UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

Facultatea de Automatica si Calculatoare
Catedra de calculatoare

Planul National de Cercetare, Dezvoltare si Inovare - PN II

Program Idei, 2007

Contract CNCSIS Nr. 333

Maestro: Compunerea automata a serviciilor Web folosind ontologii- Sintaza etapa 2008

Cuprins

Director proiect: **prof. dr. Ing. Ioan Salomie**

Octombrie 2008

1. Introducere

Serviciile Web fac posibila realizarea interoperabilitatii Business-to-Business (B2B) prin interconectarea serviciilor oferite de catre multipli parteneri de business pe baza unor procese de business. Aceasta interconectare a *serviciilor Web* pentru a satisface un process de business oarecare este numita *compunere de servicii Web*. Compunerea poate fi vazuta ca o agregare de *servicii Web* elementare sau compuse. Prin compunere, *servicii Web* existente sunt combinate impreuna pe baza unor reguli de compunere, pentru a satisface o cerere care nu poate fi indeplinita de catre un singur serviciu. Regulile de compunere specifica ordinea in care serviciile sunt invocate si conditiile in care un serviciu oarecare poate sau nu poate fi invocat. Cand compunem servicii, logica de business a clientului este implementata de mai multe servicii. Compunerea serviciilor Web este analoaga managementului de workflow-uri, unde logica aplicatiei este realizata prin compunerea de aplicatii autonome. Compunerea serviciilor Web permite definirea de aplicatii care cresc in complexitate prin agregarea in mod progresiv de noi componente la un nivel inalt de abstractizare. Odata cu cresterea rapida a numarului de servicii Web disponibile, a devenit practic imposibil pentru un utilizator uman sa analizeze toate aceste servicii si sa genereze manual un plan de compunere. Aparea astfel **nevoia de automatizare a procesului de compunere** a serviciilor Web. Necesitatea compunerii automate a serviciilor este justificata si de omniprezenta Internet-ului, care forteaza intreprinderile sa-si abandoneze modelele de business mostenite si sistemele invechite si sa se organizeze ca intreprinderi virtuale [15].

Proiectul Maestro are ca si obiectiv studiul si elaborarea unei teorii si a unui cadru unitar de compunere automata a serviciilor Web folosind ontologii. In aceasta etapa a proiectului am urmarit sa elaborarm modele si tehnici de adnotare semantica, strategii/tehnici de compunere automata care satisfac constrangeri de QoS. O descriere succinta a obiectivelor propuse, precum si a performantelor obtinute in cadrul acestei etape a proiectului va fi facuta in cele ce urmeaza.

2. Elaborare de modele si tehnici de adnotare semantica

Primul pas in elaborarea de modele si tehnici de adnotare semantica a fost acela de a studia si identifica principalele avantaje/dezavantaje pe care le oferea tehnicile de adnotare semantica existente la ora actuala. La ora actuala exista cateva initiative in curs de standardizare de catre W3C, atat din domeniul industrial cat si academic privind infrastructura serviciilor Web semantice, respectiv a tehnicilor de adnotare semantica: SAWSDL/ WSDL-S, OWL-S, WSMO. O scurta trecere in revista a celor 4 abordari, este facuta in cele ce urmaza.

Specificatia *SAWSDL* [13] defineste o modalitate de a adauga adnotari semantice la diferite parti ale unui document *WSDL* (ex: structurile de mesaje de intrare si iesire, interfete si operatii). Adnotarile semantice sunt referinte de la un element din cadrul documentului WSDL sau din cadrul schemei XML catre un concept dintr-o ontologie. Atributelor de extensie definite de *SAWSDL* sunt *modelReference*, *liftingSchemaMapping* si *loweringSchemaMapping*. *modelReference*, specifica asocierea intre componentele unui fisier *WSDL* si un concept definit intr-un model semantic (ex.: ontologie). Este

folosit pentru a adnota definitii de tip complex, definitii de tip simplu, declaratii de elemente, declaratii de atribute precum si *WSDL* interface, operation, faults. *liftingSchemaMapping* si *loweringSchemaMapping*, sunt adaugate declaratiilor de elemente, definitiilor de tipuri complexe si simple din cadrul schemei XML pentru a specifica mapari intre datele semantice si XML.

WSDL-S [4] a fost dezvoltat in cadrul proiectului Meteor-S [14] ce a avut ca obiectiv extinderea standardelor existente legate de WS cu tehnologiile Web-ului semantic. In cadrul proiectului a fost propusa o metoda de adnotare semantica a fisierelor WSDL. Pornind de la premisa ca exista deja un model semantic al serviciilor Web, WSDL-S descrie un mecanism pentru a lega acest model semantic cu descrierea functionala sintactica capturata de WSDL. Folosind atribute de exensibilitate ale WSDL, pot fi create un set de adnotari pentru a descrie semantic intrarile, iesirile si functionalitatea unui serviciu Web. Prin aceasta modelul semantic este pastrat in afara WSDL-ului, facind abordarea independenta de limbajul de reprezentare ontologic. Avantajul unei astfel de abordari se datoreaza faptului ca este o abordare incrementala, construita in variful unor standarde deja existente, care se bazeaza pe instrumente si expertize deja existente. In plus, utilizatorul poate dezvolta in WSDL intr-o manierea compatibila atat aspecte la nivel operational cat si semantica serviciilor Web.

Web Service Modelling Ontology (*WSMO*) [1] este un model conceptual pentru servicii Web semantice care defineste patru componente principale: ontologii, servicii Web, scopuri (goals) si mediatori. Dezvoltat in principal de DERI, WSMO vine cu un *limbaj de modelare a serviciilor Web* (WSML), un *limbaj care furnizeaza o sintaxa formala si semantici* pentru WSMO [2] precum si un *mediu integrat pentru executie* (WSMX).

OWL-S[2] este o ontologie generica pentru servicii Web care permita descrierea proprietatilor si capabilitatilor serviciilor Web intr-o forma interpretabila de computer, neambigua. OWL-S ofera un cadru generic pentru a descrie orice tip de serviciu Web. In OWL-S serviciile sunt clasificate in doua categori: servicii primitive (servicii elementare) si servicii complexe (servicii compuse din mai multe servicii primitive). O ontologie OWL-S consta din trei sub-ontologii: *service profile*, *service process* si *service grounding*.

In ceea ce priveste mecanismul de adnotare doar specificatia SAWSDL[5], respectiv WSDL-S ofera un mecanism de adnotare semi-automa, in celelalte adnotarea efectuandu-se manual. Pornind de la aceste premise am urmarit elaborarea/dezvoltarea unui unelte care adnoteaza in mod automat fisiere WSDL cu concepte sematice furnizate de o ontologie de domeniu. Utilizatorului are posibilitatea sa aleaga varianta de adnotare propusa de sistem sau, sa adnoteze manual in cazul in care considera solutia oferita de sistem ca nesatisfacatoare.

SAWS- Abordarea noastra de a adauga semantica la fisiere WSDL. Primul pas in procesul de adnotare l-a reprezentat o clasificare a serviciilor Web (o determinare a domeniului din care face parte serviciu). Al doilea pas il constituie selectarea unei ontologii corespunzatoare domeniului din care face parte serviciu. De obicei ontologiile de domeniu sunt selectate manual de catre utilizator, deoarece detectia automata a unei ontologii de domeniu corespunzatoare implica un efort computational mare. Al treilea pas il reprezinta procesul de adnotare efectiv, prin conectarea metadatelor care descriu

functionalitatea serviciilor Web la concepte din ontologie. Procesul de adnotare in SWAS consta din trei pasi consecutivi. In primul pas, fisierul WSDL este adnotat cu referinta la ontologia selectatat si la limbajul de descriere SAWSDL. In al doilea si al treilea pas elementele WSDL, intrarile, iesirile si operatiile sunt adnotate cu concepte din ontologie. Rezultatul procesului de adnotare este un fisier WSDL imbogatit cu noi taguri si atribute. Pentru a adnota elementele WSDL, intrarile si iesirile cu concepte ontologice, am definit un algoritm care calculeaza similaritate dintre doua stringuri folosind distanta *Levenshtein,* iar pentru a adnotarea operatiilor WSDL am folosit distanta *Levenshtein* si parametrii de intrare/iesire anteriori adnotati. Distanta *Levenshtein* este o metrica folosita pentru a masura nivelul de similaritate semantic intre doua siruri. Mai exact distanta dintre cele doua siruri se defineste ca fiind numarul minim de operatii de inlocuire (insertie/stergere/substitutie element) necesare pentru a transforma unul dintre siruri in celalalt.

Adnotarea elementelor, intrarilor, iesirilor WSDL. Pentru a adnota intrarile, iesirile dintr-un fisier WSDL, am extins limbajul WSDL cu tag-ul<*saws:ModelReference*>. Procesul de anotarea consta din doi pasii:

(i) Pentru fiecare element, parametru de intrare/iesire se traverseaza arborele ontologic si se efectueaza o comparative intre acel element particular si conceptul current din ontologie (pe baza algoritmul care calculeaza similaritate dintre doua stringuri).

(ii) Pe baza comparatiei anterior efectuate pentru fiecare element WSDL se genereaza o lista cu scoruri de similaritate. Lista este ordonata descendent - prima valoare din lista are similaritatea cea mai mare, si in marea majoritate a cazurilor corespunde conceptului cel mai apropiat din ontologie, relative la elementul, intrarea sau iesirea comparata.

Adnotarea operatiilor WSDL. Pentru a adnota operatiile dintr-un fisier WSDL am extins limbajul WSDL cu tagul<*saws:Operation*>. Operatiile sunt adnotate folosind distanta *Levenshtein* si parametrii de intrare / iesire annotati la pasul anterior. Procesul de adnotare traverseaza asccendent aborele ontologic pentru a determina conceptul parinte corespunzator conceptelor ontologice cu care au fost adnotate intrarii/iesiri corespunzatoare operatiei WSDL care se adnoteaza. Daca conceptul parinte exista deja in set, se incrementeaza numarul de apariti al acestuia. La final operatia WSDL va fi anotata cu conceptul ontological care are cel mai mare numar de aparitii.

Annotation of preconditions, postconditions and effects. Pentru adnotarea preconditilor, postconditiilor si efectelor am folosint Weka framework [14].

O descriere mai detaliata a tehnicilor de adnotare elaborate poate fi gasita in [6]

3. Elaborarea modelului extensibil de QoS

Model extensibil de calitate a serviciilor, este folosit pentru a permite selectia dintre serviciilor web disponibile a acelor care compuse satisfac cel mai bine cerintele non-functionale ale clientiilor. Modelul extensibil de calitate a serviciilor propus este compus dintr-un model generic, la care se pot adauga criterii suplimentare, in functie de domeniul problemei. Modelul generic ia in considerare trei criteria de QoS: timp, cost si calitate.

- *Timpul* este reprezentat de catre *timpul de raspuns* (*RT*) si este diferenta dintre timpul la care este trimisa o cerere catre serviciul web si timpul la care aplicatia client primeste un raspuns.
- *Costul* este compus din *costul de intretinere* (*MC*) si *costul de executie* (*EC*). Costul de intretinere este rezultatul unui contract pe termen lung cu distribuitorul unui serviciu si reprezinta o taxa care este platita pentru ca utilizatorul ca poata accesa serviciul. Costul de executie este suma de bani pe care o plateste clientul pentru a executa un serviciu, si se plateste de fiecare data cand clientul executa serviciul respectiv.
- *Calitatea* este un criteriu compus, reprezentat de *disponibilitate* (*A*) si *grad de incredere* (*R*). Disponibilitatea este procentul de timp in care un serviciu poate fi accesat si gradul de incredere este procentul de cereri care sunt executate cu succes. Gradul de incredere se calculeaza astfel:

$$R = 1 - rataErori$$

unde *rataErori* este raportul dintre numarul de executii care au esuat si numarul de executii programate.

In functie de cerintele domeniului problemei, acest model poate fi extins prin adaugarea unor noi criterii de calitate, cum ar fii: precizia, intarzierea de retea, intazierea de stocare, increderea utilizatorului, etc. Aceasta posibilitate de a extinde setul de criterii de calitate a serviciilor, face ca modelul nostru sa fie foarte flexibil.

Colectarea informatiilor de calitate a serviciilor este realizata de catre un broker. Acest lucru inseamna scutirea de un efort computational in plus la partea de client.

4. Elaborare de strategii si tehnici de compunere automata

Exista cateva criteriile de care trebuie sa tinem cont atunci cand elaboram strategii/tehnici de compunere automata:

Corectitudine- algoritmul trebuie sa garantezeca serviciul compus indeplineste intr-adevar scopul utilizatorului si satisface constrangerile impuse de catre acesta

Toleranta la erori- sa capabil sa se adapteze singur situatiei cand un serviciu implicat in compunere este indisponibil

Flexibilitate - sa necesite minim de efort uman din partea utilizatorului atunci cand scenariul de compunere se schimba

Scalabilitate- sa fie capabil sa lucreze bine cu un set mic de servicii dar si cu un numar foarte mare de servicii.

Functionalitate- sa acopere intregul proces de compunere a serviciilor Web (analiza cerintelor, crearea serviciului compus si rularea sa).

Optimizari specificate de utilizator- sa permita utilizatorilor sa specifice citerii de optimizare; exemplu: un utilizator vrea sa gaseasca solutia cea mai ieftina sau o solutie care foloseste doar servicii din tara sa

Securitate-sa furnizeze mecanism corespunzator de control al accesului; autentificarea si integritatea mesajelor, asigurarea confidentialitatii

Pronind de la aceste premise am urmarit sa dezvoltam tehnici/ strategii de compunere care sa satisfaca cat mai bine aceste criterii. Tehnicile de compunere dezvoltate folosesc ca si formalism de reprezentare calculul fuentiilor. In cele ce urmeaza vom face o scurta trecere in revista a acestui formalism, precum si a strategiilor de compunere dezvoltate.

4.1. Calculul fluentiilor

Calculul fluentilor [8,9,10,11,12] este o teorie axiomatica de actiuni ce permite exprimarea formala a domeniilor dinamice cu ajutorul unor predicate logice de ordinul intai. Calculul fluentilor are o aplicabilitate pronuntata in domeniul inteligentei artificiale, deoarece permite agentilor sa rationeze asupra actiunilor lor**,** oferindu-le posibilitatea de a stii cand pot sa efectueze anumite actiuni si ce efecte au aceste actiuni asupra lumii, respectiv sa isi actualizeze perceptia lor interna despre modelul lumii**.** Elementele de baza ale calculului fluentilor sunt: actiuni, fluentii, stari si situatii. Un fluent este o componenta atomica care descrie o stare ce se poate modifica in decursul timpului. O stare reprezinta o colectie de fluentii care se refera la o stare stabile (o stare care nu este in tranzitie). Actiunile sunt actiuni elementare care refere lucruri ce pot schimba starea lumii. O situatie este o secventa de actiuni. Semantura de baza a calcului fluentiilor consta dintr-un set finit de functii si predicate binare:

(i) *Functions into action sort*:
***S*$_0$**: -> *SIT*;
Do: *ACTION* x *SIT* -> *SIT*

(ii) *Functions into state sort*:
SIT: -> *STATE*;
Ø: -> *STATE*;
○: *STATE* x *STATE* -> *STATE*

(iii) *Predicates*:
Poss: *ACTION* x *STATE;*
KState: *SIT* x *STATE*

unde S_0 reprezinta situatia intitiala, *Do*(*a*, *s*) este situatie obtinuta prin executia actiunii *a* in situatia *s*. Termenul *State(s)* reprezinta starea in situatia *s*. Constanta *Ø* reprezinta o stare goala, si anume o stare in care nici un fluent nu este adevarat. Functia ○ mapeaza doua stari intr-o noua stare in care fluentii ambelor argument sunt pastrati. Predicatul binar *Poss* defineste preconditia unei actiuni *a* intr-o stare. Predicatul *KState* defineste o relatie intre o situatie si toate starile care sunt considerate posibile in acea situatie. Intr-o stransa relatie cu predicatul *KState* sunt macrourile *Knows* (*f, s*), *Kwhether* (*f, s*) si *KnowVal* ($\vec{x}$, *f, s*). *Knows* (*f, s*) denota ca formula f este cunoscuta in stiuatia s (ex: *Knows* (*Owns* (*AirTicket*), *s*))). *Kwhether* (*f, s*) indica daca valoare de adevar a lui f este cunoscuta (*Kwhether* (*f, s*) = *Knows* (*f, s*) $\vee$ *Knows* ($\neg$ *f, s*)), in timp ce *KnowVal* ($\vec{x}$, *f, s*) indica daca valoare functional a lui f este cunoscuta.
Teoria calculului fluentiilor este bazata pe urmatorul set de axiome:

- Σ_f, axiome fundamentale ale calculului fluentiilor;
- Σ_{una}, un set de axiome de unicitate a numelor pentru fluentii;
- Σ_{init}, o axioma pentru a definitii situatia initiala;
- Σ_{poss}, un set de axiome de preconditii pentru actiuni;

- Σ_{aux}, un set de axiome auxiliare incluzand constrangeri de domeniu
- Σ_{sua}, un set de axiome de actualizare a starii pentru a specifica efectele unei actiuni

Calculul fluentiilor furnizeaza formalismul matematic a metodei de programare logica FLUX. Un program FLUX consta din urmatoarele componente: P_{kernel}, P_{domain} and $P_{strategy}$. P_{kernel} este un set de reguli de administarea a constrangerilor impreuna cu clauze care modeleaza axiomele fundamentale ale calculului fluentiilor. P_{domain} include axiome de preconditii pentru actiuni, axiome de actualizare a starii, constangeri de domeniu si stare de cunoastere initiala, in timp ce $P_{strategy}$ *specifica* o strategie particulara conform caruia actioneaza un agent. FLUX furnizeaza constructii logice pentru a asambla actiuni primitive in actiuni mai complexe. O actiune complexe in FLUX include actiuni concurente, conditionale, secventiale si non-deterministice. Flux are un bun comportament computational legat de faptul ca foloseste progresia ca si metoda de inferenta.

4.2. Strategii de compunere automata in FLUX

In abordarea noastra problema compunerii de servicii este vazuta ca si o problema de planificare in calculul fluentiilor. Exista doua motive pentru care noi am argumentat in favoarea definirii compunerii automate de servicii ca si o problema de planificare in calculul fluentiilor. In primul rand atat problema planificarii cat si problema compunerii cauta o secventa ordonata de operatii care pot realiza scopul pornind de la o stare intiala. In al doilea rand atat domeniul planificarii cat si cel al compunerii lucreaza cu task-uri. Task-urile din domeniul AI planning sunt reprezentate de actiuni, in timp ce task-urile din domeniul compunerii de servicii sunt operatiile serviciilor web. Analog actiunilor din domeniul AI planning, operatiile serviciilor Web au intrari, iesiri, preconditii si efecte, ceea ce le fac potrivite pentru a fi utilizate in algoritmi de planificare.

O problema de planificare in calculul fluentiilor poate fi descrisa, la fel ca si in planificare clasica, printr-o tuple de forma (S, G, A, PE) unde:

- S este o descriere a starii initiale a lumii care spune ceea ce este considerat ca fiind adevarat respective fals.
- G este o descriere a scopului ce va fi realizat.
- A este o descriere a tuturor actiunilor posibile care pot fi excutate pentru a realize scopul.
- PE este un set de preconditii si efecte.

In calculul fluentiilor preconditiile sunt codificate in axiome de preconditii pentru actiuni. Axiomele de preconditii pentru actiuni specifica in ce conditii este posibil sa se execute actiune *a* in stare *s*. Efectele in calculul fluentiilor sunt condificate in axiome de actualizare a starii. Axiomele de actualizare a starii specifica schimbarile obtinute ca si rezultat al executiei actiunii *a* in starea *s*. *Efectele* pot fii fie positive ($effects^+$) fie negative ($effects^-$). $effects^+$ reprezinta setul de fluenti care vor fi adaugati la stare, in timp ce efectele negative reprezinta setul de fluenti care vor fi mutati din stare. Iesire unei problem de planificare este o situatie, exprimata ca o secventa de actiuni, care poate fi executata pentru a realize scopul dorit. Pentru procesul de planificare am dezvoltat trei strategii. In cele ce urmeaza vom prezenta sumar una dintre cele doua strategii (multi-stage, forward-chaining algorithm). In aceasta abordare, actiunile selectate intr-un stagiu pot fi executate in paralel. Algoritmul de compunere porneste de la o stare initiala Z_0

(intrarile specificate de utilizatori codificate ca o lista de fluenti) si incerca sa ajunga intr-o stare Z in care scopul este satisfacut (stare care contine toate iesirile specificate de utilizator). In fiecare stagiul al algoritmului sunt selectate toate actiunile (serviciile) care reprezinta un subset al parametrilor de intrare furnizatii in interogare de catre utilizator. In orice stagiu intermediar, iesirile generate in stagiul anterior impreuna cu parametrii de intare furnizatii initial de catre utilizator sunt consideratii ca si intrari curente. Pentru toate actiuniile selectate intr-un stagiu sunt evaluate preconditiile corespunzatoare actiuniilor respective. Actiunea devine parte a planului doar daca preconditiile sale sunt satisfacute. Odata ce actiune devine parte din plan, efectele sale vor fi aplicate aupra starii curente detrminand o tranzitie de stare. Algoritmul se sfarseste atunci cand toti parametrii de iesire specificati de utilizator in interogare sunt obtinuti. In aceasta etapa sunt eliminate serviciile redundante care nu produc parametrii de iesire. Mai exact, eliminam acele servicii care nu sunt solicitate de catre utilizator sau serviciile ale caror iesiri nu sunt solicitate ca si intrari de catre alte servicii implicate in planul de compunere.

O descriere mai detaliata a tehnicilor/strategiilor de compunere elaborate poate fi gasita in [7].

Referinte

1. C. Feier, D. Roman, A. Polleres, ”Towards intelligent Web services: The Web Service Modeling Ontology (WSMO)”, *In Proc. of the Int'al Conf on Intelligent Computing (ICIC)* , Hefei, China, August 23-26, 2005
2. D. Martin, M. Paolucci, S. McIlraith, “Bringing semantics to Web services: The OWL-S approach”, *First International Workshop on Semantic Web Services and Web Process Composition* (SWSWPC 2004), San Diego, CA, 2004.
3. D. Martin, M. Paolucci, and M. Wagner,” Towards Semantic Annotations of Web Services: OWL-S from the SAWSDL Perspective, *In OWL-S Experiences and Future Developments Workshop at ESWC* 2007, June 2007, Innsbruck, Austria
4. John Miller, Kunal Verma, Preeda Rajasekaran, Amit Sheth, Rohit Aggarwal, Kaarthik Sivashanmugam,” WSDL-S: Adding Semantics to WSDL - White Paper," Innsbruck, Austria,July 15, 2004
5. A. Patil, S. Oundhakar, A. Sheth, K. Verma, “METEOR-S Web service Annotation Framework”, Proceeding of the World Wide Web Conference, July 2004
6. I. Salomie, V.R. Chifu, Ioana Giurgiu,Mihai Cuibus, *SAWS: A Tool for Semantic Annotation of Web Services*, AQTR 2008- IEEE International Conference on Automation, Quality and Testing, Robotics AQTR 2008 - THETA 16th edition -, Cluj-Napoca, Romania, 22-24 May 2008 , ISBN978-1-4244-267.
7. I. Salomie, V. R. Chifu, Ioana Harsa, Marius Gherga, *Towards Automated Web service Composition with Fluent Calculus and Domain Ontologies* ,The 10th International Conference on Information Integration and Web-based Applications & Services (iiWAS2008) , Linz (Austria), November 2008, Appears in ACM proceeding, ISBN 978-1-60558-349-5
8. Thielscher, M.: FLUX: A logic programming method for reasoning about agents. In: Theory and Practice of Logic Programming, Cambridge University Press (eds.), 2005.
9. Thielscher, M.: Introduction to the fluent calculus, In: Electronic Transactions on Artificial Intelligence, 1998.

10. Thielscher, M.: Programming of Reasoning and Planning Agents with FLUX, In: Proc. of the 8th International Conference on Principles of Knowledge Representation and Reasoning (KR2002*)*, 2002.
11. Thielscher, M.: Handling Implication and Universal Quantification Constraints in FLUX, Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP11), 2005.
12. Thielscher, M.: The fluent calculus: A specification language for robots with sensors in nondeterministic, concurrent, and ramifying environments, In: Technical Report CL-2000-01, Artificial Intelligence Institute, 2000
13. K. Verma and AmitP. Sheth, "Semantically Annotating a Web Service", *IEEE Internet Computing*, Volume 11 (2), pp. 83-85, March/April 2007.
14. Eibe Frank, Mark A. Hall, Geoffrey Holmes, Richard Kirkby, Bernhard Pfahringer, Ian H. Witten, and Leonhard Trigg. "Weka - a machine learning workbench for data mining", In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 1305-1314. Springer, 2005
15. Heuvel, W.J. van den, Maamar, Z. (2003), Moving toward a framework to compose intelligent Web services, *Communications of the ACM*, 46(10), 103-109.