од пакетот mk.edu.ukim.feit.etnc.jena.functionalOWL. Преку релацијата parameterType на параметрите, го зема типот на влезниот параметар, односно на секој од влезните параметри кога се повеќе, го зема и типот на излезниот параметар. На тој начин, системот знае кој е податокот кој корисникот го бара преку инстанцата на Service, а кои се податоците кои ги има на располагање и со кои сака да го добие бараниот податок. Користејќи го репозиториумот на веб сервиси кои се семантички анотирани со користење на SAWSDL јазикот за анотација и чии влезни и излезни параметри се семантички опишани, системот ќе се обиде да го најде бараниот податок.

За имплементација на оваа функционалност, во системот го креиравме пакетот mk.edu.ukim.feit.etnc.jena.serviceEngine, кој ја содржи Јава класата ServiceEngineUtils. Во класата се дефинирани два методи:

- findCompatibleService(OntModel model, List<OntResource> inputTypes, OntResource outputType);
- callService(String serviceDescription);

Првиот метод, findCompatibleService(OntModel model, List<OntResource> inputTypes, OntResource outputType), е одговорен за вториот и третиот чекор од алгоритмот, односно за наоѓање на кандидат семантички веб сервиси или композиции од семантички веб сервиси, нивно рангирање и одредување на најпогодниот сервис, односно композиција од сервиси.

Вториот метод се користи за повик на одреден сервис и враќање на резултатот назад во базата на знаење. Повеќе детали за овој метод се изложени во деловите 5.2.4 и 5.2.5.

За да го најде најсоодветниот семантички веб сервис, односно најсоодветната композиција од семантички веб сервиси, системот, преку методот findCompatibleService(OntModel model, List<OntResource> inputTypes, OntResource outputType), се обидува да идентификува дали во репозиториумот на семантички веб сервиси со кои располага, постои веб сервис кој што враќа податок кој е од ист

семантички тип како и податокот кој корисникот го бара, преку дефинициите од неговата база на знаење. Доколку постои еден или повеќе такви семантички веб сервиси, системот за секој од нив проверува дали неговите влезни аргументи се од истиот тип како влезните аргументи на инстанцата на Service.

Ако со o_r го означиме типот на излезниот податокот кој корисникот го бара, а со o_i го означиме типот на излезниот податок кој го враќа *i*-тиот семантички веб сервис од репозиториумот, тогаш она што системот се обидува да направи е да ги лоцира семантичките веб сервиси за кои важи:

$$o_r = o_i \tag{5.1}$$

Ваквите семантички веб сервиси претставуваат *кандидат семантички веб сервиси* за добивање на бараниот податок.

За споредување на семантичкиот тип на излезот и на влезовите, ја користиме библиотеката SAWSDL4J, Јава библиотека за работа со семантички анотирани WSDL документи, односно SAWSDL документи [65].

За секој семантички веб сервиси од репозиториумот кој ќе го исполни условот да семантичкиот тип на излезниот податок е ист со типот на бараниот податок, односно кој ќе го задоволи равенството (5.1), системот треба да направи споредби помеѓу множеството на типови на влезните податоци $I_r = \{i_{r1}, i_{r2}, ..., i_{rn}\}$, специфицирани од корисникот, и множеството на типови на влезните податоци на семантичкиот веб сервис $I_i = \{i_{i1}, i_{i2}, ..., i_{im}\}$. Доколку за множествата I_i и I_r важи дека:

$$|I_i| < |I_r| \tag{5.2}$$

односно $m < n$, тогаш системот го елиминира *i*-тиот семантички веб сервис од потенцијалните кандидат сервиси, поради тоа што бројот на влезните параметри на сервисот е помал од бројот на влезни параметри специфицирани од корисникот. На тој начин се елиминира можноста корисникот да добие податок кој се добива со помалку ограничувања, односно се елиминира можноста добиениот податок да биде недоволно прецизен, или пак, погрешен.

Доколку множествата I_i и I_r се еднакви, односно доколку важи:

$$I_i = I_r \tag{5.3}$$

тоа значи дека бројот и типовите на влезните параметри специфицирани од корисникот и бројот и типовите на влезните параметри на i-тиот семантички веб сервис се еднакви, па системот на соодветниот семантички веб сервис му доделува вредност на коефициентот на погодност (fitting coefficient, *F*):

$$F = 1$$

Коефициентот на погодност - *F*, ја претставува погодноста на одреден семантички веб сервис, или композиција од семантички веб сервиси, да одговори на поставеното барање на корисникот.

Во случај кога е задоволено равенството (5.3), односно кога одреден семантички веб сервис прима влезни параметри кои се исти по број и по семантички тип како зададените и враќа резултат од ист семантички тип како бараниот параметар, сервисот се смета за најпогоден. Со користење на ваков сервис, системот може да го добие бараниот резултат со само еден повик до сервисот, што може да го сметаме како најдобар случај.

Секогаш кога системот ќе најде барем еден сервис со коефициент на погодност $F = 1$, алгоритамот за наоѓање на кандидат сервиси завршува.

Доколку, на пример, корисникот специфицира три влезни параметри претставени со множеството на типови $I_r = \{i_{r1}, i_{r2}, i_{r3}\}$ и излезен параметар со тип o_r, како најпогодни (со коефициент на погодност $F = 1$) се сметаат само сервисите кои го враќаат бараниот резултат со користење на сите три параметри за да го добијат бараниот резултат, односно сервисите за кои важат равенствата (5.1) и (5.3). Сервисите кои го враќаат бараниот резултат, а притоа користат помалку од три влезни параметри, односно за кои е исполнето равенството (5.2), ги отфрламе како непогодни.

Доколку при првото пребарување системот не најде соодветен веб сервис, кој како излезен параметар враќа податок со ист тип како и бараниот податок, односно сервис за кој важи равенството (5.1), системот не враќа резултат. Во овој случај алгоритмот завршува неуспешно.

Доколку за множествата I_i и I_r важи дека:

$$|I_i| \geq |I_r| \tag{5.4}$$

но притоа типовите на сите влезни параметри на *i*-тиот семантички веб сервис не соодветствуваат со типовите на влезните параметри специфицирани од корисникот, коефициентот на погодност на *i*-тиот сервис се пресметува како:

$$F_i = \frac{y_{fi}}{y_i} \tag{5.5}$$

каде y_{fi} е бројот на параметри од *i*-тиот семантички веб сервис кои според типот се совпаѓаат со влезните параметри од I_r, односно $y_{fi} = |I_{fi}|$, каде $I_{fi} \subseteq I_i$, а y_i е вкупниот број на влезни параметри на *i*-тиот семантички веб сервис, односно $y_i = |I_i|$.

За да се повика одреден веб сервис, потребно е да се испратат сите параметри на сервисот, за да може да се добие соодветниот резултат. Имајќи го тоа предвид, а со цел да се искористи пронајдениот семантички веб сервис, потребно е да се обезбедат и останатите параметри кои не припаѓаат на I_{fi}, односно параметрите кои корисникот не ги специфицирал.

За таа цел, системот започнува со повторно пребарување низ репозиториумот на сервиси, со тоа што сега типот на излезниот параметар кој се бара, o_r, е типот на влезниот параметар од *i*-тиот кандидат веб сервис кој не е специфициран од корисникот. Доколку има повеќе вакви влезни параметри, пребарувањето се прави за секој од нив.

Доколку пронајдените кандидат семантички веб сервиси од втората итерација имаат исти типови на влезни параметри како параметрите специфицирани од корисникот, тие можат веднаш да се повикаат, па нивниот резултат да се искористи како влезен параметар за кандидат семантичкиот веб сервис од првата итерација. Доколку и тие имаат параметри кои не се специфицирани од корисникот, системот ќе мора да направи уште една итерација низ репозиториумот на сервиси за да пронајде соодветен сервис за конкретните параметри. Итерациите траат сé додека системот не дојде состојба во која се сите пронајдени сервиси имаат влезни параметри од ист тип како влезните параметри специфицирани од корисникот, или пак, во состојба во која не може да најде соодветен кандидат семантички веб сервис кој враќа податок од семантички тип како бараниот податок во итерацијата.

Ваквиот алгоритам всушност пребарува композиција од семантички веб сервиси, чие извршување во правилен редослед би го вратило бараниот податок од корисникот.

Овој начин на добивање на бараниот податок, преку извршување на композиции од семантички веб сервиси, внесува дополнителни компликации. Со големината на композицијата во нивоа и според бројот на сервиси вклучени во композицијата, пропорционално расте цената на добивањето на податокот. Се зголемува времето на чекање, поради потребата од повикување на повеќе веб сервиси и поради потребата од пренесување одреден број параметри при повиците, а се зголемува и можноста за грешка при извршувањето на композицијата. Според тоа, во равенството (5.5) треба да се вклучат и овие влијанија. Тоа го правиме преку дефинирање на *коефициент на намалување на погодноста, K*:

$$K_i = \sum_{j=1}^{p} (k_1^{-1} + y_j k_2^{-1}) \qquad (5.6)$$

каде p е вкупниот број на сервиси во композицијата, без основниот сервис (сервисот од првата итерација), y_j е бројот на влезни параметри на j-тиот сервис, а k_1 и k_2 се т.н. *фактори на намалување на погодноста*, чија вредност може да се менува во рамките на системот. Факторот k_1 е *фактор на влијание на бројот на сервиси* вклучени во композицијата, додека пак факторот k_2 е *фактор на влијание на бројот на параметри на сервисите* вклучени во композицијата. Генерално, за овие фактори треба да важи $k_1 < k_2$, поради тоа што бројот на сервиси во композицијата повеќе влијае на времето потребно да се изврши целата композиција и да се добие бараниот податок, отколку бројот на параметри на секој од сервисите. Во системот предефинираната вредност за $k_1 = 10$, а за $k_2 = 100$.

Од (5.6) може да се заклучи дека системот не го зема предвид нивото во композицијата на кое се наоѓа j-тиот сервис. Причината за ова е фактот дека повикувањето на сервиси од исто ниво се извршува секвенцијално, исто како и повикувањето на сервиси од различни нивоа, па времето потребно за добивање на бараниот податок зависи само од бројот на сервиси во композицијата, не и од нивната нивовска распределеност.

Исто така, од (5.6) може да се увиди дека коефициентот на намалување на погодноста зависи од бројот на параметри кои се користат кај секој од сервисите од композицијата, без разлика дали тие параметри се специфицирани од страна на корисникот, или се добиваат преку друг семантички веб сервис. Причината за ова лежи во тоа што бројот на параметрите влијае врз големината на податоците кои треба да се пренесат при повикот на овие сервиси. Според тоа, колку повеќе параметри се користат од сервисите, толку е поголема вредноста на K.

Со додавање на коефициентот на намалување на погодноста во равенството (5.5), се добива:

$$F_i = \frac{y_{fi}}{y_i} - K$$

$$F_i = \frac{y_{fi}}{y_i} - \sum_{j=1}^{p} (k_1^{-1} + y_j k_2^{-1}) \qquad (5.7)$$

Со (5.7) се пресметува коефициентот на погодност за сите кандидат семантички веб сервиси за кои важи равенството (5.4). Специјален случај е случајот кога сите влезни параметри на сервисот се совпаѓаат по семантички тип со сите влезни параметри специфицирани од корисникот, при што вредноста на $K = 0$, а вредноста на $F = 1$.

Коефициентот на погодност е поголем доколку кандидат сервисот користи поголем дел од влезните параметри специфицирани од страна на корисникот. Коефициентот опаѓа со бројот на дополнителни сервиси и бројот на нивните

влезни параметри, кои се користат за да се добијат одредени меѓу-параметри, потребни за извршување на кандидат сервисот.

Да го земеме следниот пример: во базата на знаење постои спецификација за повик кон сервис кој прима два влезни параметри. Кога системот ќе почне со барање на кандидат семантички веб сервиси од репозиториумот, ќе ги лоцира сервисите кои враќаат податок од ист тип како бараниот. Да претпоставиме дека еден од пронајдените сервиси прима три влезни параметри, од кои два се од ист тип како оние кои ги специфицирал корисникот. Тогаш неговиот коефициент на погодност F, во првата итерација ќе се пресмета како:

$$F = \frac{y_f}{y}$$

каде y_f е бројот на влезови на сервисот кои се совпаѓаат со влезовите специфицирани од корисникот, а y е вкупниот број на влезови на сервисот. Според тоа, во овој случај имаме:

$$F = \frac{2}{3} = 0{,}66$$

Во следната итерација, системот го повторува барањето на сервиси, со тоа што сега типот на излезниот параметар кој се бара, е типот на влезниот параметар од првиот кандидат сервис кој не е специфициран од корисникот.

Доколку во втората итерација системот пронајде друг семантички веб сервис чиј излезен параметар е од ист тип со влезниот параметар од првиот сервис, новонајдениот семантички веб сервис, заедно со првиот, формираат *композиција од семантички веб сервиси*. Оваа композиција има свој коефициент на погодност:

$$F = \frac{y_f}{y} - \sum_{j=1}^{p}(k_1^{-1} + y_j k_2^{-1}) = \frac{2}{3} - (\frac{1}{10} + \frac{1}{100}) = 0{,}55$$

каде $y_1 = 1$ е бројот на влезните параметри на сервисот од втората итерација, $k_1 = 10$, а $k_2 = 100$.

На овој начин, системот им дава поголема предност на атомичните сервиси, односно семантичките веб сервиси кои без композиција, директно можат да го дадат бараниот податок, врз база на податоците специфицирани и проследени од корисникот, преку онтологијата. Истовремено, поголема предност им се дава и на пократките композиции, наспроти подолгите, како и на композициите и сервисите кои користат помалку влезни параметри. Причината за ова е доверливоста и големината на податоците кои треба да се пренесат.

Со помош на коефициентот за погодност, системот понатаму може да се одлучи кој семантички веб сервис или која композиција од семантички веб

сервиси ќе ја селектира и ќе ја повика, за да на корисникот му го врати бараниот податок.

5.2.3. Селекција на најпогодниот сервис или композиција

Селекцијата на најпогодниот сервис се прави врз основа на коефициентот на погодност. Системот едноставно ги рангира семантичките веб сервиси и композициите според коефициентот, при што го одбира сервисот со највисока вредност на коефициентот. Доколку се случи повеќе сервиси и композиции да имаат иста вредност на *F*, системот произволно одбира еден сервис или композиција.

Во случај на недостапност на селектираниот семантички веб сервис или недостапност на еден од сервисите од дадена композиција, системот автоматски ја менува селекцијата и го повикува следниот сервис или композиција од сервиси од листата.

Оваа функционалност на системот е реализирана во рамките на методот findCompatibleService(OntModel model, List<OntResource> inputTypes, OntResource outputType), кој е дел од нашиот Јава пакет mk.edu.ukim.feit.etnc.jena.serviceEngine.

5.2.4. Повик на селектираниот сервис или композиција

Следниот чекор во алгоритмот е повикот на селектираниот најпогоден семантички веб сервис или најпогодната композиција од сервиси. Бидејќи во системот се користи репозиториум на веб сервиси кои се семантички анотирани со користење на SAWSDL јазикот за анотација, системот ги има на располагање сите информации од анотираниот WSDL опис на секој од сервисите, за да може да направи повик до нив.

За оваа цел се користи методот callService(String serviceDescription) од ServiceEngineUtils класата.

Системот го зема Service елементот и од него ги чита вредностите на неговите влезни параметри. Овие параметри системот ги проследува до веб сервисот, користејќи стандардни SOAP пораки.

5.2.5. Добивање на резултатот

По стандардниот повик до веб сервисот или до композицијата од сервиси, преку методот callService(String serviceDescription) од ServiceEngineUtils класата, системот го чека резултатот. Доколку не добие одговор, се враќа повторно на селекција на следниот кандидат сервис, односно кандидат композиција и се

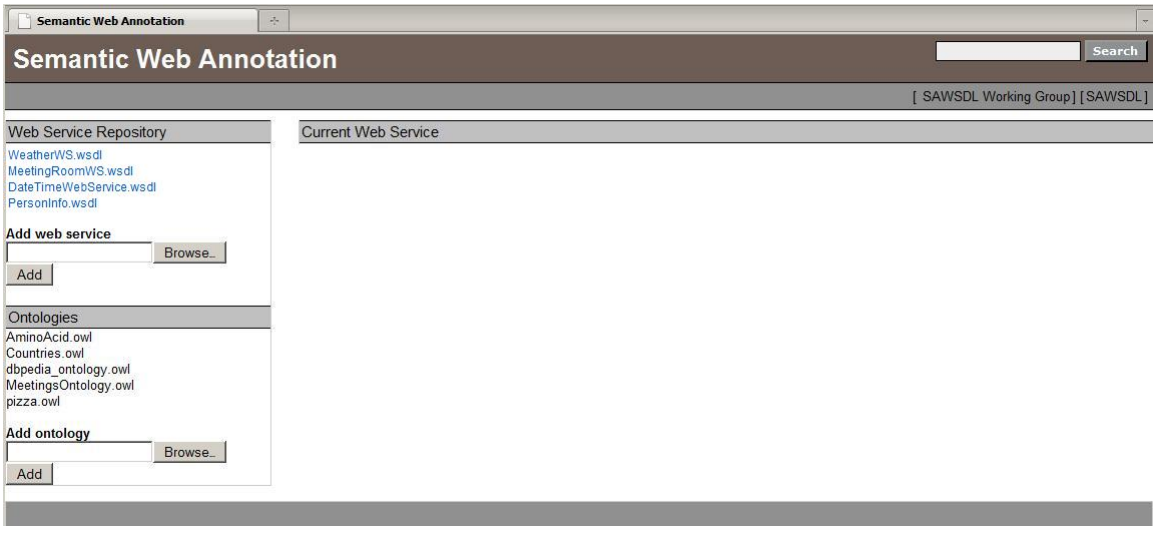
обидува повторно да направи повик. Доколку и после сите обиди системот не може да дојде до бараниот податок, се откажува и алгоритмот завршува.

Во спротивно, откако ќе го добие одговорот, системот го враќа овој одговор назад во податоците од кои и ја детектира потребата за повикување на веб сервис. Тоа може да бидат податоци запишани во RDF формат кои се наоѓаат во рамките на некоја веб страна, во рамките на презентацискиот слој на некоја апликација или пак, во рамките на базата на знаење на дадена апликација.

На тој начин, алгоритмот завршува успешно и на корисникот на, за него, едноставен начин му се овозможува пристап до динамички податоци, податоци кои не би можел да ги добие користејќи ги само технологиите на Семантичкиот Веб.

5.3. Веб апликација за семантичка анотација на веб сервиси

Системот имплементиран во Jena [18] рамката за развој на семантички апликации користи семантички веб сервиси. Овие сервиси, кои се наоѓаат во репозиториумот со кој располага овој систем, се добиваат со анотирање на класични веб сервиси. За анотација на веб сервисите со семантички описи креиравме посебна веб базирана апликација.



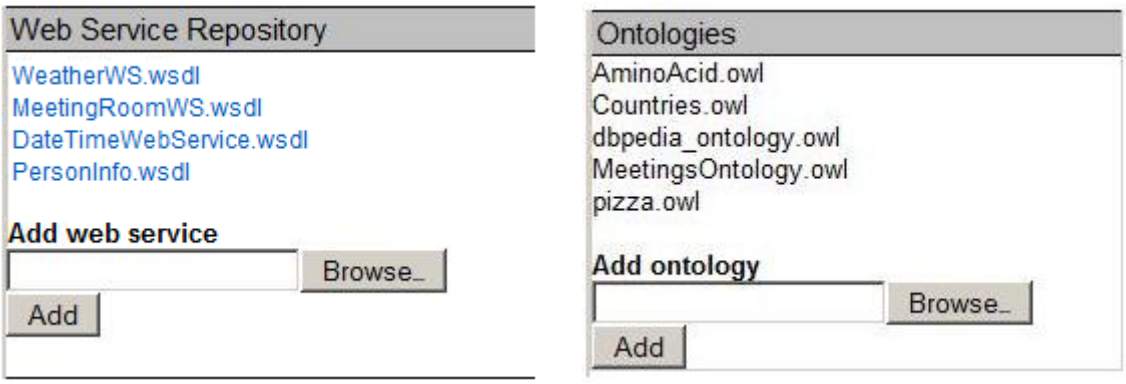
Слика 5-2: Веб апликација за семантичка анотација на веб сервиси

Веб апликацијата која ни овозможува семантичко анотирање на веб сервиси, го користи SAWSDL механизмот за поврзување на концепти од семантичките модели со WSDL компоненти. Апликацијата овозможува

полуавтоматска семантичка анотација на веб сервиси, при што корисникот избира одреден елемент од веб сервисот и го поврзува со конкретен семантички концепт од принудените онтологии. Онтологиите кои се користат при анотација се избираат од страна на корисникот. Со семантичкото анотирање на веб сервисите, овозможуваме нивно полесно откривање, како и динамичко конфигурирање.

При стартување на веб апликацијата, добиваме интерфејс како на слика 5-2.

Од левата страна се наоѓаат репозиториумот за веб сервиси и репозиториумот за онтологии (слика 5-3).



Слика 5-3: Репозиториум на веб сервиси и репозиториум на онтологии

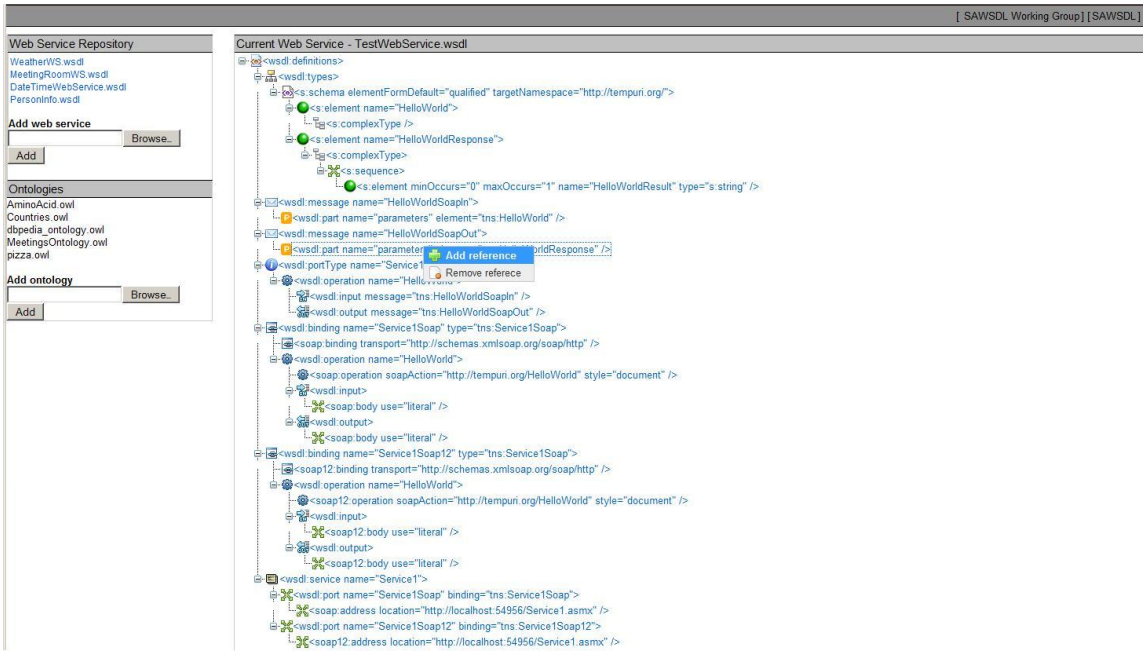
Репозиториумот од веб сервиси ги чува сите веб сервиси кои се семантички анотирани, или кои треба да бидат анотирано. Притоа, постои и опција за додавање на нови веб сервиси.

Веднаш под него се наоѓа репозиториумот од онтологии. Тој ги чува сите онтологии кои корисникот сака да ги користи при процесот на анотација на веб сервисот. Елементите од веб сервисот би се анотирале со семантичките концепти од овие онтологии. Притоа имаме и опција за додавање на нови онтологии, кои понатаму се користат за поврзување на нивните семантички концепти со компоненти од веб сервисите, во процесот на анотација.

При клик на некој од веб сервисите од репозиториумот, од десната страна (слика 5-4), се вчитува веб сервисот и се претставува како податочна структура дрво, со што започнува процесот на семантичко анотирање на елементите од веб сервисот.

Кога веб сервисот е прикажан како дрво податочна структура, претставен е секој негов елемент (операции, влезни и излезни елементи, интерфејси, итн.) заедно со сите негови атрибути. При избирање на друг веб сервис од репозиториумот, се вчитува новоизбраниот веб сервис.

Со десен клик врз некој од елементите на веб сервисот се појавува контекстно мени со две опции (слика 5-4). Првата опција служи за додавање на референца кон некој концепт од принудените онтологии, т.е. за поврзување на елементот од веб сервисот со некој семантички концепт од некоја онтологија. Втората опција, пак, се користи за бришење на веќе постоечка референца кон некој семантички концепт.



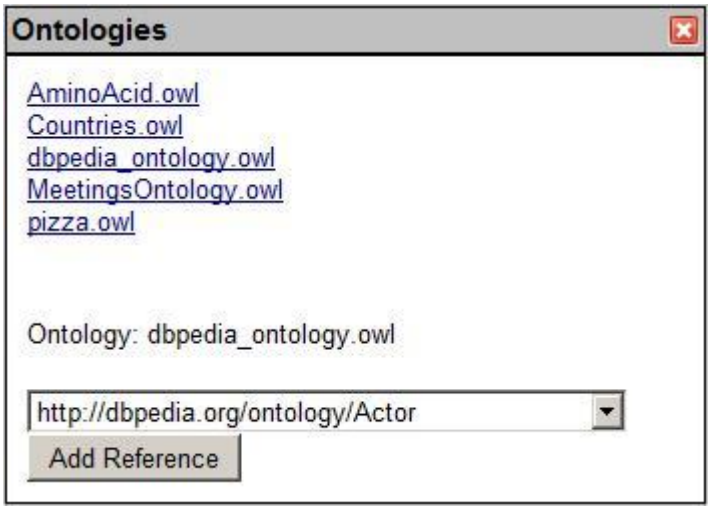
Слика 5-4: Претставување на веб сервисот како дрво податочна структура и приказ на контекстното мени

Со избор на првата опција од контекстното мени или опцијата за додавање на референца кон даден елемент, всушност, елементот се поврзува со даден семантички концепт, односно *семантички се анотира*.

Слика 5-5: Избор на дадена онтологија

Притоа, при кликнување се отвора нов прозорец, во кој се вчитуваат сите моментални онтологии кои се достапни во репозиториумот од онтологии и кој нуди можност да се избере која онтологија ќе се користи при анотацијата на елементот (слика 5-5).

При избор на некоја од принудените онтологии се вчитуваат сите нејзини семантички концепти (слика 5-6), па корисникот може да избере со кој семантички концепт сака да го поврзе WSDL елементот.



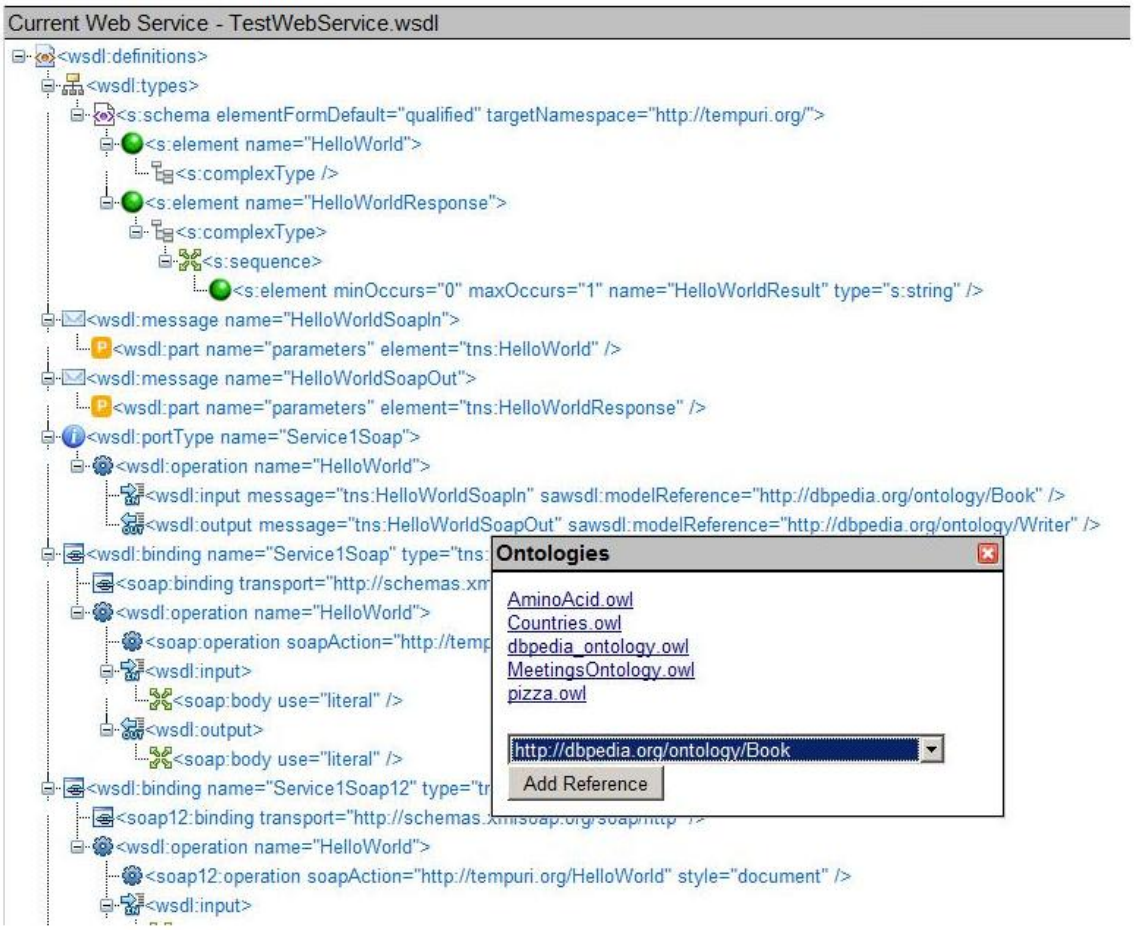
Слика 5-6: Вчитани семантички концепти од избраната онтологија

При избор на некој од принудените семантички концепти и со кликнување на „Add Reference“ копчето, се додава референца кон елементот, т.е. се додава sawsdl:modelReference кон избраниот концепт од онтологијата (слика 5-7).

```
<wsdl:operation name="HelloWorld">
    <wsdl:input message="tns:HelloWorldSoapIn" sawsdl:modelReference="http://dbpedia.org/ontology/Book" />
    <wsdl:output message="tns:HelloWorldSoapOut" sawsdl:modelReference="http://dbpedia.org/ontology/Writer" />
```

Слика 5-7: Анотирани елементи од веб сервисот

Ако корисникот има потреба од промена на некој семантички концепт од некој елемент од веб сервисот, тоа може да го изврши на два начини. Првиот подразбира прво да биде избришан постоечкиот семантички концепт, па повторно да се изврши претходната постапка. Вториот начин е директно додавање на нов семантички концепт, при што самата апликација се грижи да го отстрани постоечкиот семантички концепт и да го постави новоизбраниот.



Слика 5-8: Анотирање на WSDL елементи со концепти од онтологија

Како што веќе споменавме, постои и можност за бришење на веќе постоечка референца, со избор на опцијата "Remove reference" од контекст менито (слика 5-9).

Слика 5-9: Бришење на постоечки семантички концепт

При избор на опцијата за бришење на семантички концепт, се појавува нов прозорец кој го прашува корисникот дали сигурно сака да го избрише моменталниот семантички концепт (слика 5-10).

Слика 5-10: Прозорец за бришење на семантички концепт

При потврден одговор од корисникот, се брише поврзаноста на WSDL елементот со моменталниот семантички концепт (слика 5-11).

Слика 5-11: Избришан семантички концепт

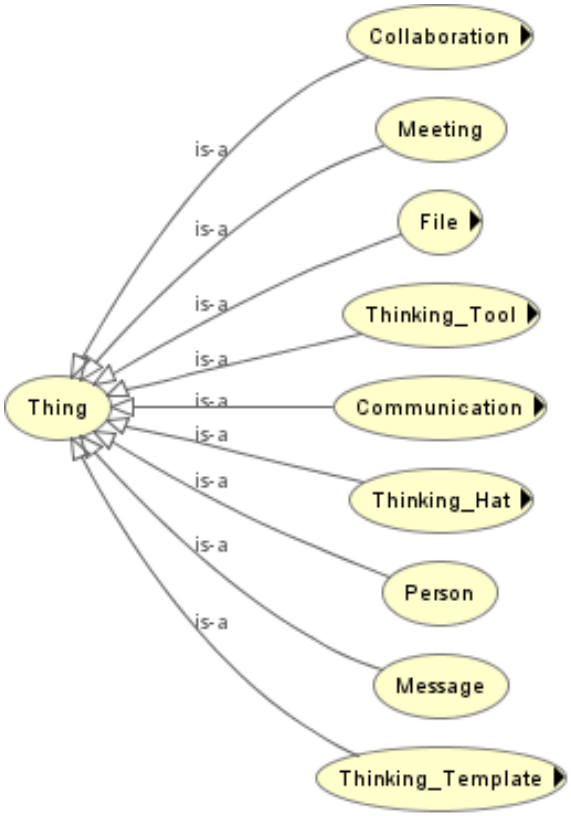
За повторно додавање на нов семантички концепт, т.е. за поврзување на WSDL елементот со даден концепт од некоја од принудените онтологии, се повторува претходно опишаната постапка за додавање на референца.

Со помош на оваа веб базирана апликација, се овозможува анотирање на кој било веб сервис, со користење на SAWSDL рамката, со цел да се изгради репозиториумот од семантички веб сервиси, од кој системот понатаму ќе може да лоцира, селектира и извршува веб сервиси, врз база на барањата на корисниците, специфицирани преку онтологија.

5.4. Практична примена

За да ги согледаме придобивките од еден ваков систем, кој овозможува онтологиите и останатите технологии на Семантичкиот Веб самостојно, без дополнителен код, да се искористат за креирање на семантички апликации, ќе разгледаме еден практичен пример.

Да претпоставиме дека ни е потребна апликација која ќе се користи за управување со состаноци. Со цел таа да биде реализирана со технологиите на Семантичкиот Веб, потребно е како податочен модел да се користи онтологија. За овој пример, ја креираме и користиме онтологијата MeetingsOntology [4] (слика 5-12).



Слика 5-12: Дел од класите од MeetingsOntology онтологијата

Користењето на онтологија како податочен модел овозможува податоците со кои работи апликацијата, претставени во RDF формат, да се придржуваат до моделот дефиниран во онтологијата, преку дефиницијата на класите, релациите и нивните ограничувања.

Во една апликација од овој тип постои потреба од дефинирање на корисничко сценарио во кое корисникот креира нов состанок. На корисникот на апликацијата треба да му се овозможи додавање на учесници на состанокот, одбирање ден и час во кој се закажува состанокот. За тоа кои точно податоци треба корисникот да ги специфицира при креирањето на нов состанок, може да се искористи онтологијата, односно дефиницијата на класата Meeting и релациите кои се поврзани со неа. Дел од релациите поврзани со класата Meeting се participants, initiated_by, subject, agenda, conclusion, date, start_time и end_time:

```
<owl:ObjectProperty rdf:ID="participants">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range rdf:resource="#Group"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="has_member">
    <rdfs:domain rdf:resource="#Group"/>
    <rdfs:range rdf:resource="#Person"/>
```

```
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#initiated_by">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range rdf:resource="#Person"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="subject">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="agenda">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="conclusion">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
    <rdfs:domain rdf:resource="#Meeting"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="date">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#date"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="start-time">
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
    <rdfs:domain rdf:resource="#Meeting"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="end-time">
    <rdfs:domain rdf:resource="#Meeting"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#time"/>
</owl:DatatypeProperty>
```

Но, во вакво корисничко сценарио мора да се внимава на следното ограничување: на корисникот треба да му се понудат само оние потенцијални учесници кои на избраниот ден во конкретниот термин се слободни, односно не се дел од некој друг состанок. Имплементацијата на ова ограничување е тривијална, доколку се напише дополнителен код во апликацијата. Но, ако сакаме да имаме апликација која целосно е изградена само со технологиите на Семантичкиот Веб, односно апликација која онтологиите ги користи не само како податочен модел, туку и за додавање функционалност во самата апликација, тогаш може да се искористи предложениот систем за проширување на онтологиите.

За да го постигнеме ова, потребно е во рамките на MeetingsOntology онтологијата да се креира класа која го дефинира корисничкото сценарио, односно класа која ги содржи податоците кои корисникот треба да ги пополни во формата за креирање и закажување на нов состанок. На одреден начин, оваа класа претставува модел за дефинирање на содржината на графичкиот интерфејс кој апликацијата треба да му го прикаже на корисникот, за конкретното сценарио.

Прво ја вклучуваме functionalOWL онтологијата во MeetingsOntology, како именски простор со префиксот fowl:

```
...
<rdf:RDF
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"

xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:fowl="http://e-
tech.feit.ukim.edu.mk/functionalOWL.owl#"
    xmlns="http://www.owl-ontologies.com/MeetingsOntology.owl#"
  xml:base="http://www.owl-ontologies.com/MeetingsOntology.owl">
...
```

Потоа, во онтологијата дефинираме класа за корисничкото сценарио, CreateNewMeeting. Оваа класа, преку дефинираните релации во онтологијата, има листа од учесници на состанокот, наслов и агенда на состанокот, датум на одржување, почетно време и крајно време. Поради тоа што оваа класа кон себе ги има поврзано истите релации како и класата Meeting, таа може да се дефинира како класа наследена од Meeting класата:

```
<owl:Class rdf:ID="CreateNewMeeting">
      <rdfs:subClassOf rdf:resource="#Meeting"/>
</owl:Class>
```

За да на корисникот му се прикаже одреден интерфејс за дефинираното корисничко сценарио, потребно е во онтологијата, односно базата на знаење, да се дефинира инстанца од класата CreateNewMeeting. Откако корисникот ќе пополни дел од бараните податоци, инстанцата го добива следниот изглед, во RDF формат:

```
<CreateNewMeeting rdf:ID="UseCaseScenario_01">

    <initiated_by rdf:resource="#Tim_B_L"/>
    <moderated_by rdf:resource="#Tim_B_L"/>

    <participants fowl:hasNonstaticValue
    rdf:resource="#GetAvailableParticipants" />

    <fowl:Service rdf:ID="#GetAvailableParticipants">
```

```
            <hasName>
                GetAvailableParticipantsService
            </hasName>
            <hasInput rdf:resource="#MeetingDate"/>
            <hasInput rdf:resource="#StartTime"/>
            <hasInput rdf:resource="#EndTime"/>
            <hasOutput rdf:resource="#AvailableParticipants"/>
        </owl:Service>

        <Input rdf:about="#MeetingDate">
            <parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#date"/>
            2010-03-28
        </Input>
        <Input rdf:about="#StartTime">
            <parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#time"/>
            12:00:00
        </Input>
        <Input rdf:about="#EndTime">
            <parameterType
    rdf:datatype="http://www.w3.org/2001/XMLSchema#time"/>
            14:00:00
        </Input>
        <Output rdf:about="#AvailableParticipants">
            <parameterType rdf:resource="#Group"/>
        </Output>

        <agenda
        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
          1. Review of the Rules Interchange Format (RIF).</agenda>

        <subject
        rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
          Rules Interchange Format (RIF).</subject>

        <date rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
          2010-03-28 </date>

         <start-time
         rdf:datatype="http://www.w3.org/2001/XMLSchema#time">
          12:00:00 </start-time>

        <end-time
        rdf:datatype="http://www.w3.org/2001/XMLSchema#time">
          14:00:00 </end-time>

    </CreateNewMeeting>
```

Како што се гледа од инстанцата, корисникот ги пополнил податоците за тоа кој е иницијатор на состанокот, кој ќе биде модератор на истиот, ги специфицирал темата и агендата на состанокот, датумот на одржување, како и часот на почетокот и крајот на состанокот.

Во означениот дел од инстанцата, листата на потенцијални учесници на состанокот од која корисникот може да избира учесници, е дефинирана како нестатичка вредност. Со користење на релацијата hasNonstaticValue, дефинирана во онтологијата functionalOWL, а референцирана преку именскиот простор fowl, вредноста на релацијата participants се добива преку сервис кој има три влезни параметри и еден излезен параметар. Еден од влезните параметри е од тип *датум* и другите два се од тип *време*. Излезниот параметар е инстанца од класата Group дефинирана во MeetingsOntology онтологијата, а претставува група од инстанци од класата Person. Според тоа, типот на излезниот параметар на сервисот е група од луѓе, односно потенцијални учесници на состанокот.

Кога системот ќе добие ваква онтологија, односно база на знаење, се обидува во неа да ги лоцира сите нестатички податоци, односно сите инстанци на класата fowl:Service. За оваа функционалност е задолжена класата FunctionalOWLEngine од системот. Со помош на оваа класа, откако системот ќе ги лоцира сите инстанци од класата fowl:Service, ги наоѓа влезните параметри заедно со нивниот тип, како и излезниот параметар и неговиот тип.

Преку FunctionalOWLEngine класата се упатува барање до методот findCompatibleService, дефиниран во ServiceEngine делот од системот, кој врз база на информациите за типовите на влезните параметри и излезниот параметар ќе се обиде да пронајде семантички веб сервис кој може да одговори на барањето, односно семантички веб сервис кој може да врати листа од потенцијални учесници на состанокот, кои во одредената временска рамка, на соодветниот ден, се слободни.

Како дел од развојот на апликацијата, претпоставуваме дека веќе имаме класични веб сервиси кои извршуваат дел од потребните функционалности за апликацијата. Претходно, со користење на алатката за семантичка анотација на веб сервиси, претставена во делот 5.3, овие сервиси се семантички анотирани. На тој начин, тие имаат точно дефиниран семантички тип на влезните параметри и излезниот параметар. Сите семантички веб сервиси со кои располагаме, се дел од репозиториумот на сервиси со кои работи системот.

Врз база на поставеното барање, ServiceEngine класата ќе го пребара репозиториумот и според алгоритмот претставен во делот 5.2.2, ќе се обиде да најде кандидат семантички веб сервиси. Доколку пронајде такви сервиси, врз база на коефициентот на погодност ќе го селектира најсоодветниот семантички веб сервис или композиција од семантички веб сервиси. Потоа, системот е задолжен да го повика овој сервис или композиција од сервиси, со податоците кои се дефинирани во самата база на знаење. Резултатот се враќа назад во онтологијата, односно базата на знаење.

Да претпоставиме дека во репозиториумот на семантички веб сервиси нема еден единствен веб сервис кој ги прима сите три типа на податоци на влез, а го враќа бараниот податок. Наместо тоа, во репозиториумот постојат два веб сервиса, од кои едниот, GetAvailableParticipants, како влезни параметри прима еден податок од семантички тип *датум*, еден податок од семантички тип *време* и уште еден податок од семантички тип *времетраење*, а излезниот параметар на сервисот е од бараниот семантички тип - Group. Вториот семантички веб сервис, GetDuration, има два влезни параметри од семантички тип *време*, а како резултат враќа параметар од типот *времетраење*.

Според алгоритмот, системот прво ќе го лоцира семантичкиот веб сервис GetAvailableParticipants, затоа што семантичкиот тип на неговиот излезен параметар се совпаѓа со семантичкиот тип на бараниот податок. По анализа на бројот и типовите на влезни параметри на GetAvailableParticipants, системот ќе заклучи дека само два од трите параметри се совпаѓаат со влезните параметри од барањето. Тоа се параметрите од семантички тип *датум* и *време*. Бидејќи во репозиториумот нема други веб сервиси чиј што тип на излезниот параметар се совпаѓа со типот на бараниот податок, системот ќе го земе семантичкиот веб сервис GetAvailableParticipants како единствено потенцијално решение и ќе се обиде да го најде преостанатиот влезен податок, од тип *времетраење*, кој му се потребен за да го обезбеди бараниот податок на излез.

Така, системот ќе продолжи со алгоритмот, сега барајќи сервиси кои како излез враќаат податок од типот *времетраење*, поради тоа што тој податок е недостапен за да сервисот GetAvailableParticipants го врати бараниот податок. Како потенцијален сервис ќе го лоцира сервисот GetDuration, затоа што семантичкиот тип на излезниот параметар е *времетраење*. Двата влезни параметри на овој сервис се од семантички тип *време*, што одговара на типовите на влезните параметри специфицирани во барањето. Со тоа барањето на семантички веб сервиси завршува, затоа што системот заклучува дека може да ги добие сите барани податоци, со користење само на влезните параметри специфицирани во барањето на корисникот.

Како резултат од пребарувањето на соодветни семантички веб сервиси, системот пронајде композиција на два семантички веб сервиси, кои доколку се извршат во соодветниот редослед, го даваат бараниот податок.

Следна задача на системот е извршување на сервисите, за да се добие бараниот податок. Според композицијата, прво треба да се повика семантичкиот веб сервис GetDuration, а потоа, со користење на резултатот од овој сервис, да се повика GetAvailableParticipants. Податоците кои треба да се проследат до семантичките веб сервиси, системот ги чита директно од базата на знаење.

Со вчитаните вредности од типот *време*, кои во примерот се 12:00:00 и 14:00:00, системот го повикува веб сервисот GetDuration. Резултатот од повикот на веб сервисот е податок со вредност 02:00:00 и семантички тип *времетраење*.

Овој податок, заедно со вчитаниот податок 12:00:00 од тип *време* и вчитаниот податок 2010-03-28 од тип *датум*, се испраќа при повикот на веб сервисот GetAvailableParticipants. Како резултат на повикот се добива податок од типот Group, кој ги содржи сите потенцијални учесници на состанокот, кои на 2010-03-28, почнувајќи од 12:00:00 часот, па во наредните 02:00:00 часа се слободни, односно не присуствуваат на некој друг состанок или некоја друга активност во рамките на компанијата. Овој податок од страна на системот се враќа назад во базата на знаење:

```
<CreateNewMeeting rdf:ID="UseCaseScenario_01">

    <initiated_by rdf:resource="#Tim_B_L"/>
    <moderated_by rdf:resource="#Tim_B_L"/>

    <participants>
       <Group>
            <has_Member rdf:resource="Clark_C">
            <has_Member rdf:resource="Bill_G">
            <has_Member rdf:resource="Steve_J">
            <has_Member rdf:resource="Bruce_W">
       </Group>
    </participants>

    ...

    <agenda
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      1. Review of the Rules Interchange Format (RIF).</agenda>

    <subject
    rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      Rules Interchange Format (RIF).</subject>

    <date rdf:datatype="http://www.w3.org/2001/XMLSchema#date"
      2010-03-28 </date>

     <start-time
     rdf:datatype="http://www.w3.org/2001/XMLSchema#time">
      12:00:00 </start-time>

    <end-time
    rdf:datatype="http://www.w3.org/2001/XMLSchema#time">
      14:00:00 </end-time>

</CreateNewMeeting>
```

На овој начин, со дефинирање на инстанци од functionalOWL онтологијата во рамките на онтологијата MeetingsOntology, апликацијата е проширена со дополнителна функционалност над податочниот модел, без притоа да се пишува

дополнителен код во некој програмски јазик. Проширената онтологија, освен што го дефинира податочниот модел на апликацијата, дефинира и функционалности кои овозможуваат додавање на нестатични и специфични вредности во рамките на базата на знаење со која работи апликацијата.

6. Заклучок

Идејата за Семантичкиот Веб е навистина голема, па понекогаш се добива впечаток дека можеби е и премногу амбициозна. Нејзината реализација денеска е некаде на половина [66]. Она што денеска де факто постои и што сé повеќе се користи, се елементите кои се основата на Семантичкиот Веб. Стандардизацијата на XML, RDF, RDFS, OWL и SPARQL, а и напорите за стандардизација и на јазикот за изразување правила RIF, делува охрабрувачки и дозволува истражувачките проекти и имплементациите на технологиите на Семантичкиот Веб да се фокусираат на погорните слоеви од архитектурата. Според некои истражувања, сé повеќе компании ги вградуваат препораките и стандардите на W3C во своите производи [13]. Според тоа, идеите и технологиите на Семантичкиот Веб се заживеани и се користат не само во научни, туку и во комерцијални цели [67][4].

Со цел да се оствари визијата на Семантичкиот Веб, денешните истражувања се фокусираат на погорните слоеви од архитектурата. Откако тие ќе се стандардизираат, останува вниманието на истражувачите и нивните партнери од индустријата да се сврти кон за сега недопрените области - доказот и довербата, кои ќе играат клучна улога во широкото прифаќање на Семантичкиот Веб од страна на корисниците на постоечкиот Веб.

Додека се чека на стандардизација и имплементација на останатите слоеви од архитектурата на Семантичкиот Веб, развивачите на софтвер се соочуваат со ограничувања кои моменталните семантички технологии не можат да ги надминат. Едно од тие ограничувања е и ограниченоста на онтологиите и базите на знаење да работат добро само со податоци кои ретко ја менуваат својата вредност, односно податоци кои можеме да ги класифицираме како статички.

Иако онтологиите се користат за дефинирање на податочниот модел на апликациите, истите можат да се прошират со цел да опфатат поширок спектар од семантичките апликации, односно да внесат функционалност во нив.

Идејата претставена во овој магистерски труд опфаќа дефинирање на рамка за проширување на онтологиите, со цел да се постигне поголема функционална моќ на онтологиите. На тој начин, онтологиите ќе можат да се искористат за развој на апликации во кои тие нема да го дефинираат само податочниот модел и базата на знаење, туку ќе внесат и одредена функционалност во самите апликации. Така ќе се избегне потребата од пишување дополнителен код во некој програмски јазик за дефинирање на функционалностите, динамиката и ограничувањата на семантичките апликации; самите технологии на Семантичкиот Веб стануваат доволни за дефинирање на целосно функционална семантичка апликација.

За да го постигнеме проширувањето, дефиниравме онтологија наречена funtionalOWL, која има за цел да воведе т.н. сервисни вредности во онтологиите, односно на ресурсите од одредена онтологија, а со тоа и на инстанците од одредена база на податоци, да им придружи вредности кои се добиваат како резултат од повикување на одреден сервис. За да може да се изведе проширувањето, развивме и систем кој може да се справи со наоѓањето на соодветен семантички веб сервис или композиција од семантички веб сервиси, и повикување на сервисот или композицијата, со цел да се добие бараниот податок.

Главните придобивки кои ги воочивме со имплементацијата на проширувањето на онтологиите и со имплементацијата на системот, се во насока на добивање поголема експресивна и функционална моќ на онтологиите, кои сега можат да се користат не само како податочен модел во рамките на семантичките апликации, туку и како функционален модел, кој на апликациите им дава поголема функционална моќ, без притоа да има потреба од пишување на дополнителен код. Како што беше илустрирано во примерот од дел 5.4, сé што беше потребно развивачот на апликацијата за работа со состаноци да направи, за да обезбеди интерфејс во кој на корисникот му се понудени специфични податоци (само лицата кои се слободни во конкретниот термин), беше да напише дефиниција на сервис во рамките на базата на знаење и истиот да го поврзе со ресурсот participants. Со помош на системот, се лоцираше потребниот семантички веб сервис, односно композиција од семантички веб сервиси, која го врати потребниот резултат и истиот го запиша назад во базата на знаење. Со тоа беше илустрирана зголемената функционална моќ на проширените онтологии, кои можат да се користат како градбени единки на семантичките апликации во кои ќе покриваат поголем дел од потребните функционалности.

Во овој магистерски труд е направен детален преглед на постоечките технологии кои ги сочинуваат различните нивоа во архитектурата на

Семантичкиот Веб, како и детален преглед и анализа на постоечките механизми и апликации кои се користат за анотација на веб сервисите и нивна трансформација во семантички веб сервиси. Потоа, развиена е веб апликација за семантичка анотација на веб сервиси, со користење на .NET платформата и SAWSDL стандардот за семантичка анотација на веб сервиси. Даден е предлог за проширување на онтологиите со функционалности, кои овозможуваат употреба на специфични податоци во рамките на семантичките апликации и нивните бази на знаење. Дизајнирана е и изработена онтологија која ги обезбедува предложените проширувања на концептот на онтологии, со која се овозможува дефинирање на вредност на одреден ентитет од онтологијата, преку соодветна релација, како *нестатички* тип, добиен од сервис. Во трудот е даден и предлог за систем кој ги обработува проширените онтологии и кој за соодветно барање дефинирано во онтологијата, е способен да го пронајде најсоодветниот семантички веб сервис, односно најсоодветната композиција од семантички веб сервиси. Предложен е алгоритам за оценување на погодноста на пронајдените кандидат семантички веб сервиси, односно кандидат композиции, за извршување на поставеното барање. Направена е и имплементација на алгоритмот во Java програмскиот јазик. Направена е имплементација на предложениот систем со користење на Jena рамката за развој на семантички апликации.

Како идна работа е планирано проширување на системот, со цел да се постигне подобро лоцирање и рангирање на семантичките веб сервиси кои можат да го обезбедат бараниот податок. Во моменталната изведба на системот, семантичките веб сервиси се пребаруваат единствено врз база на семантичкиот тип на влезните параметри и излезниот параметар. Овој пристап може да се прошири со барање на совпаѓање и на други семантички елементи кај семантичките веб сервиси, како што е функционалниот опис, на пример. Тоа би барало и одредени измени во functionalOWL онтологијата, со цел развивачите на апликации да можат подетално да специфицираат каков сервис бараат, преку базата на знаење.

Системот работи со семантички веб сервиси опишани со SAWSDL јазикот, бидејќи SAWSDL е јазик препорачан од страна на W3C конзорциумот. Планирано е проширување на системот со цел да се обезбеди работа и со семантички веб сервиси анотирани со други јазици, како што се OWL-S, WSMO и WSDL-S.

Како идна работа е планирано и проширување на репозиториумот на семантички веб сервиси со кои работи системот, од репозиториум со локални семантички веб сервиси, во репозиториум кој во себе може да вклучува и други репозиториуми кои се достапни на Вебот. На тој начин би се овозможила работа на системот со многу поголем број семантички веб сервиси, со што би се зголемила употребливоста и ефикасноста на целиот систем.

Со до сега реализираното и со планираните проширувања веруваме дека придонесуваме кон надминувањето на дел од ограничувањата со кои се среќаваме при развојот на апликации кои се базираат на технологиите на Семантичкиот Веб, а со тоа и кон побрзо остварување на визијата на сер Тим Бернерс-Ли, за креирање и проширување на Семантичкиот Веб, како мрежа од прецизно дефинирани информации која ќе овозможи автоматизација на голем дел од работите кои корисниците на Вебот денес ги извршуваат мануелно.

7. Референци

[1] Berners-Lee, T., J. Hendler, O. Lassila, "The Semantic Web", Scientific American, May 2001.

[2] W3C Semantic Web Activity, http://www.w3.org/2001/sw/

[3] Interview with Tim Berners-Lee, Business Week, April 2007.

[4] Nikov, A., Jovanovik, M., Stojanov, R., Petkovski, M., Trajanov, D., "Use of Semantic Web Technologies for Meeting Management Software Development", ICT Innovations 2009, September 2009.

[5] World Wide Web Consortium (W3C), http://www.w3.org/

[6] XML, Extensible Markup Language, http://www.w3.org/XML/

[7] RDF, Resource Description Framework, http://www.w3.org/RDF/

[8] RDFS, RDF Schema, http://www.w3.org/TR/rdf-schema/

[9] OWL, Web Ontology Language, http://www.w3.org/TR/owl-features/

[10] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/

[11] RDF Vocabulary Description Language 1.0: RDF Schema, http://www.w3.org/TR/rdf-schema/

[12] RIF Use Cases and Requirements, W3C Working Draft 18 December 2008, http://www.w3.org/TR/rif-ucr/

[13] Janev, V., "Primena semantičkog jezika SPARQL za pretraživanje u okviru Semantičkog Web-a", Institut Mihajlo Pupin, Volgina 15, 11060, Belgrade.

[14] "Resource Description Framework (RDF): Concepts and Abstract Syntax", W3C Recommendation, February 10, 2004, Graham Klyne, Jeremy J. Carroll, eds.

[15] Hebeler, J., Fisher, M., Blace, R., Perez-Lopez, A.,"Semantic Web Programming", Wiley 2009.

[16] Yu, L., "Introduction to the Semantic Web and Semantic Web Services", Chapman & Hall, 2008.

[17] Colombo, R.M.,"Ontology and the Semantic Web", IOS Press, 2009.

[18] Jena – A Semantic Web Framework for Java, http://www.jena.sourceforge.net

[19] Sesame – An open source framework for storage, inferencing and querying of RDF data, http://www.openrdf.org/

[20] Joseki – A SPARQL Server for Jena, http://www.joseki.org/

[21] 4Suite – An open-source platform for XML and RDF processing, http://www.4suite.org/

[22] The OWL API, http://owlapi.sourceforge.net/

[23] RAP – RDF API, http://sourceforge.net/projects/rdfapi-php/

[24] Redland RDF Libraries, http://librdf.org/

[25] LinqToRdf – A Semantic Web framework for .NET, http://code.google.com/p/linqtordf/

[26] Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., "Jena: Implementing the Semantic Web", WWW Alt. '04: Proceedings of the 13th international World Wide Web conference on Alternate track papers \& posters, page 74--83. ACM Press, 2004.

[27] Antoniou, G., and van Harmelen, F., "A Semantic Web Primer", The MIT Press, 2008.

[28] Pellet: OWL 2 Reasoner for Java, http://clarkparsia.com/pellet/

[29] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[30] SOAP Version 1.2, W3C Recommendation (Second Edition) 27 April 2007, http://www.w3.org/TR/soap/

[31] UDDI Specifications, http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm

[32] North American Industry Classification System (NAICS), http://www.census.gov/eos/www/naics/

[33] Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J. A., and Zien, J. Y. 2003. "SemTag and seeker: bootstrapping the semantic web via automated semantic annotation", Proceedings of the 12th international Conference on World Wide Web (Budapest, Hungary, May 20 - 24, 2003). WWW '03. ACM, New York, NY, 178-186.

[34] Heflin, J., Hendler, J., Luke, S. "SHOE: A Knowledge Representation Language for Internet Applications", *Techical Report*, UM Computer Science Department, 1999.

[35] Kogut, P., Holmes, W., "AeroDAML: Applying Information Extraction to Generate DAML Annotations from Web Pages", First International Conference on Knowledge Capture (K-CAP 2001). Workshop on Knowledge Markup and Semantic Annotation, 2001.

[36] Hammond, B., Sheth, A., Kochut, K., and Semagix Inc, "Semantic Enhancement Engine: A Modular Document Enhancement Platform for Semantic Applications over Heterogeneous Content", Real World Semantic Web Applications, 2002.

[37] Staab, S., Maedche, E., and Handschuh, S., "An annotation framework for the semantic web", Proceedings of the First Workshop on Multimedia Annotation, 2001.

[38] Goble, C., Bechhofer, S., De Roure, D., Carr, L., Hall, W., "Conceptual Open Hypermedia = The Semantic Web?", Proceedings of the WWW2001, Semantic Web Workshop, Hongkong, 2001.

[39] Handschuh, S., Staab, S., "Authoring and annotation of web pages in CREAM", International World Wide Web Conference, Proceedings of the 11th international conference on World Wide Web, 2002.

[40] Handschuh, S., Staab, S., Ciravegna, F., "S-CREAM – Semi-automatic CREAtion of Metadata", Springer Berlin / Heidelberg, 2002.

[41] Kahan, J., Koivunen, M.R., Prud'Hommeaux, E., Swick, R.R., "Annotea: An open RDF infrastructure for shared Web annotations", Computer Networks Vol. 39, no. 5, pp. 589-608. 5 August 2002.

[42] Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., and Kirilov, A., "KIM – a semantic platform for information extraction and retrieval", Natural Language Engineering 10, 3-4, September 2004, 375-392.

[43] Guha, R., McCool, R., "Tap: Towards a web of data", http://tap.stanford.edu/

[44] OWL-S: Semantic Markup for Web Services, W3C Member Submission, 22 November 2004, http://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/

[45] WSMO: Web Service Modeling Ontology, http://www.wsmo.org/

[46] Web Service Semantics – WSDL-S, W3C Member Submission, 7 November 2005, http://www.w3.org/Submission/WSDL-S/

[47] Semantic Annotations for WSDL and XML Schema, W3C Recommendation, 28 August 2007, http://www.w3.org/TR/sawsdl/

[48] WSML – Web Service Modeling Language, http://www.wsmo.org/wsml/

[49] WSMX – Web Service Modelling eXecution environment, http://www.wsmx.org/

[50] Fensel, D., Bussler, C.,"The Web Service Modeling Framework WSMF", Electronic Commerce Research and Applications, 2002.

[51] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, W3C Member Submission, 21 May 2004, http://www.w3.org/Submission/SWRL/

[52] Organization for the Advancement of Structured Information Standards (OASIS), http://www.oasis-open.org/

[53] Pan, J. Z., Horrocks, I.. "OWL-E: Extending OWL with Expressive Datatype Expressions", IMG Technical Report, School of Computer Science, the University of Manchester, April 2004.

[54] Bouquet, P., Giunchiglia, F., van Harmelen, F., Serafini, L., Stuckenschmidt, H., "C-OWL: Contextualizing ontologies", Proceedings of the Second International Semantic Web Conference, pages 164-179, October 2003.

[55] METEOR-S: Semantic Web Services and Processes, http://lsdis.cs.uga.edu/projects/meteor-s/

[56] Thompson, H. S., "Functional XML", http://www.ltg.ed.ac.uk/~ht/planx_fx.html

[57] Halpin, H., Thompson, H. S., "One Document to Bind Them: Combining XML, Web Services, and the Semantic Web", Proceedings of the 15th international Conference on World Wide Web (Edinburgh, Scotland, May 23 - 26, 2006).

[58] Halpin, H., "From typed-functional semantic web services to proofs", Proceedings of the 5th International semantic web conference, ISWC 2006, Athens, GA, USA, November 5-9, 2006.

[59] Domingue, J., Cabral, L., Galizia, S., Tanasescu, V., Gugliotta, A., Norton, B., Pedrinaci, C., "IRS-III: A Broker-based Approach to Semantic Web Services", Journal of Web Semantics, 6, 2, pp. 109-132, Elsevier, 2008.

[60] Cabral, L., Domingue, J., Galizia, S., Gugliotta, A., Norton, B., Tanasescu, V., Pedrinaci, C., "IRS-III: A Broker for Semantic Web Services based Applications", The 5th International Semantic Web Conference (ISWC 2006), Athens, GA, USA, 2006.

[61] Internet Reasoning Service III, http://technologies.kmi.open.ac.uk/irs/

[62] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D., "Web Service Modeling Ontology", Applied Ontology, 1(1): 77 - 106, 2005.

[63] Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Bussler, C., Fensel, D., "WWW: WSMO, WSML, and WSMX in a nutshell", Proceedings of the first Asian Semantic Web Conference (ASWC 2006), Beijing, China, September 3-7, 2006.

[64] WSMO Studio, http://www.wsmo.org/

[65] SAWSDL4J - a Java API for the SAWSDL specification, http://knoesis.wright.edu/opensource/sawsdl4j/

[66] Јовановиќ, М., Трајанов, Д., „Преглед на проектите за развој на технологиите на Семантичкиот Веб", Зборник на трудови ЕТАИ 2009, септември 2009.

[67] Cardoso, J., "The Semantic Web Vision: Where are We?", IEEE Intelligent Systems, September/October 2007, pp.22-26, 2007.