UNIVERSITATEA **POLITEHNICA** DIN BUCUREŞTI
Facultatea de Automatică □i Calculatoare
Departamentul de Calculatoare

REZUMATUL TEZEI DE DOCTORAT

Interoperabilitatea semantică a aplicațiilor din domeniul medical bazate pe agenți software şi servicii web

Semantic Interoperability in Healthcare Systems Based on Software Agents and Web Services

Autor:
Ing. Sorin-Alexandru Cristescu

Conducător ştiințific:
Prof.dr.ing. Florica Moldoveanu

Bucureşti, 2012

Table of Contents

1. INTRODUCERE

1.1 Motivatie

Institutiile medicale din intreaga lume au operat si continua in cea mai mare masura sa opereze ca entitati separate, fara a comunica intre ele. Acest lucru are un impact puternic asupra calitatii actului medical.
Sa ne gandim de exemplu la dosarul medical al pacientului, care in momentul de fata consta – in cele mai multe cazuri – in documente tiparite pe hartie si disparate intre spitalele unde pacientul a fost la un moment dat investigat. In mod normal, pentru un anumit diagnostic sau tratament, un medic are nevoie de istoria medicala a pacientului – de exemplu un anumit medicament nu poate fi prescris pentru ca pacientul a manifestat la un moment dat o alergie la anumite substante, sau se recomanda operatie de cezariana daca pacienta a avut operatie de hernie de disc relative recent.
Un alt exemplu de lipsa a interoperabilitatii il ofera domeniul monitorizarii pacientilor. Un pacient dependent de dializa probabil trebuie sa mearga la spital saptamanal si acolo este conectat la un aparat complex de dializa. Tele-dializa este un concept mai degraba in faza de cercetare.
O persoana care are un atac de cord in timp ce se afla pe strada sau in casa, are sanse de supravietuire daca poate alerta serviciul de urgenta/ambulanta si daca acesta ajunge la timp si ofera primul-ajutor necesar cazului respectiv.

In timp ce medicina progreseaza si informatizarea institutiilor medicale capata amploare peste tot in lume, intrepatrunderea celor doua ofera beneficii evidente, avand ca scop final imbunatatirea actului medical si prin urmare a calitatii vietii. In zilele noastre vorbim despre Electronic Patient Record (EPR), spre deosebire de informatiile fragmentate cu care eram obisnuiti pana acum, inovam in domeniul tele-dializei, astfel incat un pacient dependent de dializa va putea fi tratat acasa, avem protocoale proprietar pentru monitozirarea la distanta a persoanelor varstnice, spitalele si alte institutii medicale, precum si firmele de asigurari comunica mai eficient intre ele, etc.

Cu toate astea suntem inca departe de un flux optim al operatiilor in domeniul medical. Cum ar fi daca un pacient in varsta care lesina intr-un parc sau in propria casa ar fi monitorizat de o entitate software care alerteaza in mod autonom serviciul electronic de urgenta, care la randul lui informeaza cel mai apropiat serviciu de ambulante, pe baza simptomelor pacientului si a coordonatelor geografice exacte ? Nici o alta interventie umana nu ar fi necesara pana in momentul cand personalul ambulantei este alertat si se grabesc sa ajunga la pacient. Pentru ca simptomele pacientului sunt cunoscute personalului ambulantei, acestia pot de exemplu sa aduca un defribilator care poate salva viata pacientului. Mai mult decat atat, in timp ce pacientul este transportat catre spital, receptia spitalului este alertata electronic in legatura cu starea pacientului, prin urmare specialistul potrivit este pregatit sa preia cazul. Exemplele de interoperabilitate

electronica pot continua cu domeniul tele-dializei, unde pacientii sunt monotorizati la distanta, sau cu EPR, unde diverse entitati software aduc si pun cap la cap intregul dosar electronic al pacientului, spre a fi analizat de un doctor in timpul unei consultatii.

Acest document propune o arhitectura bazata pe agenti software si servicii web, care au ca scop oferirea interoperabilitatii semantice intre entitatile din domeniul medical, astfel incat comunicarea intre aceste entitati sa fie mai eficienta si sa cuprinda cat mai multi actori din domeniul medical (spitale, doctori, pacienti, firme de asigurare, etc.). Ca nivel de baza al acestei arhitecturi, propunem folosirea standardelor existente si a ontologiilor.

1.2 Contributii originale

1. In aceasta lucrare propunem o platforma pentru domeniul medical bazata pe agenti software si servicii web, cu alte cuvinte ceea ce am putea numi arhitecturi orientate pe agenti (AOA – Agent-Oriented Architectures) si arhitecturi orientate pe servicii (SOA – Service Oriented Architectures). Agentii si serviciile comunica folosind ontologii[1] si standarde existente, precum HL7, DICOM, FIPA, SOAP, SAWSDL, OWL, etc. In felul acesta urmarim sa aliniem institutiile medicale si departamentele lor de software prin aceasta noua platforma, cu scopul obtinerii interoperabilitatii semantice reale intre diverse institutii din domeniul medical. Asa cum va rezulta pe parcursul tezei, cele mai semnificative eforturi pentru ca o institutie sa adopte sistemul propus, sunt dedicate adnotarii agentilor si serviciilor si eventual crearii de asocieri intre ontologii si inregistrarea acestor asocieri.

2. Propunem aici adnotarea descrierilor agentilor cu *Inputs*, *Outputs*, *Preconditions* si *Effects* (IOPE), fiecare adnotare fiind exprimata cu ajutorul standardului SAWSDL [16], propus de W3C pentru adnotarea serviciilor web. Pentru ca acest lucru sa poata avea loc, avem nevoie de o extensie a standardului FIPA [17] pentru agenti, anume Directory Facilitator (DF) trebuie extins cu IOPE adnotate semantic.

3. Cu serviciile si agentii adnotati semantic, propunem ca si coloana vertebrala pentru sistemul nostru un motor de cautare semantic, care functioneaza atat ca un index pentru servicii si agenti, cat si ca vehicul folosit pentru descoperirea lor semantica. Am numit acest motor de cautare *Semmed*, care sugereaza ca este bazat pe semantica si ca este destinat domeniului medical (de fapt este mai general decat atat, dar din cauza complexitatilor maparii ontologiilor, este mai realist a fi aplicat in anumite domenii, cu ontologii bine definite; acest lucru va fi detaliat mai tarziu pe parcursul lucrarii).

[1] Prin *ontologie* intelegem desigur definitia folosita in stiinta calculatoarelor, anume: o ontolgie reprezinta o baza de cunostinte, reprezentate ca si concepte dintr-un anumit domeniu si relatiile dintre ele; poate fi folosita pentru a rationa in legatura cu entitatile din acel domeniu si de asemenea pentru a descrie domeniul.

4. Algoritmul de cautare este si el o contributie originala. Bazat pe conceptul cunoscut de distanta semantica, algoritmul incearca o potrivire semantica in ordine crescatoare a complexitatii, in speranta minimizarii complexitatii totale.

5. Propunem folosirea unor structuri de date optimizate de catre componentele motorului de cautare, astfel incat complexitatea algoritmului de cautare sa fie minimizata. Mai precis, cautam sa minimizam accesele la disc, componenta care degradeaza cel mai mult performanta.

6. De asemenea, propunem extinderea platformei noastre cu servicii web RESTful si explicam avantajele care rezulta de aici. Acest lucru implica adnotarea semantica a descrierilor WADL ale serviciilor RESTful.

1.3 Publicatii stiintifice in conexiune cu teza

Cristescu S., Moldoveanu F. *An Agent-Oriented and Service-Oriented Architecture in Medicine*. In: Annals of DAAAM for 2009 & Proceedings of the 20th International DAAAM Symposium, Published by DAAAM International, Vienna, Austria, 2009, pp. 1767-1769

Cristescu S., Moldoveanu F. *Patient Monitoring with Agents and Semantic Web.* In: Annals of DAAAM for 2010 & Proceedings of the 21th International DAAAM Symposium, Published by DAAAM International, Vienna, Austria, 2010, pp. 809-811

Cristescu S. *High Level Design of a Semantic Search Engine.* In: Annals of DAAAM for 2011 & Proceedings of the 22th International DAAAM Symposium, Published by DAAAM International, Vienna, Austria, 2011, pp. 567-568

Cristescu S., Moldoveanu F. *A Semantic Search Engine Implemented with Open Source*. In: International Conference on Innovative Technologies, IN-TECH 2012

Cristescu S., Moldoveanu F. *An Agent-Oriented and Service-Oriented Architecture in Medicine*, UPB Scientific Bulletin, ISSN 1454-234x, seria C, vol. 75, nr. 1/2013

Cristescu S., *Semantic Search and Invocation of Web Services*, Journal of Control Engineering and Applied Informatics, 2012 (submitted)

2. FUNDAMENTE TEORETICE

2.1 Sisteme multiagent

Conform unei definitii agreate de publicul larg:
"In stiinta calculatoarelor, un agent software este un program care actioneaza in numele unui utilizator sau a altui program in virtutea relatiei de <<agentie>>, care deriveaza di cuvantul latin <<agere>> (a face): o intelegere pentru a actiona in numele cuiva. Astfel de <<actiune in numele cuiva>> implica autoritatea de a decide care anume actiuni sunt potrivite."

Un alt concept important este cel de sistem multiagent (MAS = Multiagent System), definit ca:
" …. un sistem alcatuit din mai multi agenti inteligenti care interactioneaza intre ei. Sistemele multiagent pot fi folosite pentru a rezolva probleme dificile sau imposibil de rezolvat de catre un agent individual sau de catre un sistem monolitic."

In aceasta lucrare ne referim la un sistem alcatuit din agenti si servicii web care comunica intre ei in mod transparent (mai precis, fara a sti ca se adreseaza unui agent sau unui serviciu). Ne concentram pe aspectele legate de inregistrarea, descoperirea semantica si invocarea agentilor si serviciilor mai degraba decat pe cooperarea intre ei pentru a rezolva diferite probleme. Cu toate acestea, comunicarea este piatra de temelie pentru orice evolutii ulterioare, fara de care un sistem multiagent nu poate exista. Peste aceasta putem adauga algoritmi complecsi. Sa presupunem ca trei agenti comunica la un moment dat: agentul pacientului, agentul doctorului specialist si agentul firmei de asigurari. Impreuna ajung la un echilibru Nash [18], intr-o problema tipica pentru teoria jocurilor in care scopul este de a determina cea mai buna decizie cu privire la calea de urmat: trimiterea pacientului acasa cu tratamentul prescris sau spitalizarea acestuia si daca da, pentru cate zile. Acesta este doar un exemplu de problema rezolvata de un MAS si in viata de zi cu zi sunt multe alte probleme de rezolvat, de la stabilirea unei programari pentru pacient cu medicul sau de familie, pana la colectarea fragmentelor disparate din fisa pacientului si prezentarea lor in mod unitar unui specialist spre analiza. Mai multe detalii asupra echilibrului Nash sunt in afara domeniului acestui document, dar cu siguranta acesta este un subiect pentru cercetarea ulterioara.
In afara de medicina, sistemele multiagent sunt subiect de cercetare in multe domenii ale tehnicii, cele mai notabile fiind managementul traficului si logistica.
Exista de asemenea mai multe platforme MAS care permit utilizatorilor dezvoltarea aplicatiilor multiagent. Cea mai cunoscuta este probabil JADE [1], o platforma open source si care respecta standardele FIPA si pe care o folosim si noi in prototipuri. Ultima versiune la momentul de fata, anume 4.1.1, publicata in Noiembrie 2011, suporta sistemul de operare Android, facilitand astfel scenariile tipice sistemelor MAS: agentii software JADE pot fi instalati acum si pe telefoanele mobile.

Este important sa numim aici caracteristicile care fac agentii interesanti si, de fapt, diferiti de alte tipuri de componente software.
Pentru aceasta sa ne imaginam mai intai urmatorul scenariu:
"Domnul Ionescu, un bunic in varsta de saptezeci de ani, se plimba in parc cu nepotelul sau de doi ani. La un moment dat domnului Ionescu i se face rau si isi pierde cunostinta. Nu este nimeni in imprejurime. Din fericire, domnul Ionescu poarta ceasul inteligent, care, prin intermediul unui agent software preinstalat, alerteaza serviciul de urgenta. Mai precis, trimite serviciului de urgenta datele pesonale ale pacientului, simptomele medicale, fisa medicala completa si locatia sa geografica actuala. Cunoscand locatia pacientului, serviciul de urgenta cauta cea mai apropiata ambulanta disponibila, informand agentii dispeceri de ambulante din vecinatate. In mai putin de cinci minute o ambulanta soseste cu defribilatorul pregatit si transporta pacientul la spitalul cel mai apropiat. In drum spre spital agentul ambulantei colecteaza simpotomele pacientului si istoria sa medicala de la agentul pacientului si informeaza agentul receptiei spitalului despre sosirea iminenta a pacientului. Avand simpotomele pacientului si istoria sa medicala, agentul receptiei spitalului cauta un specialist disponibil, care va fi mai tarziu ajutat de agentul sau in diagnoza pacientului. Din acest moment, agentul medicului specialist monitorizeaza starea pacientului prin comunicarea cu agentul acestuia."

2.1.1 Autonomia

Este o caracteristica esentiala, ce permite agentilor sa actioneze in numele utilizatorilor, dar fara interventie din partea acestora. Scenariul de mai sus se bazeaza pe aceasta importanta proprietate a agentilor, pentru ca pacientul poate fi in stare de inconstienta, si totusi agentul sau va alerta serviciul de urgenta.
Agentul "simte" mediul inconjurator – in acest scenariu mediul este format din mici senzori fara fir lipiti pe corpul pacientului; acesi senzori si ceasul "inteligent" monitorizeaza inima si alte organe. La interval de cateva secunde agentul instalat pe ceasul inteligent colecteaza informatie de la senzori, foloseste un motor simplu de inferente pentru a deriva noi reguli si, daca simptomele se deterioreaza, alerteaza serviciul de urgenta. Prototipul nostru foloseste platforma open source JADE pentru sisteme multiagent si motorul de inferente Jena [13].

2.1.2 Mobilitatea

Reprezinta capacitatea agentilor de a transporta algoritmii catre date, executand operatii cu acele date si intorcand rezultate. Aceasta proprietate este utila in special in situatii precum conectivitatea sporadica la Internet si/sau largime de banda mica, adica exact ceea ce se intampla in scenariul de mai sus: domnul Ionescu este in parc, unde conectivitatea la Internet wireless exista, dar este limitata. In acest caz agentul gaseste in mod autonom momentul propice migrarii spre serverul destinatie, unde alerteaza serviciul de urgenta. Astfel, combinatia intre autonomie si mobilitate ajuta in abordarea propice a unui astfel de scenariu.
Asa cum am mentionat mai sus, prototipul nostru foloseste platforma JADE, care suporta agentii mobili care migreaza intre masini sau intre procese pe aceeasi masina. Recent

JADE a introdus suport pentru sistemul de operare Android, permitand astfel telefoanele inteligente sa devine gazde pentru agentii software.

2.1.3 Inteligenta

Este capacitatea unui agent de a invata din experientele anterioare si a-si imbunatati solutiile la problemele pe care le rezolva.
Inteligenta este manifestata fie de catre un agent individual, fie de un MAS. Primul caz este important in situatii precum un agent care ajuta medical specialist sa analizeze imagini medicale. Un astfel de agent ar putea sa invete de exemplu cum sa identifice polipi in intestinul gros si sa notifice specialistul de prezenta si dimensiunea acestora. Cel de-al doilea caz este important in scenariul in care agentii ce compun un MAS invata din experienta si isi imbunatatesc contributia la solutia finala, crescand astfel calitatea solutiei totale.
JADE nu suporta in mod intrinsec agenti inteligenti, dar exista librarii externe pentru asta. Una dintre ele este Jadex, o implementare a modelului BDI (*Beliefs, Desires, Intentions*) – un model cunoscut pentru agentii inteligenti cognitivi.
Desigur, nimic nu ne opreste sa programam inteligenta in agentii nostri construiti cu JADE, folosind oricare dintre tehnicile de invatare: invatarea supervizata, nesupervizata sau ghidata (“reinforcement learning”).
Un exemplu recent de agent inteligent este *Siri* create de compania Apple, instalat pe telefoanele iPhone 4S (si portat pe alte versiuni), care recunoaste vorbirea si chiar invata dialectul/accentul vorbitorului, astfel incat intelege din ce in ce mai bine pe masura ce i se vorbeste. Daca intrebarea este neclara sau vaga, Siri intreaba pentru a obtine clarificari. Invata din exemple si isi imbunatateste comportamentul in timp. Spre deosebire de alti asa-zisi agenti, Siri nu doar extrage cuvintele cheie din conversatie si le cauta in dictionar, ci incearca sa deriveze intelesul cuvintelor, adica semantica lor.

2.2 Web Semantic

Serviciile web sunt cunoscute de ceva vreme deja. Le folosim in multe scenarii, de la verificarea unui mers al trenurilor online, pana la aprobarea platii cu cartea de credit sau aflarea prognozei vremii oriunde in lume.
Webul semantic este un concept relativ nou (Figura 1). Pana acum a fost un subiect “fierbinte” in cercetare, fara prea multe aplicatii in industrie. Dar acest lucru s-ar putea schimba odata cu intrarea Google in lumea webului semantic, asa cum mentioneaza Wall Street Journal in editia din 15 Martie 2012.
Webul “normal”, adica ne-semantic, contine numai cele doua niveluri de la baza din Figura 1. Restul este specific webului semantic.
Nivelul XML ofera sintaxa pentru structurarea continutului documentelor online, fara a asocia nici o semantica cu datele continute.
RDF este un standard fundamental al webului semantic, exprimand modelele de date care reprezinta concepte si care sunt legaturile dintre ele. RDFS (RDF Schema) extinde RDF si este un limbaj generic pentru reprezentarea vocabularelor simple RDF pe web.

OWL (Web Ontology Language) este o extensie a RDFS care permite aplicatiilor sa proceseze si sa interpreteze continutul informatiilor, spre deosebire de simpla lor prezentare cititorilor umani. Astfel, OWL imbunatateste intepretabilitatea webului, oferind un vocabular aditional XML, RDF si RDFS si semantica formala.
Nivelele superioare – *unifying logic*, *proof* si *trust* – nu sunt inca complet realizate. *Proof* se refera la abilitatea de a demonstra ca anumite asertiuni online sunt adevarate sau false. Acest lucru are la baza logica – anume, cum este derivat raspunsul. *Trust* este ceea ce spune numele – ajuta partenerii autonomi care comunica sa ajunga la intelegeri bilaterale bazate pe incredere reciproca.

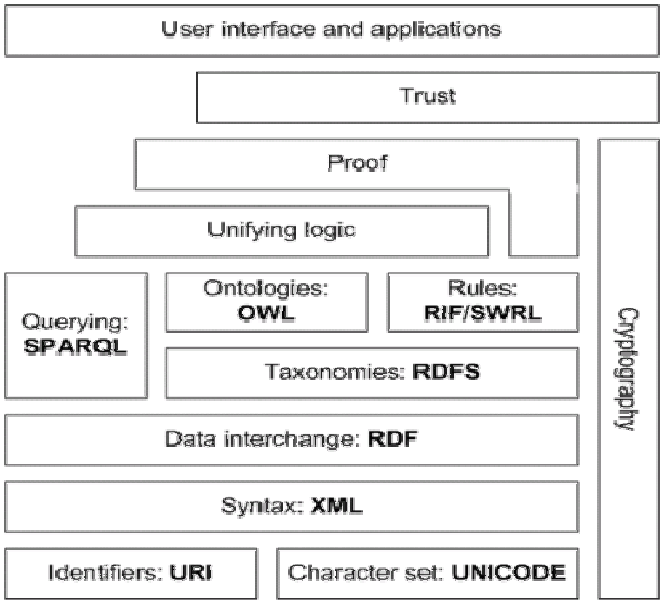

Figura 1: Celebrul "tort etajat" al webului semantic " (sursa: "Semantic Web – XML 2000", slide 10, Tim Berners-Lee, http//www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html)

Scenariul din capitolul 2.1 se refera la "agenti" si "servicii". Spre deosebire de agenti, serviciile pot fi considerate entitati publice, disponibile mereu (in mod ideal), deci care pot fi invocate de oriunde si la orice moment, presupunand ca adresa, *binding*ul si contractul sunt cunoscute. Agentii sunt mai degraba entitati private care executa sarcini in numele utilizatorilor. Un agent nu poate fi folosit de oricine la orice moment dat.
Spre exemplu, in scenariul mentionat mai sus, are mai mult sens sa implementam entitatea de urgenta ca un serviciu, pentru ca trebuie sa fie disponibila in mod public tuturor. Implementand-o ca un agent ar insemna sa fie accesibila numai altor agenti (si sa fie folosita prin comunicare intre agenti).
Pana la urma, alegerea de a implementa o entitate ca un serviciu sau agent nu este esentiala. Platforma JADE de exemplu ofera posibilitatea "impachetarii" unui agent in serviciu si expunerea sa publicului. Conceptual, atat agentii cat si serviciile executa operatii asupra anumitor date de input si returneaza rezultate. Esentiala pentru noi este posibilitatea de a integra serviciile si agentii intr-o singura platforma. Vom detalia aceasta observatie in capitolele urmatoare.

Un serviciu nu este altceva decat o operatie care respecta un anumit “contract”. Datorita caracterului eterogen al domeniului medical, nu ne putem astepta sa avem aceleasi servicii cu aceleasi date de intrare si de iesire folosite de toate institutiile medicale. Comunicarea intre institutiile medicale are nevoie de interoperabilitate semantica, care poate fi obtinuta prin folosirea ontologiilor. De exemplu, cand agentul unui doctor invoca un serviciu pentru a obtine fisa electronica medicala a unui pacient, trebuie sa ofere serviciului invocat anumite date de identificare ale pacientului si trebuie sa inteleaga si sa interpreteze raspunsul primit. Toate datele schimbate intre agent si serviciul respectiv trebuiesc exprimate in ontologii intelese de actorii implicati. De asemenea, cand doctorul scrie diagnosticul pus in fisa electronica a pacientului, acesta trebuie mapat pe anumite ontologii, astfel incat sa poate fi inteles de alte entitati (umane sau nu) care pot interpreta acele ontologii. Procesul este similar cu doua persoane care vorbesc limbi diferite si nu se inteleg, dar cand recurg la o a treia limba comuna, reusesc sa comunice.

2.3 Integrarea agentilor si serviciilor

Scenariul din capitolul 2.1 este descris din perspectiva agentilor software – entitati care ii ajuta pe posesorii lor sa atinga anumite scopuri. Agentii functioneaza pe baza unor paradigme de comunicare (schimbarea cunostintelor intre ei) si pe executarea unor sarcini prin migrarea in retea: algoritmii merg catre date. De exemplu, un agent migreaza catre o baza de date cu informatii medicale, citeste niste inregistrari, executa niste calcule si aduce rezultatele inapoi de unde a pornit.

Cealalta paradigma, mai populara deocamdata, pentru executarea sarcinilor la distanta prin Internet este serviciul web, caz in care datele migreza catre algoritmi.

Asa cum vom vedea in capitolele urmatoare, adnotarea semantica, inregistrarea, descoperirea semantica si invocarea, sunt aspecte similare in cazul serviciilor si al agentilor. Exista deci multe similaritati intre agenti si servicii folosite pentru interoperabilitate semantica intre ele.

Datorita folosirii pe scara larga a serviciilor web in zilele noastre, exista beneficii imense in invocarea lor de catre agenti sau de catre alte servicii. Putem de exemplu sa ne gandim la integrarea serviciilor web existente in diferite institutii medicale in scenarii noi propuse prin proiectul de fata.

3. SEMANTICA SI ONTOLOGII

3.1 Ontologii

In scenariul din capitolul 2.1 am vazut agentii software comunicand intre ei si potential cu servicii web. Dar din moment ce agentii si serviciile servesc scopuri diferite, sunt construiti de diverse companii si comunica prin protocoale diverse, mediul despre care vorbim este cat se poate de eterogen. In mod evident, nu ne putem astepta ca doi agenti oarecare sa poata comunica intre ei fara nici o interventie. Ei pot folosi protocoale de mesagerie diferite, limbaje diferite, ceea ce face comunicarea lor directa imposibila.
Ma mult, aceeasi informatie poate fi reprezentata in diferite feluri. De exemplu locatia fizica a pacientului poate fi reprezentata prin doua coordonate – latitidine si longitudine – sau printr-o adresa.
Solutia este interoperabilitatea semantica, bazata pe ontologii. Fiecare entitate care comunica trebuie sa foloseasca anumite ontologii in care isi exprima mesajele pentru alte entitati. Entitatea care primeste mesajele trebuie fie sa inteleaga aceste ontologii, fie sa execute o traducere a mesajelor in ontologii cunoscute.

HL7 [2] este un protocol de comunicatie intre institutii medicale. Versiunea 3 a acestui protocol a fost creata pentru a suporta interoperabilitatea semantica intre aplicatiile medicale. Figura 2 reprezinta HL7 RIM, care este nucleul versiunii 3 a acestui standard. Asa cum se poate vedea in figura, RIM are la baza cinci concepte fundamentale din care deriva multe altele.

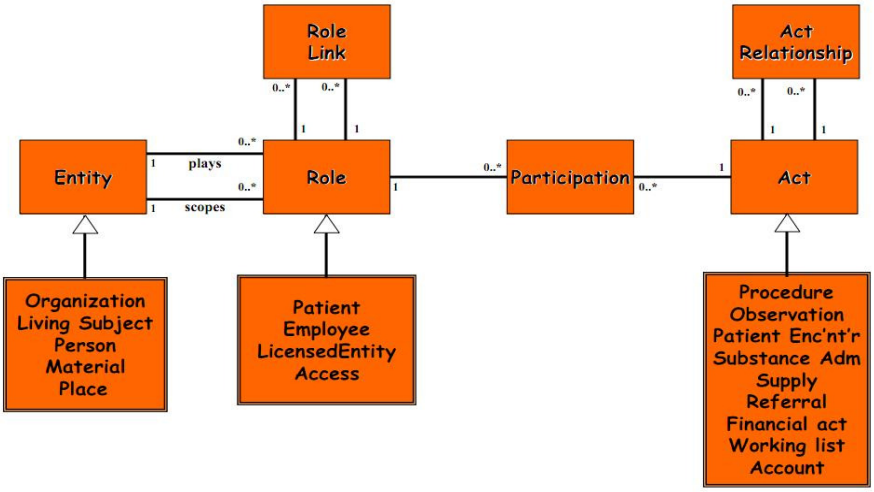

Figura 2: HL7 v3 RIM (Reference Information Model)

Ca sa intelegem mai bine rolul RIM-ului, sa ne gandim la urmatorul scenariu: avand o ontologie (precum SNOMED sau LOINC), de ce mai avem nevoie de un model informational (RIM = Reference Information Model)? Un concept exprimat in acea ontologie nu este suficient pentru a fi inteles de alte sisteme?

Ideea este ca acel concept poate insemna diverse lucruri, in functie de context. De exemplu, „tensiunea arteriala” poate reprezenta o observatie, o diagnoza sau poate o problema pentru pacient (de exemplu, tensiunea este peste limita acceptata). Un model informational ca RIM poate defini acest context.

Considerand cele spuse pana acum, este usor sa ne imaginam cum un scenariu ca cel descris in sectiunea 2.1 ar putea functiona: toti agentii si toate serviciile incapsuleaza informatii bazate pe HL7 RIM in mesajele pe care le trimit. Receptorul intelege si el HL7 RIM si prin urmare poate descifra mesajele intr-un mod lipsit de ambiguitate.
Totusi problema nu este rezolvata prin asta. O serie de alte intrebari trebuie sa-si gaseasca raspunsul:

- In scenariul nostru nu avem de-a face numai cu domeniul medical; ce se intampla deci cu celelalte concepte, care nu pot fi exprimate prin HL7 RIM ?
- Cum sunt adnotate mesajele astfel incat recepturul sa stie ce ontogolii sunt folosite ?
- Un agent care comunica cu un alt agent sau serviciu trebuie mai intai sa-si descopere cumva interlocutorul; cum se face acest lucru ?
- Cum sunt stocati agentii si serviciile, astfel incat sa poate fi descoperite de entitatile care vor sa comunice cu acestia ?

Toate aceste intrebari vor fi lamurite in capitolele urmatoare.

3.2 Adnotari semantice

Adnotarile semantice nu sunt noi. Poate cele mai raspandite adnotari curente sunt *tag*urile atasate paginilor web, cu scopul de a accelera cautarea, ajutand in acelasi timp la gasirea informatiei mai relevante si mai precise.
Totusi tagurile nu reprezinta adnotare semantica propriu-zisa. Adnotarea semantica adevarata implica un nivel mai adanc, anume imbogatirea datelor online nestructurate cu un context legat de un domeniu bine structurat. Scopul final al adnotarilor semantice nu este doar sa imbunatateasca procesul de cautare al datelor, ci sa transforme o simpla colectare a informatiei (bazata pe potrivire textuala) intr-o colectare structurata de date (bazata pe concepte dintr-un domeniu).

3.2.1 Adnotari semnatice pentru servicii web

Serviciile web sunt descrise folosind standardul WSDL, bazat pe XML. Descrierea este inregistrata intr-o baza de date – in mod traditional bazata pe UDDI – si astfel serviciul web poate fi descoperit si invocat. Cel putin aceasta este teoria. In practica insa pasul inregistrarii in repertoriul UDDI este sarit, iar adresa serviciului este codata in client. De asemenea, un proxy la serviciu este generat pe baza descrierii WSDL. Proxy-ul este compilat impreuna cu clientul serviciului web si astfel clientul poate invoca serviciul prin intermediul acestui proxy local, clientului fiindu-i de fapt transparent cu cine comunica.

Exista cateva probleme cu descrierea WSDL a serviciilor web. In primul rand, WSDL este o reprezentare pur sintactica a operatiilor cu parametrii si rezultatele lor. WSDL nu spune nimic despre comportamentul serviciului sau despre semantica operatiilor sale.

Asa cum o pagina de web oarecare poate fi adnotata semantic, tot asa serviciile web pot fi si ele adnotate semantic. Recomandarea W3C este SAWSDL – Semantic Annotations for WSDLS and XML Schema, standard bazat pe mai vechiul WSDL-S.
Pe scurt, SAWSDL suporta adnotarea anumitor elemente ale WSDL.
Comparativ cu alte standarde, precum WSDL-S, WSMO, etc., SAWSDL ofera un model simplificat de adnotare semantica, direct in fisierul WSDL. Fisierul WSDL adnotat poate fi inregistrat intr-un repertoriu bazat pe UDDI, la fel ca orice descriere WSDL standard.

SAWSDL propune doua tipuri de adnotari semantice:

1. *modelReference*, care specifica o mapare intre un element WSDL sau o componenta a unei XML Schema si un concept intr-un anumit model semantic; conform recomandarii W3C, „un modelReference poate fi folosit cu fiecare element al unui WSDL sau XML Schema; totusi, SAWSDL defineste semnificatia numai pentru wsdl:interface, wsdl:operation, wsdl:fault, xs:element, xs:complexType, xs:simpleType si xs:attribute”
2. *liftingSchemaMapping* si *loweringSchemaMapping*, care anoteaza elemente ale XML Schema si definitii de tip pentru a specifica mapari intre XML si date semantice

Sa luam de exemplu scenariul din capitolul 2.1 si sa definim o operatie a serviciului de urgenta – anume sfatul dat pacientului pe baza simptomelor, a istoricului medical si a locatiei curente a pacientului. Un extras simplificat din descrierea WSDL a operatiei este dat mai jos.

```
<wsdl:binding name="EmergencyServiceSoapBinding"
type="es:EmergencyServicePortType">

    <soap:binding style="document"
 transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="GetAdvice">
      <soap:operation
soapAction="http://www.emergencyservice.org/GetPatientAdvice"/>
      <wsdl:input name="Patient">
        <soap:body       use="literal"       namespace="http://schemas.
emergencyservice.org /GetPatientAdvice.xsd"/>
      </wsdl:input>
      <wsdl:output name="MedicalAdvice">
        <soap:body       use="literal"       namespace="http://schemas.
emergencyservice.org /GetPatientAdvice.xsd"/>
      </wsdl:output>
      <wsdl:fault>
        <soap:body       use="literal"       namespace="http://schemas.
emergencyservice.org /GetPatientAdvice.xsd"/>
      </wsdl:fault>
    </wsdl:operation>
 </wsdl:binding>
```

Acest extras defineste operatia cu inputul, outputul si mesajul de eroare. Tipurile de date ale acestora sunt definite in XML Schema atasata:

```
    <xsd:schema
targetNamespace="http://namespaces.emergencyservice.org"
      xmlns:xsd="http://www.w3.org/1999/XMLSchema">

      <xsd:element name="Patient">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="name" type="string"/>
             <xsd:element name="addr" type="string"/>
<!– many other fields, corresponding to HL7-RIM ontology →
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="Location">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="latitude" type="string"/>
            <xsd:element name="longitude" type="string"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="MedicalAdvice">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="hospital" type="string"/>
            <xsd:element name="location" type="#Location"/>
<!-- many other fields -->
          </xsd:all>
        </xsd:complexType>
      </xsd:element>

      <xsd:element name="GetPatientAdviceFault">
        <xsd:complexType>
          <xsd:all>
            <xsd:element name="errorMessage" type="string"/>
          </xsd:all>
        </xsd:complexType>
      </xsd:element>

    </xsd:schema>
```

Aceasta este descrierea WSDL neadnotata semantic. Pentru a da semnificatie acestei descrieri si a facilita procesul de cautare, se pot adauga adnotari SAWSDL, asa cum aratam in continuare:

```
     <xsd:element name="Patient">
        <xsd:complexType  sawsdl:modelReference="http://www.hl7.org/spec
/ontology/rim#Patient">
          <xsd:sequence>
            <xsd:element name="name" type="string"/>
```

```
            <xsd:element name="addr" type="string"/>
<!– many other fields, corresponding to HL7-RIM ontology →
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
```

Atributul *sawsdl:modelReference* din exemplu spune ca tipul *Patient* folosit de serviciul nostru corespunde conceptului *Patient* din ontologia HL7-RIM (nota: URLul de mai sus este fictional, el doar ilustreaza idea).
Adnotarile SAWSDL asa cum sunt descries de standard, sunt primul pas in directia descoperirii semantice. Cu toate acestea, descoperirea semantica ramane relativ imprecisa. De exemplu, adnotarea semantica la care se refera modelReference din exemplul urmator nu specifica natura informatiei semantice, adica nu stim daca "validPatientInfo" defineste o preconditie, un rezultat sau un efect.

```
<operation name="getPatientRecord" modelReference= "validPatientInfo
patientRecord">
```

Cu cat adaugam mai multa informatie semantica procesului de matching, cu atat mai precisa este potrivirea. Exista inca multa cercetare in zona imbunatatirii SAWSDL cu mai multe tipuri de informatie semantica [3]. In general aceasta informatie suplimentara se refera la IO, adica *Inputs* si *Outputs*.
Unii cercetatori [3] merg mai departe si propun adaugarea precoditiilor si a efectelor operatiilor. Ajungem astfel la adnotarile IOPE, adica *Inputs, Outputs, Preconditions, Effects*. Conform propunerii YASA4WSDL [3], exemplul de mai sus ar putea fi rescris pentru a include "serviceConcept":

```
<operation name="getPatientRecord" serviceConcept=
"ServiceOntology;#precondition ServiceOntology;#effect" modelReference=
"validPatientInfo patientRecord">
```

Adaugand acest *serviceConcept*, spunem de fapt ca prima informatie semantica, *validPatientInfo*, este o preconditie si ca a doua, *patientRecord*, este un efect al operatiei. Aceasta imbunatateste considerabil procesul de cautare semantica, dar si alte operatii, precum compunerea serviciilor.

3.2.2 Adnotari semantice pentru agenti

Serviciul de "pagini aurii" pentru agenti software se numeste Directory Facilitator (DF) si este descris de o specificatie FIPA [4]. Desigur, DF nu prevede si cautarea semantica, de care am avea nevoie, ca si in cazul serviciilor web.
Sa luam ca exemplu agentul receptiei unui spital, a carui descriere este inregistrata in DF. Serviciul tipic pe care un astfel de agent il ofera este de a primi simptomele pacientului, pe baza carora cauta un specialist potrivit in acel spital, sau eventual in altul apropiat.

```
(df-agent-description
  :name
    (agent-identifier
      :name ReceptionHospitalXYZ@xyz.com
```

```
    :addresses (sequence iiop://xyz.com/acc))
:services (set
  (service-description
    :name reception
    :type reception
    :ontology (set SNOMED)
    :properties (set
      (property
        :name "hospital reception area"
        :value 224890005)
      (property
        :name "temperature symptoms"
        :value 271399003)
    ....
     )))
:protocol (set FIPA-Request FIPA-Query)
:ontology (set SNOMED FIPA-Agent-Management)
:language (set FIPA-RDF))
```

Sa remarcam mai intai faptul ca ontologia SNOMED este folosita in aceasta descriere: valoarea 224890005 in SNOMED corespunde cu "*hospital reception area*" [5]. Dupa acest cod, enumeram toate simptomele care pot fi trimise acestui agent, impreuna cu codurile SNOMED corespunzatoare. Aceste proprietati sunt de fapt un fel de adnotari semantice folosite aici pentru agenti, asa cum pentru servicii web foloseam adnotarile din SAWSDL.

Totusi, aceste adnotari nu sunt suficiente, in general. De exemplu, descrierea de mai sus poate fi folosita pentru a raspunde la intrebarea *"Exista un agent care foloseste codul SNOMED 224890005?"*, dar nu si la intrebarea mai concreta si mai precisa *"Exista un agent care accepta ca input simptome descrise in ontologia X si returneaza un medic specialist descris in ontologia Y ?"*.

Mai intai sa observam faptul ca standardul FIPA permite prezenta urmatorilor parametri in descrierea unui agent inregistrat in DF:

Parametru	**Descriere**	**Prezenta**	**Tip**	**Valori rezervate**
`Name`	The name of the service.	Optional	`String`	
`Type`	The type of the service.	Optional	`String`	`fipa-df` `fipa-ams`
`Protocol`	A list of interaction protocols supported by the service.	Optional	Set of `String`	
`Ontology`	A list of ontologies supported by the service.	Optional	Set of `String`	`FIPA-Agent-Management`
`Language`	A list of content languages supported by the service.	Optional	Set of `String`	
`Ownership`	The owner of the service	Optional	`String`	
`properties`	A list of properties that discriminate the service.	Optional	Set of `property`	

Asa cum se poate vedea, pe langa proprietatile obisnuite, precum *Name*, *Type*, etc., exista niste proprietati in format liber care iau forma de perechi cheie-valoare si care pot fi folosite pentru a discrimina serviciul descris. Prin urmare, cautarea agentului se poate reduce la potrivirea sintactica a valorilor acestor proprietati.
Ceea ce avem nevoie este totusi o potrivire semantica, asa cum am discutat si in cazul serviciilor web. Propunerea noastra este extiderea standardului FIPA, astfel incat sa accepte noi parametrii pentru agentii inregistrati. Acesti noi parametrii ar trebui sa fie optionali (folositi numai pentru cautare semantica) si ar trebui sa include acele IOPE (Inputs, Outputs, Preconditions, Effects) despre care am vorbit in sectiunea anterioara. Cu aceasta idee, exemplul anterior devine:

```
(service-description
      :name reception
      :type reception
      :ontology (set SNOMED)
      :properties (set
        (property
          :name "hospital reception area"
          :value 224890005)
        (property
          :name "temperature symptoms"
          :value 271399003)
      ....
      )
      :inputs (set
        (input
          :name "symptoms"
          :value "http://www.ihtsdo.org/snomed/ontology#Symptom"
        )
      )
      :otputs (set
        (output
          :name "specialist"
          :value "http://www.fipa.org/agents/ontology#Specialist"
        )
      )
)
```

Am adaugat aici inputul si outputul, doua dintre extensiile propuse pentru standardul FIPA DF. Acum putem raspunde la intrebarea care fara aceste noi proprietati nu avea un raspuns: "Exista un agent care accepta ca input simptome descrise in ontologia SNOMED si returneaza datele unui medic specialist descrise in ontologia FIPA ?". Cu aceasta intrebare nu mai suntem legati de un anume cod SNOMED. Putem chiar sa omitem ontologiile din intrebare: "Exista un agent care accepta ca input simptome si returneaza descrierea unui medic specialist?". In acest caz se poate pune si problema unei potriviri intre ontologii (ontology matching), anume intre conceptele cautate si cele inregistrate in DF.
Sa observam ca Directory Facilitator nu are nevoie de nici o extensie pentru a suporta descrierea propusa, adnotata cu IOPE. Aceste IOPE trebuie inregistrare in baza de cunostinte folosita de DF, ceea ce inseamna de exemplu pastrarea perechilor cheie-valoare precum agentID – IOPE.

Desigur ca pentru descoperirea semantica a acestor descrieri adnotate este nevoie de un motor de inferente, asa cum vom discuta pe parcursul lucrarii.

3.3 Inregistrarea si descoperirea semantica

Sectiunea 3.2 a pus bazele cautarii semantice a serviciilor si agentilor, prin prezentarea adnotarilor semantice care pot fi adaugate acestora.

3.3.1 Inregistrarea semantica a serviciilor

Initiativa UDDI [6] a fost standardul de facto pentru publicarea serviciilor web. Ideea era ca un serviciu sa poata fi publicat pentru ca apoi sa fie descoperit de un client. Odata descoperit, descrierea WSDL era gasita si un proxy la serviciu era generat pe baza acesteia.
Mai mult decat atat, exista cercetare care arata cum pot fi mapate conceptele din SAWSDL in UDDI:

- fie adnotarea directa a conceptelor UDDI cu informatie din SAWSDL; de exemplu, adnotarile semantice din SAWSDL sunt transformate in *categoryBags* din UDDI [7]; aceasta abordare este simpla, nu necesita un rationament semantic, dar asta inseamna o pierdere de flexibilitate (pentru ca astfel cautarea in UDDI ramane sintactica)
- fie anumite module cu capabilitati de procesare semantica sunt adaugate peste UDDI; aceste module realizeaza de fapt cautarea semantica, de exemplu procesare ontologica bazata pe OWL si rationament DL (*Description Logic*)

Totusi Microsoft, IBM si SAP au renuntat in 2007 sa mai suporte standardul UDDI pe care tot ei l-au creat. Ca urmare, este timpul pentru ceva nou, care raspunde limitarilor principale ale UDDI:

1. complexitate mare
2. ignorarea securitatii
3. lipsa uneltelor potrivite (autorul a petrecut o buna bucata de timp incercand sa instaleze o implementare functionala a UDDI; in cele din urma a reusit doar cu *jUDDI for Tomcat 3.0.1*)

Acestea fiind spuse, in opinia autorului viitorul inregistrarii si descoperirii semantice este in motoarele de cautare semantice. Caracteristicile principale ale unui astfel de motor de cautare semantica ar trebui sa fie:

1. simplitate – trebuie sa ofere o interfata clara si simplu de folosit prin care fie un utilizator, fie un program poate sa caute si sa gaseasca serviciile dorite
2. trebuie sa ofere un mod sigur de a accesa un serviciu
3. trebuie sa ofere un API prin care sa poata fi inregistrate si descoperite serviciile
4. trebuie sa fie scalabil la milioane de utilizatori simultani si in acelasi timp sa poata raspunde in cateva sute de milisecunde (similar cu un motor de cautare current, precum Google)

Principiile unui astfel de motor de cautare semantica nu ar trebui sa difere mult de cele ale unui motor clasic. Cel semantic trebuie sa poata indexa informatie semantica. Aceasta indexare reprezinta de fapt inregistrarea seviciilor. Mai precis, ce anume va indexa acest motor de cautare ? In cazul serviciilor web, descrierile WSDL adnotate semantic (de exemplu conform standardului SAWSDL, asa cum am discutat in sectiunea 3.2.1). Sau in cazul serviciilor RESTful, ar putea indexa descrierile WADL [9] adnotate semantic.
Descoperirea semantica va fi bazata pe cautarea unui profil dat in repertoriul de servicii indexate si gasirea celor mai potrivite in ordine descrescatoare, conform anumitor masuri de potrivire. Aceasta inseamna deci ca va trebui sa definim o masura a potrivirii semantice. Pe parcursul acestui document vom detalia cercetarea curenta referitoare la astfel de masuri de potrivire semantica. De asemenea ne vom referi la cerintele ne-functionale ale unui astfel de repertoriu.

3.3.2 Descoperirea semantica a serviciilor

Descoperirea serviciilor web traditionale are loc prin cautare textuala in UDDI, asa cum am discutat mai sus. Pentru servicii semantice, avem nevoie de potrivire semantica, nu doar textuala.
Cercetarea din aceasta arie propune module semantice adaugate peste arhitectura UDDI. Aceste module ar fi responsabile pentru procesare ontologica (folosind OWL-S [10] sau DAML-S [11]) si rationare logica. Ideea este de a descrie ontologiile folosite de servicii cu OWL sau DAML si a adauga aceasta descriere in UDDI.
Problema fundamentala cu aceasta propunere este ca UDDI nu mai este sustinut de marile companii, asa cum am aratat mai sus.
Alte directii de cercetare [12] se concentreaza pe potrivirea comportamentala a serviciilor web, definind protocolul de conversatie al unui serviciu ca un graf si folosind apoi un algoritm de potrivire pe grafuri. Acest lucru ofera o foarte precisa potrivire semantica (cercetatorii definesc si o masura a potrivirii pentru a cuantifica serviciile web), dar complexitatea algoritmului creste exponential cu numarul pasilor conversatiei.

Indiferent de modul de stocare a serviciilor (in UDDI sau nu), propunem urmatoarea definitie pentru descoperirea semantica: *procesul prin care un profil de serviciu cautat (cerut) este potrivit cu profile de servicii inregistrate, o astfel de potrivire fiind masurata ca o distanta semantica intre profile*.
De exemplu, putem descrie IOPE (la care am facut referire mai sus) ale serviciilor folosind un limbaj pentru descrierea ontologiilor precum OWL. Descrierile WSDL ale serviciilor inregistrate sunt adnotate cu aceste IOPE adnotate semantic (vezi sectiunea 3.2.1). Un profil care descrie un serviciu cautat este si el definit in termeni IOPE, din nou folosind ontologii. Acestea fiind spuse, descoperirea semantica implica potrivirea semantica intre profilul cautat si cele inregistrate, masurand distantele semantice intre I, O, P, E corespunzatoare.
Ca si in capitolul precedent, descoperirea semantica poate avea la baza un motor de cautare semantic, lucru care va fi detaliat pe parcursul lucrarii.

3.3.3 Inregistrarea semantica a agentilor

Descoperirea semantica este desigur strans legata de adnotarea semantica: un serviciu/ agent este descoperit pe baza adnotarilor care i s-au dat.
Specificatia FIPA a DF [4] spune ca un agent care se inregistreaza intr-un DF poate sa-si specifice ontologiile folosite. Totusi, spre deosebire de serviciile web (vezi SAWSDL), descrierea unui agent conform FIPA nu poate fi adnotata semantic.
In sectiunea 3.2.2 am propus o extensie a standardului FIPA DF pentru a permite adnotarea semantica a agentilor, urmarind aceeasi idee ca si in cazul adnotarii serviciilor web, anume adaugarea IOPE adnotate semantic. Practic acest lucru inseamna ca un agent isi declara un serviciu specificandu-i inputurile, outputurile, preconditiile si efectele asteptate. Aceste adnotari trebuiesc adaugate cumva la informatiile prezente in DF, astfel incat alti agenti sau servicii sa poata cauta semantic functionalitatea oferita de agentii nostri, pentru o mai buna potrivire.
Vom detalia aceste idei pe parcursul tezei.

3.3.4 Descoperirea semantica a agentilor

In sectiunea 3.2.2 am propus o extensie a standardului FIPA DF, astfel ca inregistrarea agentilor sa devina similara cu cea a serviciilor web. Acest lucru este bazat pe parametrii IOPE, al caror scop este sa adauge informatie semantica despre comportamentul serviciilor.
Urmand aceasta idee, descoperirea agentilor devine si ea similara cu cea a serviciilor web. Prin urmare, descoperirea semantica a agentilor ale caror descrieri FIPA au fost annotate cu IOPE inseamna de fapt o potrivire semantica intre adnotarile IOPE ale profilelor agentilor inregistrati si cele ale profilului agentului cautat.
Pentru aceasta potrivire semantica, una din metodele de mai jos poate fi folosita:

- fie baza de cunostinte folosita de DF este extinsa cu capabilitati de potrivire semantica pentru IOPE; aceasta poate include potrivire intre ontologii, atunci cand ontologiile agentilor inregistrati sunt diferite de cele ale agentilor cautati
- fie un nou nivel de potrivire semantica este adaugat peste DF, lasandu-l pe cel din urma nemodificat; procesul de potrivire semantica a IOPE s-ar petrece atunci in afara DFului

In capitolele 3.3.1 si 3.3.2 am propus un motor de cautare semantic ce ar putea fi folosit atat pentru inregistrarea, cat si pentru descoperirea serviciilor web. De fapt am putea extinde aceasta idee si la agenti, pentru ca motorul de cautare este dependent numai de potrivirea IOPE (deci de descrierile profilelor in care apar IOPE), nu si de tehnologia folosita. Mai precis, un agent, ca si un serviciu web, poate fi descris printr-un profil ce contine IOPE adnotate semantic.

4. COMUNICAREA

4.1 Comunicarea intre servicii

Mesajele SOAP, folosite in comunicarea intre servicii web clasice, pot fi compuse la baza din mesaje RDF peste care este scris nivelul SOAP [19]. Ideea este de a comunica un mesaj care ofera posibilitatea rationamentului ontologic, astfel incat sa obtinem interoperabilitatea semantica. Ontologia fundamentala in cazul nostru este bazata pe HL7, asa cum arata Figura 3.

Avem acum nu numai adnotare semantica (sectiunea 3.2.1), descoperire semantica (sectiunea 3.3.2), dar si comunicare semantica intre servicii.

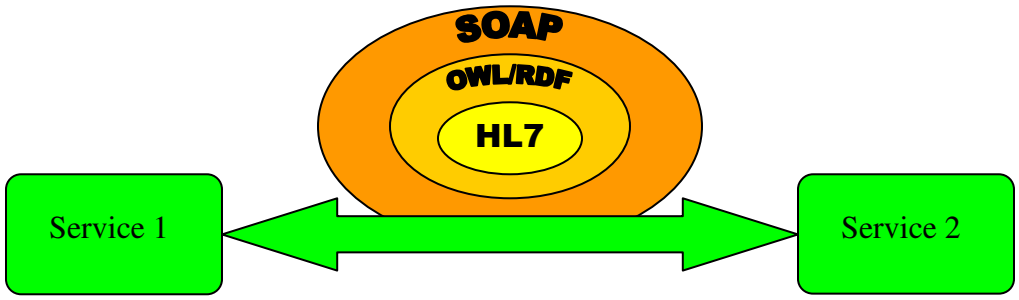

Figura 3: Comunicarea intre servicii: un mesaj HL7 codat in RDF, impachetat intr-un mesaj SOAP

4.2 Comunicarea intre agenti

In mod similar cu comunicarea intre servicii, unde sunt folosite mesaje OWL/RDF impachetate in SOAP, agentii pot comunica si ei folosind OWL/RDF in mesaje FIPA ACL. Documentul oficial [12] se refera la continut RDF in mesaje FIPA, iar cercetarea curenta [20] propune OWL (a carui sintaxa este de fapt RDF) ca *FIPA content language*.

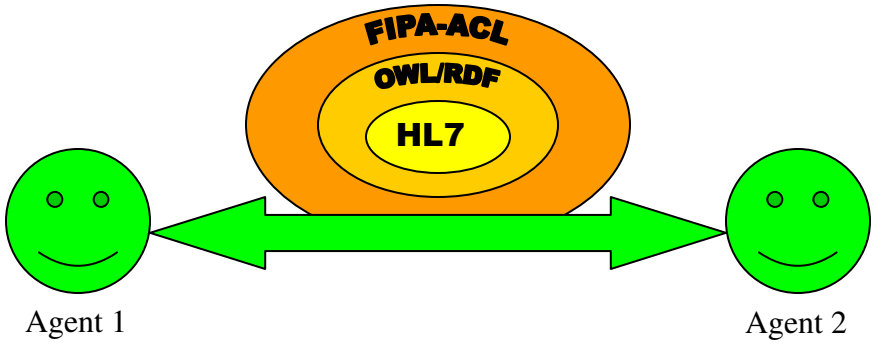

Figura 4: Comunicarea intre agenti: mesaj HL7 exprimat in OWL/RDF, impachetat in FIPA ACL

Fiecare strat din mesajele schimbate are un scop bine definit: HL7 este folosit pentru interoperabilitate semantica; OWL/RDF este folosit pentru a transforma conceptele HL7 intr-o ontologie reala, cu expresivitatea pe care HL7 singur nu o poate atinge. De exemplu, pot fi exprimate conditii precum: daca un anumit simptom are o anumita valoare, atunci alte simptome pot fi prezente. Se poate invoca faptul ca HL7-RIM deja descrie o ontologie. Intradevar, exista entitati descrise de HL7-RIM si relatii intre ele, dar sunt exprimate in diagrame UML. Avem nevoie de o traducere a acestor diagrame intr-un limbaj "real" conceput pentru descrierea ontologiilor, precum OWL.
Cand un mesaj OWL/RDF trebuie comunicat unui alt agent software, el are nevoie de un strat standard FIPA ACL.

4.3 Comunicarea intre agenti si servicii

Un caz mai interesant de comunicare este intre un agent si un serviciu. In acest caz, stratul FIPA ACL trebuie inlocuit cu un strat SOAP in fiecare mesaj trimis de la agent la serviciu si vice-versa in cazul invers. Cine face aceste conversii ?

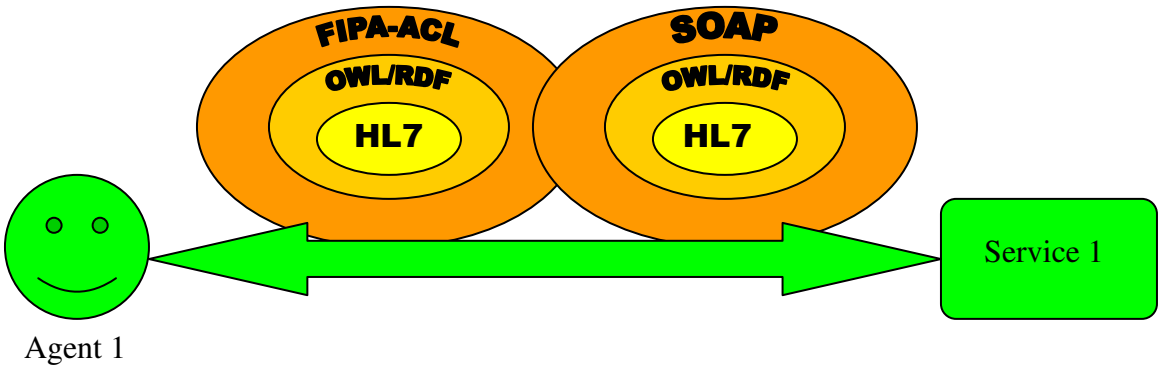

Figura 5: Comunicarea agent-serviciu: stratul FIPA ACL este inlocuit de un strat SOAP

Pentru a raspunde la intrebare, urmeaza cateva consideratii generale:

1. Agentii si serviciile web trebuie sa fie complet ignorante in privinta interlocutorului, cu alte cuvinte sa nu stie cu cine comunica – cu un agent sau cu un serviciu web
2. Operatiile agentilor si serviciilor web sunt inregistrate intr-o baza de date
3. Agentii propriu-zisi sunt inregistrati in baze da date FIPA DF

In mod normal, cand un client invoca un serviciu web, de fapt clientul comunica cu un proxy local, generat din descrierea WSDL a serviciului. Un agent este si el un client al unui seviciu web, deci poate urma aceeasi cale, asa cum arata Figura 6. Pe langa mesajele FIPA ACL pe care agentul le trimite altor agenti, el poate oricand cauta un serviciu si-l poate invoca prin intermediul proxyului.

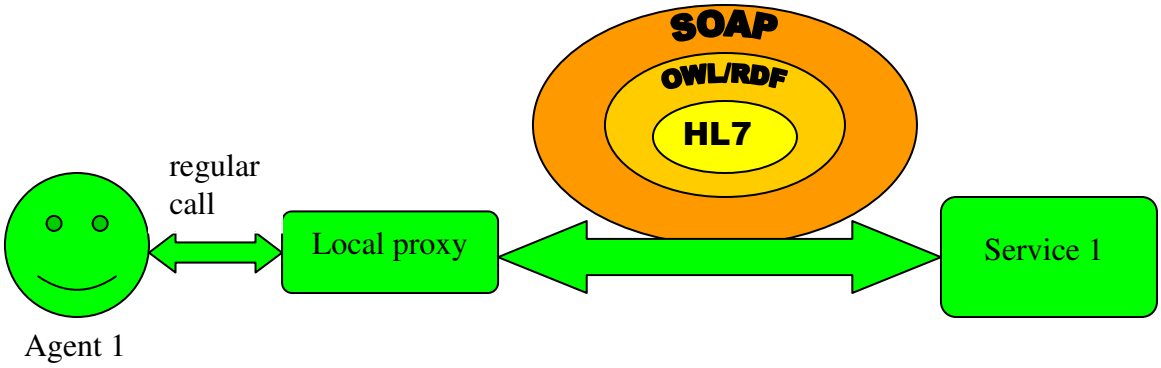

Figura 6: Invocarea unui serviciu printr-un proxy local

Un caz mai interesant este atunci cand un serviciu web invoca un agent. Motorul de cautare semantic pe care il propunem in a doua parte a lucrarii ca un indexer pentru servicii web poate indexa desigur si agenti software. Mai precis, asa cum descrierile WSDL ale serviciilor web sunt parsate si indexate, tot asa si descrierile agentilor, in mod normal inregistrate in DF, pot fi parsate si indexate (dupa ce sunt adnotate semantic, asa cum am descris in sectiunea 4.2).
Concret, putem exploata mecanismul WSIG [1] oferit de platform JADE, care permite ca agentii sa fie expusi ca servicii web. Acest lucru este sugerat de Figura 7. Cand un agent este inregistrat in DF, agentul WSIG, care subscribe acestor evenimente de inregistrare/ deregistrare, creeaza o descriere WSDL din descrierea agentului si o publica in motorul nostru de cautare, *Semmed*. Urmarind propunerea de adnotare a agentilor cu IOPE din sectiunea 4.2, un agent cu descrierea:

```
:inputs (set
    (input
        :name "symptoms"
        :value "http://www.ihtsdo.org/snomed/ontology#Symptom"
    )
)
:outputs (set
    (output
        :name "specialist"
        :value "http://www.fipa.org/agents/ontology#Specialist"
    )
)
```

…. este transformat intr-o descriere WSDL cu urmatoarele adnotari:

```
<wsdl:input>
  <xsd:element name="symptoms">
      <xsd:complexType sawsdl:modelReference=
          "http://www.ihtsdo.org/snomed/ontology#Symptom">
        <xsd:sequence>
          ….
        </xsd:sequence>
      </xsd:complexType>
  </xsd:element>
</wsdl:input>
<wsdl:output>
```

```
<xsd:element name="specialist">
    <xsd:complexType
sawsdl:modelReference="http://www.fipa.org/agents/ontology#Specialist">
      <xsd:sequence>
        ….
      </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</wsdl:output>
```

Odata ce inregistrarea WSDL adnotata este terminata, Service 1 din figura urmatoare poate cauta serviciul dorit cu ajutorul motorului de cautare semantic Semmed, fara a sti ca descrierea gasita vine de la un agent. Apoi Service 1 invoca agentul apeland Servletul WSIG din infrastructura WSIG, prin proxyul local.

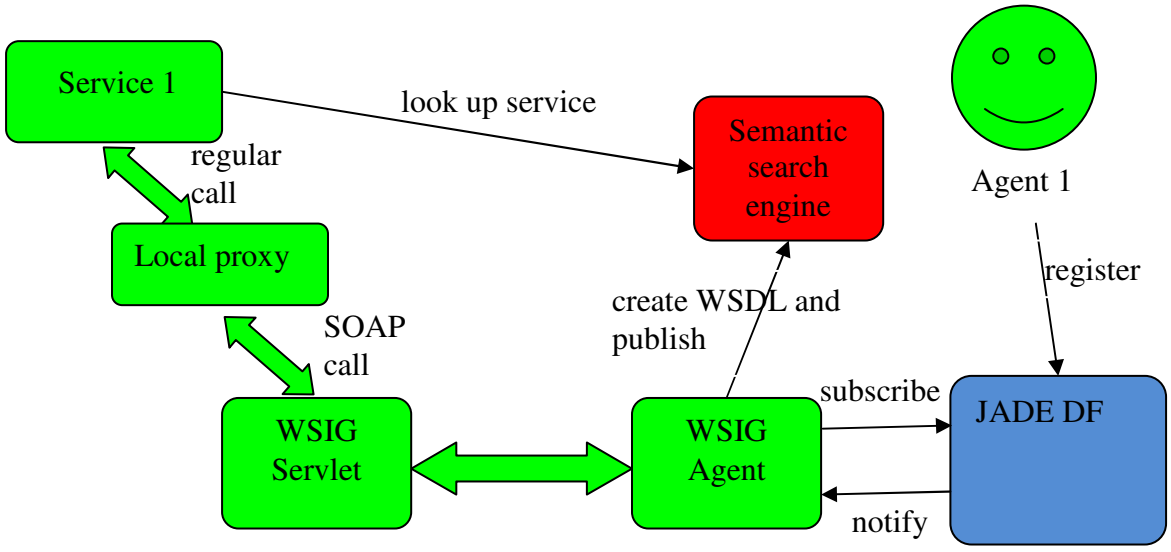

Figura 7: Invocarea unui agent de catre un serviciu web

5. UN MOTOR DE CAUTARE SEMANTICA

5.1 Introducere

In sectiunile precedente am discutat despre faptul ca cercetarea curenta in legatura cu inregistrarea si descoperirea semantica a serviciilor web este orientata in jurul UDDI. Insa, asa cum am remarcat, UDDI a pierdut suportul companiilor care l-au creat.
In opinia autorilor, viitorul inregistrarii si descoperirii semantice a serviciilor web se afla in motoarele de cautare semantice, sa zicem un Google cu capabilitati semantice. Caracteristicile esentiale ale unui astfel de motor semantic ar fi:

1. Simplitatea – trebuie sa ofere o interfata simpla si clara prin care fie un utilizator sau un alt program sa poata cauta si gasi serviciile dorite
2. Trebuie sa ofere un mod securizat de acces la sevicii
3. Trebuie sa ofere APIuri pentru inregistrarea si descoperirea serviciilor
4. Trebuie sa fie scalabil la milioane de utilizatori simultani si totusi sa raspunda in cateva sute de milisecunde – nu departe de performanta motorului de cautare Google

Cautarea semantica inseamna cautarea unor concepte inregistrate (*„advertised profile”*) care pot fi potrivite semantic cu conceptele date (*„requested profile”*). In mod normal dorim mai mult decat potrivire in cadrul aceleiasi ontologii; anume ne intereseaza si *ontology matching* atunci cand avem de-a face cu concepte din ontologii diferite.

In aceasta lucrare, cautarea semantica la care ne referim este folosita de programe software care cauta servicii (operatii cu anumita functionalitate). Aceste servicii pot fi oferite atat de catre servicii web, cat si de agenti software. Pentru indexarea serviciilor, motorul de cautare va utiliza descrierile WSDL adnotate conform standardului SAWSDL. Pentru agenti, asa cum am aratat in sectiunea 4.3, un sistem gen JADE WSIG este repsonsabil pentru transformarea descrierilor agentilor in fisiere WSDL, si ele adnotate tot conform SAWSDL. Prin urmare, atat in cazul serviciilor web, cat si in cazul agentilor, motorul nostru de cautare ar indexa tot descrieri WSDL, fiindu-i transparent faptul ca acestea provin de la servicii web reale sau de la agenti.

5.2 Cercetarea curenta

In domeniul potrivirii semantice pentru servicii web, cea mai mare parte a cercetarii se concentreaza pe potrivirea pe baza distantei semantice intre concepte. Sunt mai multe tipuri de algoritmi de potrivire semantica, de exemplu cei care includ numai intrarile si iesirile operatiilor oferite de servicii, cei care includ preconditii si efecte, potrivire bazata pe logica sau nu, potrivire hibrida, etc.
In aceasta lucrare propunem potrivirea bazata pe IOPE (inputs, outputs, preconditions, effects) pentru descoperirea serviciilor web. Un astfel de algoritm este descris in [14]. Este vorba despre un algoritm hibrid, bazat pe distanta semantica pentru fiecare din I, O, P si E, calculat pe baza reprezentarii fiecarui concept ca un element intr-un arbore al ontologiei.

Chiar daca algoritmul nu depinde de inregistrarea serviciilor web, lucrarea [14] si altele mentioneaza UDDI ca mijoc de a inregistra serviciile.
In [10], autorii afirma faptul ca UDDI nu are doua caracteristici importante:

1. Mecanismul de cautare este limitat la cautare textuala pe baza de cuvinte cheie si nu suporta nici o inferenta pentru a extrage informatie mai semnificativa; de exemplu, cautand dupa „Doctor" nu am descoperi nici un „Chirurg", pentru ca relatia de mostenire dintre aceste concepte nu este inteleasa de catre UDDI
2. Cautarea este bazata pe un anumit atribut din UDDI, de exemplu Category; pentru cautari mai precise ar trebui sa luam in considerare cel putin intrarile si iesirile serviciilor web; aceste informatii nu sunt exploatate de UDDI

Pentru a rezolva aceste probleme, autorii lucrarii [14] propun crearea unui plugin pentru UDDI care foloseste OWL-S pentru a adauga capabilitati semantice si de inferenta standardului UDDI. Totusi, asa cum am mentionat in repetate randuri, UDDI a pierdut suportul marilor companii si deci nu mai reprezinta o solutie de viitor.
In [15] autorii propun potrivirea serviciilor web pe baza asa-numitului *model Tversky*, in care similaritatea semantica este bazata pe proprietatile conceptelor (exprimate in ontolgiile lor). Autorii afirma ca aceasta metoda ofera rezultate mai precise decat metodele traditionale bazate pe distanta semantica. Mai mult, metoda aceasta poate fi folosita si pentru concepte din ontologii diferite, prin urmare s-ar tine seama si de *ontology matching*.
In aceasta lucrare, folosind cercetarea curenta, ne concentram pe un inlocuitor realistic al UDDI. Viziunea noastra este a unui motor de cautare semantica, pe care l-am denumit *Semmed*.

6. ARHITECTURA MOTORULUI DE CAUTARE SEMANTICA - *Semmed*

6.1 Similaritatea semantica

Un motor de cautare semantica trebuie sa caute si sa indexeze descrierile serviciilor web. Motorul de cautare trebuie sa raspunda cererilor cu o lista de servicii potrivite, in ordinea descrescatoare a gradului de potrivire. Acest lucru implica faptul ca trebuie definita o masura a potrivirii semantice, astfel incat rezultatele sa poate fi sortate corespunzator. Practic, potrivirea serviciilor inseamna potrivirea intrarilor si respectiv a iesirilor operatiilor acestora, eventual a preconditiilor si respectiv efectelor operatiilor – asa cum sunt ele definite in descrierile WSDL. Folosim pentru aceasta urmatoarea definitie a distantei semantice:

1. potrivire textuala, in care conceptul input/output din profilul cautat este alfanumeric identic cu conceptul oferit de un serviciu inregistrat, de exemplu: http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#OrderRequest (atat namespace-ul, cat si numele conceptului trebuie sa corespunda)
2. potrivire semantica bazata pe principiul covariantei: daca inputul din profilul cautat este o subclasa a unui input al unui serviciu inregistrat, acestia se potrivesc; intuitiv, asta inseamna ca daca un serviciu inregistrat poate lucra cu un anumit concept de intrare, poate atunci intelege si conceptele derivate din acesta, pentru ca ele sunt niste restrictii ale primului; vice-versa, pentru outputuri se aplica principiul contravariantei: daca un serviciu inregistrat are un anumit concept ca output (rezultat), nu putem cere ca rezultat o versiune restrictionata a acestuia, adica o subclasa; numai o superclasa a acelui concept reprezinta o potrivire semantica
3. potrivire semantica bazata pe similaritatea semantica Tversky [15]. In linii mari, similaritatea Tversky intre doua concepte A si B se defineste ca numarul de proprietati comune impartit la numarul total de proprietati ale conceptului B; rezultatul este un numar intre 0 (concepte complet diferite) si 1 (concepte identice); similaritatea Tversky se aplica si conceptelor din ontologii diferite

Motorul de cautare semantic propus in aceasta lucrare trebuie sa fie scalabil, tolerant la defecte, disponibil, sigur, mentenabil, etc. Pe scurt, trebuie sa aiba toate abilitatile unui motor de cautare traditional. De asemenea, viteza de reactie ar trebui sa fie comparabila cu cea a motoarelor de cautare clasice.

6.2 Blocurile functionale

Asa cum am remarcat mai devreme, cu cat mai multe elemente din descrierea WSDL sunt adnotate semantic, cu atat mai buna este potrivirea serviciilor. Am mentionat adnotarile pentru IOPE, adica inputs, outputs, eventual preconditions si effects. In aceasta lucrare ne referim la serviciile ale caror descrieri au IOPE adnotate cu concepte ontologice.

Figura 8 arata blocurile functionale ale motorului de cautare propus, *Semmed*. Cele trei functii principale sunt efectuate de blocurile Crawler, Indexer si Searcher.

Crawlerele

Am implementat prototipul nostru folosing Crawler4J, o implementare Java open source a crawlerelor. Am programat crawlerele noastre sa aduca informatia WSDL de la diferite siteuri, precum www.service-repository.com. Din pacate majoritatea bazelor de date de servicii online nu contin adnotari SAWSDL, prin urmare a trebuit sa dam manual pagini SAWSDL crawlerelor noastre.

Crawlerele parseaza paginile SAWSDL cautand ontologii, marcate de atributele de forma *sawsdl:modelReference*. Ontologiile(de exemplu http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder) sunt si ele aduse de crawlere intr-o baza de date.

Pentru a parsa SAWSDL folosim Apache Woden combinat cu niste extensii pentru SAWSDL.

Crawlerele executa si un task specific mai degraba indexerilor: deoarece ele deja parseaza documentele WSDL, extrag din acestea IOPE (inputurile, outputurile, preconditiile si efectele) si le stocheaza in baza de date. Astfel, crawlerele creeaza ceea ce se numeste *forward index*, care mapeaza documentele (IDurile unice ale acestora) cu IOPE gasite in ele.

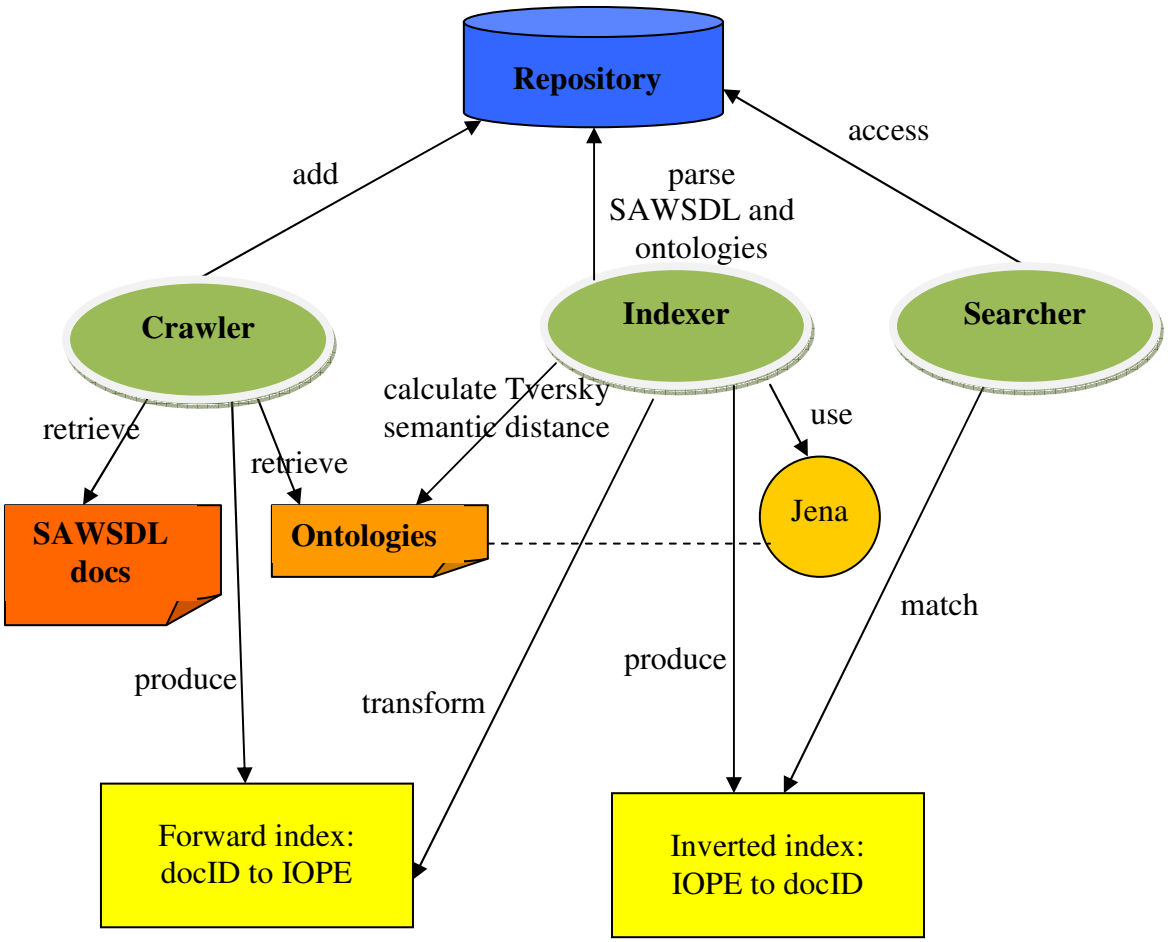

Figura 8: Blocurile functionale ale motorului de cautare semantic

Baza de date

Pentru implementarea prototipului motorului de cautare am ales BerkeleyDB ca baza de date. BerkeleyDB este un produs Oracle ce poate fi folosit gratis pentru aplicatii ne-comerciale. Arhitectura sa este bazata pe stocarea perechilor (key, value) in structuri de date eficiente (de exemplu B-trees), oferind concurenta inalta si viteza mare, impreuna cu simplitate in programare. Suporta tranzactii ACID si ofera suport pentru replicare, imbunatatind astfel disponibilitatea si scalabilitatea stocarii datelor. Aceste caracteristici fac din BerkeleyDB un bun candidat pentru implementarea indecsilor intr-un motor de cautare.

Fisierele WSDL si ontologiile la care fac referinta (fisiere OWL sau RDF) sunt stocate in baza de date comprimate cu zlib, care ofera un compromis bun intre performanta si dimensiunea comprimata.

Indexerul

Paginile WSDL sunt deja parsate si adnotarile IOPE din acestea sunt stocate in baza de date de catre crawlere, prin urmare munca indexerului devine mai usoara. Acesta construieste urmatorii indecsi:

- Unul pentru ontologii: searcherul trebuie sa gaseasca fiecare fisier cu ontologia cat se poate de repede; acest index mapeaza fiecare docID al fisierului WSDL unde apar IOPE cu lista de docID pentru fiecare fisier cu ontologia folosita de acele IOPE
- Unul pentru intrarile (inputs) serviciilor: intrarile trebuies sa fie potrivite (*matching*) in mod eficient
- Similar, pentru iesiri (outputs), preconditii si efecte, avem nevoie de cate un index

Indexerul are ca intrare indexul forward creat de crawler si din acesta creeaza indecsii pentru IOPE. Indexerul mai face un pas important, care accelereaza searcherul: pentru fiecare Input/Output, creeaza o lista cu conceptele din aceeasi ontologie ca si Inputul/Outputul curent, ordonate descrescator in functie de distanta Tversky de la Inputul/Outputul curent.

Aceste structuri de date sunt stocate in baza de date implementata cu BerkeleyDB ca tuple de forma: <*Input1,* <*lista de documente unde apare*>>. Intern, baza de date foloseste B-tree eficienti pentru reprezentarea indecsilor.

Searcherul

Avand un profil de cautat (IOPE), searcherul incearca sa gaseasca cea mai buna potrivire printre serviciile inregistrate (indexate). Ca sa-si execute algoritmul, searcherul acceseaza mai intai indecsii creati de BerkeleyDB in memorie, reprezentati ca structuri B-tree. In consecinta, accesul la disc este minimizat, lucru extrem de important pentru searcher, componenta pentru care performanta este o caracteristica mult mai importanta decat pentru crawlere sau pentru indexer. Searcherul foloseste distantele semantice Tversky precalculate de catre indexeri, pentru a optimiza gasirea serviciului cel mai potrivit.

7. ALGORITMUL DE CAUTARE

In continuare descriem algoritmul implementat de motorul de cautare semantic, mai precis de Searcher. Dandu-se un profil cautat cu descrierile IOPE de forma (RI*, RO+, RP*, RE*), Searcherul trebuie sa gaseasca cea mai buna potrivire cu profilele serviciilor inregistrate. RI* inseamna ca profilul cautat poate avea oricate Inputuri sau nici unul, dar RO+ spune ca trebuie sa existe cel putin un Output. Acest lucru este logic, pentru ca un serviciu poate sa nu aiba parametrii de intrare, dar va intoarce sigur un rezultat, fie si null. De remarcat ca aceste profile pot reprezenta atat servicii web, cat si agenti software, lucrul acesta fiindu-i total transparent motorului de cautare.

1. For each output, ROi
- listMatchedOutputs(ROi) <- nil
- lookup ROi in the Outputs index
- if found
    - foreach matched docID // add all docIDs of the WSDL files referred to by this output
        - listMatchedOutputs(ROi).Add(docID)
        - listMatchedOutputs(ROi)[docID] = 0 // mark the semantic distance as 0, the best match
- for each subclass RSubOi of ROi in its ontology
    - lookup RSubOi in the Output index
    - if found
        - foreach matched docID
            - listMatchedOutputs(ROi).Add(docID)
            - listMatchedOutputs(ROi)[docID] = semantic_distance(RSubOi, ROi)

// try the Tversky distance

- foreach class RAnyOi <> RSubOi in the same ontology
    - lookup RAnyOi in the Output index
    - if found
        - foreach matched docID
            - listMatchedOutputs(ROi).Add(docID)
            - listMatchedOutputs(ROi)[docID] = Tversky_distance(RAnyOi, ROi)

- if listMatchedOutputs(ROi) is nil, return; //the algorithm doesn't find a match; all outputs need to be matched by the algorithm above

2 seek in all listMatchedOutputs(ROi) lists and find docIDs common to all lists // all outputs need to be matched
- make a list containing all common docIDs, let's call it listMatchedOutputs

3 if listMatchedOutputs is nil, return; // no match found

4 for each input, RIi

- listMatchedInputs(RIi) <- nil
- lookup RIi in the Input index

- if found
    - foreach matched docID // add all docIDs of the WSDL files referred to by this input
        - listMatchedInputs (RIi).Add(docID)
        - listMatchedInputs (RIi)[docID] = 0 // mark the semantic distance as 0, the best match

- for each superclass RSuperIi of RIi in its ontology
    - lookup RSuperIi in the Input index
    - if found
        - foreach matched docID
            - listMatchedInputs (RIi).Add(docID)
            - listMatchedInputs(RIi)[docID] = semantic_distance(RSuperIi, RIi)

// try the Tversky distance
- foreach class RAnyIi <> RSuperIi from the same ontology
    - lookup RAnyIi in the Input index
    - if found
        - foreach matched docID
            - listMatchedInputs (RIi).Add(docID)
            - listMatchedInputs(RIi)[docID]= Tversky_distance(RAnyIi, RIi)

5 seek in all listMatchedInputs(RIi) lists and find docIDs common to all lists
- make a list containing all common docIDs, let's call it listMatchedInputs

6 seek in listMatchedOutputs and listMatchedInputs all common docIDs

7 for each docID in the final list, calculate the match factor as the average over the semantic distances of all matched Inputs and Outputs

8 order the final list by the matching factor, increasingly (the lower the better)
- in case of equality the docID that also matches the preconditions and/or effects takes priority

9 return the WSDL file corresponding to the first docID in the final list

Algoritmul incearca mai intai o potrivire textuala (sintactica), apoi face o prima incercare semantica, considerand subclasele/superclasele directe. In cele din urma, incearca o potrivire semantica pe baza similaritatii Tversky. Aceasta ordine asigura si ordinea finala a rezultatelor, adica cele mai bune potriviri apar primele.

8. CONCLUZII

- In aceasta lucrare am inceput prin a prezenta cum ar putea arata un sistem bazat pe agenti software si servicii web care comunica intre ele utilizand concepte din diferite ontologii. Acest sistem are aplicabilitate practica directa in medicina, asa cum am sugerat prin scenariul din sectiunea 2.1.
- Apoi am aratat ca este logic a incerca o unificare a agentilor si serviciilor, pentru ca acestia prezinta aceleasi provocari:
  - Trebuie sa fie adnotati semantic pentru a putea beneficia de interoperabilitatea lor semantica
  - Trebuie sa fie inregistrati in anumite baze de date
  - Trebuie sa fie descoperiti semantic
  - Trebuie sa fie consumati, adica sa comunice intre ei

- Am propus extinderea descrierilor serviciilor in standardul FIPA prin adaugarea adnotarilor IOPE (inputs, outputs, preconditions si effects), tot asa cum pentru servicii web exista standardul SAWSDL care propune adnotarea descrierilor WSDL cu concepte IOPE bazate pe ontologii. Prototipul nostru arata faptul ca procesul de cautare este imbunatatit enorm ca acuratete prin adaugarea adnotarilor IOPE, in acelasi timp ramanand scalabil si eficient (cu alte cuvinte adnotarile IOPE nu afecteaza semnificativ eficienta cautarii).

- Apoi am explorat posibilitatea inlocuirii registrelor bazate pe UDDI cu un motor de cautare semantic, care are urmatoarele responsabilitati:
  - inregistrarea (indexarea si stocarea) serviciilor web pe baza adnotarilor semantice
  - descoperirea (cautarea) semantica a serviciilor web, bazata pe distanta semantica intre concepte
- De asemenea, am extins domeniul acestui motor de cautare la agenti software, astfel incat descrierile acestora sa poata fi transformate in descrieri WSDL cu adnotari semantice, iar aceastea indexate si stocate in baza de date.
- Am prezentat arhitectura acestui motor de cautare semantic si implementarea lui cu unelte open source. Baza de date care stocheaza descrierile serviciilor si ontologiile este implementata cu BerkeleyDB (baza de date nerelationala), bazat pe B-trees, structuri de date foarte eficiente pentru implementarea indecsilor de care are nevoie un motor de cautare puternic. Spre exemplu, gasirea unei inregistrari intr-o tabela cu 100000 de inregistrari dureaza 20-30 milisecunde pe o masina pe care o operatie SELECT in SQL clasic pe o tabela cu acelasi numar de inregistrari intr-o baza de date relationala ar dura cu un ordin de marime mai mult.

Pentru viitor prevedem o implementare mai eficienta a acestui motor de cautare semantic. Structurile de date folosite si stocate in baza de date trebuie optimizate la maximum, astfel incat orice operatie a *Searcher*ului sa implice cat mai putine accese la disc, care reprezinta cea mai mare piedica in calea performantei.

BIBLIOGRAFIE

[1] Fabio Bellifemine, Giovanni Caire, Dominic Greenwood, *Developing Multi-Agent Systems with JADE*: Wiley Series in Agent Technology, 2007

[2] HL7 – Health Level 7, http://www.hl7.org/

[3] Yassin Chabeb, Samir Tata, and Alain Ozanne, *YASA-M: A Semantic Web Service Matchmaker*: AINA '10 Proceedings of the 2010 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 966-973

[4] FIPA DF, http://www.fipa.org/specs/fipa00023/XC00023H.html

[5] SNOMED browser: http://terminology.vetmed.vt.edu/SCT/menu.cfm

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

[7] Lucas API = SAWSDL to UDDI mapping, Thales technical note, 29 May 2006

[8] Dimitrios Kourtesis, Iraklis Paraskakis, *Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery*: ESWC'08 Proceedings of the 5th European semantic web conference on the semantic web: research and applications, pp. 614-628

[9] WADL, http://www.w3.org/Submission/wadl/

[10] Naveen Srinivasan, Massimo Paolucci, Katia P. Sycara, *An Efficient Algorithm for OWL-S Based Semantic Search in UDDI*: SWSWPC 2004, pp. 96-110

[11] Massimo Paolucci, Takahiro Kawamura, Terry Payne, Katia Sycara, *Importing the Semantic Web in UDDI*, Lecture Notes in Computer Science, 2002, Volume 2512/2002, pp. 815-821

[12] Daniela Grigori, Juan Carlos Corrales, Mokrane Bouzeghoub, *Behavioral matchmaking for service retrieval*: ICWS '06 Proceedings of the IEEE International Conference on Web Services

[13] Jena, http://jena.apache.org/

[14] Pensri Pukkasenung, Peraphon Sophatsathit, Chidchanok Lursinsap, An *Efficient Semantic Web Service Discovery Using Hybrid Matching*: Lecture Notes in Computer Science, 2010, Volume 6162/2010, 110-119, DOI: 10.1007/978-3-642-14415-8_8

[15] Jorge Cardoso, John A. Miller, Savitha Emani, *Web Services Discovery Utilizing Semantically Annotated WSDL*: Reasoning Web, Springer-Verlag Berlin, Heidelberg, 2008

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

[17] FIPA, The Foundation for Intelligent Physical Agents, http://www.fipa.org

[18] John Nash, *The Essential John Nash*: Princeton University Press, 2002, ISBN 0-691-09527-2

[19] Uche Ogbuji, *Using RDF with SOAP*: http://www.ibm.com/developerworks/webservices/library/ws-soaprdf/?loc=dwmai

[20] Bernhard Schiemann, Ulf Schreiber, *OWL DL as a FIPA ACL content language*: Proceedings of the Workshop on Formal Ontology for Communicating Agents (FOCA), 18th European Summer School of Language, Logic and Information, pages 73–80