Universitatea din Pitești

Facultatea de Matematică și Informatică

Teza de doctorat

Baze de cunoștințe. Interfețe pentru interogare

- Rezumat -

***Doctorand**: Mohammed Saeed Ali Amer*

***Coordonator Științific:** Prof. univ. dr. Tudor Bălănescu*

- 2018 -

Rezumat

Pornind de la necesitatea identificării stadiului cercetărilor în domeniul interogării bazelor de cunoştinţe şi formularea de alternative pentru îmbunătăţirea performanţelor sistemelor expert, prezenta teză de doctorat este structurată în şapte părţi (capitole) şi bibliografie.

Prima parte, cu rol introductiv, analizează modelul DIKW (Data-Information-Knowledge-Wisdom) și prezintă idei filosofice şi tehnice privind nivelul sistemelor inteligente de reprezentare și procesare a cunoașterii. Se propune stabilirea nivelului de încadrare în una din categoriile D, I, K, respectiv W, folosind patru reguli. În finalul acestei părți este descrisă, pe scurt și structura tezei.

Al doilea capitol propune un model arhitectural, suficient de general pentru sistemele de reprezentare şi procesare a cunoaşterii (RPC), precum şi principalele activităţi care preced implementarea modulelor componente: achiziţia (§2.2), preprocesarea (§2.3), stocarea şi regăsirea cunoşterii, procesarea interogărilor (§2.4), motoare inferenţiale specifice sistemelor RPC (§2.5), interfeţe utilizator (§2.6) şi studii de caz (§2.7). Pentru ilustrarea practicilor şi implicaţiilor generate de utilizarea anumitor unelte şi modele sunt descrise cinci studii de caz, astfel: platforme şi lumi Cyc (§2.7.1), dezvoltarea aplicaţiilor folosind CLIPS (§2.7.2), mediul de programare bazat pe reguli JESS (§2.7.3), realizarea interfeţelor grafice în PROLOG (§2.7.4), sisteme RPC în educaţie (§2.7.5).

Cele mai importante rezultate au fost prezentate la workshopul [55] și trimise spre publicare, în context aplicativ în domeniul medical, la revista B.R.A.I.N [18].

Capitolul al treilea este dedicat metodelor declarative şi metodelor procedurale utilizate în reprezentarea faptelor şi aserţiunilor, respectiv proceselor. Modelele utilizate în reprezentarea cunoaşterii au la bază relaţii, baze de cunoştinţe, tipologii ale interogărilor (§3.1), modele orientate-obiect (§3.2), modelarea folosind concepte şi grafuri conceptuale (§3.3) şi reţele semantice (§3.4). Sunt descrișii doi algoritmi de interogare bazați pe grafuri. Se studiază proprietățile acestora. Cele mai importante rezultate au fost publicate în lucrarea [58].

Al patrulea capitol porneşte de la necesitatea modelării prin tehnici soft computing a cunoaşterii imprecise, incomplete sau aproximative (§4.1) şi analizează arhitecturi ale sistemelor RPC bazate pe mulţimi şi logică fuzzy (§4.2), numere şi logică intuiţionistic-fuzzy (§4.3), mulţimi neutrosofice, numere neutrosofice de tip a+bI și TIF, inclusiv a+bT+cI+dF și logică neutrosofică (§4.4). Sunt ilustrate modele și algoritmi la numere, grafuri, reţele, diferite hărți conceptuale și sisteme RPC. Rezultatele din acest capitol au fost publicate în [18], [59] și [63].

Capitolul al cincilea tratează mecanisme Markup pentru descrierea cunoaşterii, dar şi pentru interfeţe prin voce. Astfel, se analizează utilizabilitatea şi eficienţa, în cotext RPC, a următoarelor abordări: SGML/XML (§5.1), extensiile RDF (§5.2), modelarea bazată pe stări - SCXML (§5.3), Voice XML (§5.4). În §5.5 se analizează tehnologiile Java [131] pentru procesarea documentelor Markup (JAXB, JAXP, XMLBeans). Din categoria extensiilor RDF (the Resorce Description Framework), în contextul sistemelor RPC sunt utile dezvoltările: CKML (Conceptual Knowledge Markup Language), DLML (Description Logic Markup Language) şi OML (Ontology Markup Language). Alte abordări se bazează pe OIL (Ontology Interface Layer) şi DAML (DARPA Agent Markup Language). Dintre uneltele de dezvoltare ontologică, cele mai utilizate sunt: DUET (DAML UML Enhanced Tool), UBOT, Protégé şi Ontolingua. Un exemplu de procesare folosind descrieri în limbaj natural este ilustrat folosind SCXML. Rezultatele din acest capitol au fost publicate în lucrările [61] și [62].

Ultima parte (a şaptea) este dedicată concluziilor finale şi direcţiilor viitoare de dezvoltare ale unor modele și tehnici din teză.

1. INTRODUCERE

1.1 Date. Informaţii. Cunoştinţe. Nivelul de inteligență al sistemelor RPC

În vederea descrierii formale a unor modele de *reprezentare* şi *procesare* a *cunoştintelor* (RPC), trebuie clarificaţi anumiţi termeni precum: dată, cunoaştere şi informaţie. Kurfess [44], urmăreşte lanţul "*data* ➔ *knowledge* ➔ *information*" şi explică diferenţele folosind definiţiile existente în literatură. Astfel, "datele" sunt *piesele de bază* ale oricărei *reprezentări a cunoaşterii*. Ackoff [1] a adăugat o nouă interpretare și a propus modelul DIKW, bazat pe o viziune de conectare-înțelegere. Prin înțelegerea principiilor, se obține un nou nivel: înțelepciunea. În principal, "înțelepciunea este o abstractizare a cunoașterii, care este ea însăși o abstractizare a informațiilor, care este ea însăși o abstractizare a datelor", conform lui Swetnam și alții [98], care au definit nivelele inteligenței unui sistem.

Modelul DIKW urmăreşte lanţul "data ➔ information ➔ knowledge ➔ wisdom" [60, 77]. Astfel, (1) *informaţia este definită pornind de la date*, (2) *cunoaşterea este definită pornind de la informaţii*, iar (3) *înţelegerea profundă are la bază cunoaşterea*.

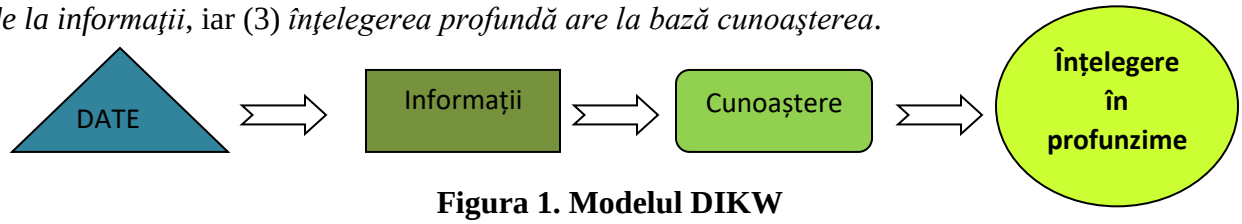


Figura 1. Modelul DIKW

Cunoaşterea poate fi informală sau tacită (implicită) sau formală (explicită). Cunoaşterea implicită există în mintea omului, dar este dificil de formalizat, de comunicat, de copiat sau *furat* şi este rezultatul *experienței, acţiunii şi revelaţiei* **individual**e. *Cunoaşterea explicită* există independent de individ, dar poate fi formalizată, distribuită, copiată, procesată, stocată și este uşor de *furat* şi este rezultatul *aplicării unor principii, proceduri şi procese asupra conceptelor*.

Definiţiile cunoaşterii se completează unele pe altele, iar unele sunt mai utile din punct de vedere practic [74]. O definiţie cu mare impact a fost dată de Hobbes [34], prin care sunt evidenţiate patru proprietăţi necesare ale cunoaşterii: (p1) *integrarea conceptelor*, (p2) *identificarea conceptelor* – prin nume, (p3) *utilizarea numelor în crearea de propoziţii*, (p4) *probarea propoziţiilor din punct de vedere al validităţii*.

Aceste consideraţii reprezintă o *motivaţie importantă în utilizarea logicii formale în reprezentarea cunoaşterii*. Alte definiţii fac apel la existenţa asociaţiilor dintre concepte, precum în modelul reţelelor semantice (Vygotsky [113]). Elementele comune ale teoriilor existente sunt [74]: 1) *Cunoaşterea este compusă din piese elementare*, numite *concepte*; 2) Conceptele se află în *relaţii / asociaţii* unele cu altele; 3) Conceptele şi asociaţiile sunt *părţi ale unor structuri* a căror dinamică se stabilizează în timp.

Din această perspectivă, *cunoaşterea* poate fi *factuală* (prin descrierea atributelor conceptelor), respectiv *procedurală* (prin descrierea modului de evoluţie a structurilor). Mai precis, *cunoaşterea declarativă descrie ceea ce se cunoaşte despre problema pe care trebuie să o rezolvăm, în timp ce cunoaşterea procedurală ne indică paşii de parcurs pentru a obţine soluţia problemei*.

Atât cunoaşterea declarativă cât şi cea procedurală pot fi specifice domeniului problemei sau au la bază elemente ale cunoaşterii comune specifice fiinţei umane. Printre procesele

specifice abordării procedurale se numără şi cele de natură euristică. De asemenea, în anumite contexte, se poate vorbi de ***meta-euristici*** sau ***meta-cunoaştere***, adică o *cunoaştere de grad superior* care permite experţilor să aleagă din strategiile de rezolvare a problemelor, pe cele mai eficiente.

Deoarece pot fi elaborate mai multe sisteme de reprezentare a cunoaşterii, se pune problema evaluării calităţii sistemelor RPC specifice [60]. Unui sistem RPC trebuie să-i fie evaluate următoarele proprietăţi: 1) **reprezentabilitatea** – *abilitatea de a permite reprezentarea tuturor formelor de cunoaştere specifice domeniului şi necesare rezolvării problemei*; 2) **deductibilitatea** – *abilitatea de a permite evoluţia structurilor* (obţinerea de noi structuri pornind de la cele existente); 3) **eficacitatea** – *abilitatea de a permite alegerea celui mai potrivit mecanism evolutiv/deductiv pentru a obţine soluţia cu cel mai mic efort*; 4) **autoînvăţarea** – *abilitatea de a produce cunoaştere nouă prin metode automate ori de câte ori este posibil*. 5) **nivelul de inteligență**.

Referitor la piesele de bază ale oricărui sistem RPC, este clar că 1) faptele sunt reguli de derivare fără premisă, 2) date sunt reguli de derivare fără premise și 3) este posibil să se presupună că datele pot fi reguli cel puțin într-un caz. Prin urmare, modelul DIKIW poate fi aplicat sistemelor inteligente RPC.

Pornind de la analiza efectuată în [98], din punct de vedere calitativ, nivelul de inteligență al unui sistem RPC, cu pragul de încadrare de 90%, poate fi evaluat folosind regulile [60]:

1. *Un sistem RPC poate funcționa la nivelul D (date) dacă indicatorul D depășește 90* de puncte atunci când se evaluează capacitatea sistemului RPC din punct de vedere al: asigurării inputului pe baza măsurătorilor (date de intrare) (20p), capabilității de a stoca intrările într-un container (depozit, baza de date) (20p), capabilitatea de a permite adăugare sau revizuire a datele stocate (20p), facilitează utilizarea secvențelor de date (20p) și oferă capabilități specifice bazelor de date pentru gestiunea depozitului datelor (20p).

2. *Operabilitatea la nivel I (informație) este asigurată dacă se obține mai mult de 90p* atunci când se verifică dacă: sunt permise interogări (25p), se asigură rezolvarea interogărilor prin accesarea mai multor elemente ale bazei de date (25p), sistemul este capabilă să furnizeze un model al tendinței datelor (10p), sistemul este capabil să coreleze datele sau să determine corelații între fragmente de date, prezicând date prin interpolare stărilor (10p), sistemul poate combina datele din mai multe baze de date (10p).

3. *Sistemul funcționează cel puțin la nivelul K (cunoaștere), dacă scorul este mai mare de 90p* atunci când este evaluat pentru: integrarea sau construirea de informații (20p), capabilitatea de a construi un model în funcție de informațiile recepționate (20p), include un model de cauzalitate care facilitează înțelegerea comportamentului sistemului atunci când primește unele intrări (10p), sistemul oferă predicții utile despre sistemul modelat (10p), permite utilizarea mulțimilor de valori corespunzătoare unei stări a sistemului modelat (10p) este capabil de auto-explicare (10p), permite predicția prin extrapolare (10p), este capabil să evalueze structura sistemului prin elemente neobservabile (5p), este capabil să construiască modele despre diferite sisteme (5p).

4. *Un sistem funcționează la nivelul W (înțelepciune) dacă scorul depășește 90p* atunci când se analizează dacă sistemul este capabil: să ingereze mai multe modele ale unui sistem compus (20p), să sprijine meta-modelarea (20p), să suporte experimentarea prin modificarea modelelor de intrare (10p), să optimizeze meta-modelul conform nivelurilor indicatorilor informațiilor produse (10p), să furnizeze informații de îmbunătățire a comportamentului către

stări mai bune, în funcție de un anumit indicator de performanță (10p) să sprijine îmbunătățirea, susținerea actualizării parametrilor pentru a îndeplini starea previzionată optimală (10p).

Din cele de mai sus rezultă că orice demers al cercetărilor privind interogarea bazelor de cunoştinţe are loc în cadrul mai larg al modelelor teoretice și arhitecturale ale sistemelor RPC.

2. SISTEME PENTRU REPREZENTAREA ŞI PRELUCRAREA CUNOAŞTERII

2.1. Introducere

Reprezentarea şi prelucrarea cunoaşterii (RPC) joacă un rol important în realizarea sistemelor expert ca motor al acceleării proceselor dezvoltării vieţii economico-sociale. Pentru a fi eficace, un sistem RPC adecvat unui domeniu, trebuie să îndeplinească următoarele funcţionalităţi [54]: 1) Achiziţia şi preprocesarea (în vederea structurării optimale a) cunoaşterii; 2) Stocarea şi regăsirea cunoaşterii în/dintr-o bază de cunoştinţe; 3) Procesarea cererilor; 4) Producerea de cunoaştere în urma raţionamentelor asupra cunoaşterii existente; 5) Oferirea unei interfeţe cu utilizatorul prin care se asigură introducerea şi extragerea pieselor cunoaşterii.

Pornind de la aceste cerinţe, rezultă că orice model arhitectural al unui sistem RPC trebuie să conţină cel puţin cinci componente: M1-M5 [54].

Modulul M1 preia date / informaţii şi interpretări ale utilizatorului care în urma unor procese de analiză (în vederea respingerii sau acceptării spre reprezentare – adecvanţa la domeniul problemei) va declanşa proceduri de segmentare, extragere de caracteristici şi va constitui structuri adecvate memorării în baza de cunoştinţe M2. Achiziţia cunoaşterii, proces reprezentativ al componentei M1, este tratată în patru etape: definirea şi extragerea informaţiei, conceptualizarea, formalizarea şi implementarea.

Modulul M2- *baza de cunoştinţe* conţine piese ale cunoaşterii achiziţionate de la experţi umani în legătură strânsă cu domeniul problemei şi care descriu situaţii certe, fapte reale, reguli, etc. În general, în cadrul unui sistem RPC dedicat, baza de cunoştinţe utilizează datele stocate într-o bază de date BD (specifice problemei de rezolvat), modulului M2 revenindu-i sarcina de a indica cum se procesează inteligent datele stocate în BD pe baza cunoaşterii specifice rezolvării problemelor dintr-un anumit domeniu.

Modulul M3 preia fapte din baza de cunoştinţe şi le trimite direct utilizatorului sau le trimite rezolvitorului (Modulul M4) pentru a "deduce", folosind principiul rezoluţiei, noi cunoştinţe pe care le va livra utilizatorului sau le va înregistra în baza de cunoștințe (cazul sistemelor cu auto-învățare). Interfaţa om-maşină este reprezentată de modulul M5.

Modulul M4 este motorul de inferenţă. Aceasta poate fi un program (sau un circuit integrat microprogramat) care dispune de mecanisme inferenţiale generale pentru prelucrarea cunoştinţelor folosind raţionamente din cele mai diverse. În urma raţionamentelor efectuate este posibil să se producă modificări asupra pieselor cunoaşterii. Unele reguli vor fi eliminate sau înlocuite cu altele. Practic, modulul M4 dispune de un sistem de gestiune a bazei de reguli (BR). În plus, modulul M4 trebuie să pună la dispoziţia utilizatorului informaţii privind modul în care a obţinut soluţia oferită (paşii raţionamentului).

Modulul M5 asigură interfaţa cu utilizatorul şi permite accesul la informaţiile şi cunoştinţele înregistrate sau furnizate de diverse componente ale sistemului RPC. Modulul M5 poate asigura mai multe niveluri de acces: utilizator obişnuit, administrator bază de cunoştinţe (specialist în problematica domeniului de activitate), administrator bază de date (specialist în

rezolvarea problemelor de un anumit tip din domeniul de activitate), administrator sistem RPC (informatician).

Definiția 2.1. Modelul unui sistem RPC este $S = (Q, \{M_i\}_{i=1, 2, ...5}, \{P_i\}_{i=1, 2, ..., 5}, \{I_i\}_{i=1, 2, ..., 5}, \{C_{ij}\}_{i=1, 2, ..., 5; j=1, 2, ..., 5})$, unde Q este o mulțime de variabile de stare ale sistemului, $M_{i, i=1, 2, ...5}$ sunt modulele prezentate mai sus, P_i este o mulțime de metode de procesare specifice componentei M_i, I_i este o mulțime de interfețe/protocoale care permit interconectarea modulului M_i în structura sistemului S, iar C_{ij} sunt mulțimi de componente care asigură buna funcționare a conexiunii dintre modulele M_i și M_j.

O instanţă a unui sistem RPC care achiziţionează şi procesează cunoaştere pentru rezolvarea unei probleme dintr-un anumit domeniu de activitate se numeşte *sistem expert*. Implementarea tehnică a unui sistem RPC poate face apel la diverse componente (plug-ins) care sunt independente de datele, faptele sau regulile necesare rezolvării de probleme concrete.

Deducem, că în contextul modelului arhitectural de mai sus, un sistem expert este un obiect (ca instanţă a unei clase RPC - în terminologie obiectuală), urmând ca implementarea fizică şi logică să fie specifică rezolvării unei anumite probleme.

Definiția 2.2. Un sistem expert E este o instanță a unui model S de sistem RPC.

În continuare sunt descrise principii, bune practici și tipuri de proceduri utile în implementarea modulelor M_i, i = 1, 2, ... 5.

2.2. Achiziţia cunoaşterii

Metodologia captării (achiziţiei) cunoaşterii influenţează atât calitatea cunoaşterii, dar mai ales efortul necesar (resursa umană necesară, preprocesarea informaţiilor culese, unelte hardware şi software, logistică etc.). Achiziţia cunoaşterii în contextul calităţii impactului produs de sistemul RPC presupune existența unei metodologii cu cel puţin patru faze: (**1**) planificare, (**2**) identificarea cunoaşterii, (**3**) analiza cunoaşterii identificate, (**4**) verificarea ipotezelor.

2.3. Preprocesarea cunoaşterii

Dezvoltatorul modulului de achiziţie a cunoaşterii ia în considerare trei surse majore [55]: 1) cunoaşterea stocată în medii relevante: cărţi, ziare, medii audio-video; 2) cunoaşterea obţinută prin observarea şi modelarea domeniului relevant (folosind tehnicile din secţiunea anterioară) şi 3) cunoaşterea transmisă de o comunitate relevantă (experţi, forumuri, comunităţi de socializare, clienţii dezvoltatorului unui astfel de modul).

De obicei, datele (text, imagine, audio, video, hypertext) înregistrate în timpul captării cunoaşterii de la experţi trebuie supuse unui proces de evaluare. Este posibil ca o anumită informaţie să fie incompletă (lipsesc atribute, lipsesc valori ale unor atribute), pot exista informaţii categorisite ca aberante sau influențiale (eng. *outliers*) sau chiar pot exista părţi (ale cunoaşterii) inconsistente.

Sintetic, în continuare sunt prezentate bunele practici ce trebuie aplicate de analistul pieselor cunoaşterii [55]: 1) Custodele bazei de cunoştinţe asigură disponibilitatea cunoaşterii, fiabilitatea sistemului RPC, precum şi securitatea bazei de cunoştinţe; 2) Analistul bazei de cunoştinţe menţine piesele cunoaşterii la nivel calitativ din punct de vedere al consistenţei şi integrităţii (ca urmare a aplicării unor transformări sau operaţii de actualizare); 3) “Purificarea” cunoaşterii constă în activități de detectare și corectare a pieselor dintr-o bază de cunoştinţe incorecte, incomplete, formatate (modelate) în mod necorespunzător sau redundante. În plus, se va asigura consistenţa cu piesele stocate în alte baze de cunoştinţe. Această sarcină devine

dificilă în contextul bazelor de cunoştinţe distribuite şi a noilor abordări bazate pe concepte Big Data [139]. 4) "Purificarea" cunoaşterii se realizează folosind algoritmi de "reparare" ce au la bază reguli specifice dependente de domeniul problemei de rezolvat, dificultatea implementării fiind dată de dependenţa de context a utilizării pieselor cunoaşterii stocate în baza de cunoştinţe. 5) Validarea cunoaşterii este diferită de "purificarea" cunoaşterii şi se referă doar la corectitudinea înregistrării pieselor cunoaşterii în sistemul RPC. 6) Există o mare varietate de metode şi unelte software pentru "purificarea" şi validarea datelor, dar acestea pot fi utilizate doar parţial în contextul bazelor de cunoştinţe şi in mod special doar pentru descoperirea cunoştinţelor. 7) Integrarea cunoștințelor este procesul de sintetizare a mai multor modele (sau reprezentări) într-un model comun (aceeaşi reprezentare).

2.4. Stocarea şi regăsirea cunoaşterii. Procesarea interogărilor.

Regăsirea cunoaşterii trebuie tratată prin comparaţie cu regăsirea datelor, respectiv a informaţiilor [52, 71, 72]. Sunt valabile 8 criterii asupra cărora trebuie realizată analiza [118]: *(1) potrivirea (matching), (2) procesul inferenţial, (3) modelul conceptual, (4) natura interogărilor, (5) modul de organizare, (6) modul de reprezentare, (7) natura stocării, (8) rezultatul regăsirii*.

Regăsirea informaţiilor presupune *potrivire parţială*, dar şi *cea mai bună potrivire*, utilizează un proces inferenţial inductiv, modelul conceptual este de natură stochastică (probabilistă), interogările se realizează în limbaj natural, informaţia este organizată tabelar şi indexat, reprezentarea utilizează limbajul natural precum şi limbaje de marcare. Informaţia este stocată în colecţii de documente, iar rezultatul obţinut este constituit din secţiuni (fragmente) de documente. O aplicaţie interesantă în domeniul regăsirii informaţiilor este cea referitoare la rezumarea documentelor.

Un tip aparte de regăsire a informaţiei (RI) apare în contextul întrebărilor formulate textual în sisteme întrebare-răspuns. La astfel de întrebări se cere un răspuns privind: "CINE a realizat CEVA pentru CINEVA, UNDE, CÂND şi DE CE?". Se utilizează atât tehnici de regăsirea informaţiei, dar şi metode de descoperire a cunoaşterii de-a lungul a trei etape: clasificarea întrebărilor, regăsirea informaţiei sau procesarea documentului, şi extragerea răspunsului.

2.5. Motoare inferenţiale specifice RPC

Componenta decizională a unui sistem RPC este numită *procedură decizională evolutivă* deoarece, în timpul funcţionării, la fel ca în orice proces inferenţial, se generează piese noi de cunoaștere (fapte și reguli) folosind baza de cunoștințe existente, folosind regulile specifice procesului inferențial asupra înregistrărilor din baza existentă.

Un mecanism elementar de activare a regulilor funcţionează pe baza următorilor trei paşi: Filtrare, Selecție, Executare. Algoritmul de selectare implementează, de fapt, strategia de control a sistemului RPC, deoarece determină direcţia de căutare în vederea atingerii scopului [70, 107]: înlănțuire înainte (forward links), respectiv înlănțuire înapoi (backward-links).

Pentru creşterea eficienţei motorului inferenţial, sistemele RPC pot utiliza o agendă de reguli (de exemplu motorul CLIPS [123]). Poziția din agendă (listă) are la bază un indicator numit *prioritate*, stabilit de analistul RPC, sau are la bază eficientizarea algoritmului de inferență în contextual eliminării conflictelor. Practic, este vorba de o metodă de sortare a elementelor, de obicei descrescător după indicatorul prioritate, iar la priorități egale, ordine este stabilită în raport cu strategia de eliminare a conflictelor.

2.6. Interfeţe utilizator

Interfaţa cu utilizatorul poate fi de tip text, interfaţă grafică, dar şi interfeţe audio-video care sunt în dezvoltare şi se bazează pe tehnici specifice de sinteză şi recunoaştere audio-video. Primele sisteme RPC au fost de tip întrebare-răspuns, iar conversaţia utilizatorului cu sistemul s-a realizat cu ajutorul textului. În zilele noastre, comunicarea utilizator – sistem se realizează şi prin intermediul interfeţelor grafice, inclusiv online, respectiv cu suport pentru recunoașterea gesticii.

2.7. Studii de caz

2.7.1. Platforme Cyc

Cyc este un ansamblu de platforme dezvoltate de Cycorp [126, 127], disponibile gratuit, pentru proiecte de cercetare. Cyc utilizează limbajul de reprezentare CycL, bazat pe logica predicatelor de ordinul I cu extensii asupra manipulării egalității, raţionamentului implicit, skolemizării şi câteva caracteristici ale logicii de ordinul II.

Conform Siegel et al. [81], sistemul Cyc are la bază următoarele componente şi mecanisme: baza de cunoştinţe, lumile Cyc, motorul inferenţial, interfeţe utilizator, serverul de transcripturi, partiţiile, unitatea de integrare a cunoaşterii şi interfeţele programatorului.

Interfețele cu utilizatorul sunt multiple şi adaptate nivelului de utilizare. Paginile HTML (pentru interogare, căutare, editare şi adăugare de conţinut) sunt generate dinamic. Editorul de fapte este o interfaţa şablon Java [131] care permite adăugarea de noi fapte şi editarea faptelor existente. Biblioteca de interogări este o interfaţă Java [131] care permite utilizarea de interogări preformulate, dar şi compunerea de noi interogări prin asamblarea şi editarea interogărilor preformatate şi a şabloanelor. Editorul de fapte şi Biblioteca de interogări rulează în navigatotul Cyc şi sunt implementate folosind interfaţa Cyc Java API [127] care permite unui program Java [131] extern să comunice cu o imagine Cyc.

Limbajul CycL [127] constă din specificaţii pentru *constante, formule, propoziţii* şi *predicate, termeni neatomici, constante logice* şi *conectori, variabile* şi *cuantificatori*. Regulile de bună formare folosesc noţiunea de aritate. Specificarea colecţiilor se realizează folosind #$isa, iar a indivizilor prin #$genIs. Proiectantul și implementatorul unui sistem RPC care utilizează CycL, trebuie să respecte caracteristicile relațiilor: #$genls și #$isa: Relaţia #$genls **este** *reflexivă* şi *tranzitivă*, în schimb relaţia #$isa **nu are** proprietatea de *tranzitivitate*.

2.7.2. CLIPS

CLIPS a fost proiectat pentru a permite o dezvoltare rapidă de aplicaţii care să modeleze şi să proceseze cunoaşterea folosind limbaje de programare precum C, C++, Java (Jess), PHP şi Python (PyClips) [123]. Pentru reprezentarea cunoaşterii, CLIPS foloseşte reguli (modelarea cunoaşterii obţinute prin experienţă), funcţii definite şi funcţii generice apelabile în cadrul programelor procedurale şi concepte ale modelării şi programării orientate pe obiecte precum: clase, mesageri, abstratizări, încapsulare, moştenire şi polimorfism. Definirea funcţiilor (în stilul programării procedurale) este posibilă prin intermediul regulilor sau utilizând *deffunction*.

În descrierea regulilor, respectiv a funcţiilor se pot utiliza predicate. Pentru definirea claselor se utilizează **defclass**. CLIPS utilizează **is-a** pentru a descrie relaţia de specializare. Predicate precum **superclassp** şi **subclassp** pot verifica relaţia dintre clase, iar **list-defclasses** furnizează lista claselor definite.

2.7.3. JESS

JESS (Java Expert System Shell) este un mediu de programare bazat pe reguli util dezvoltatorilor de sisteme RPC folosind Java [131] şi implementează o extensie a limbajului CLIPS [29, 123]. Versiunea JESS 7 este disponibilă şi sub platform ECLIPSE și permite implementarea de interfețe grafice mult mai prietenoase cu utilizatorul datorită atât a existenţei pachetului AWT [131], dar şi a ierarhiei de clase Swing [131].

2.7.4. Interfeţe grafice în Prolog

Limbajul Prolog [53, 135] se bazează pe o metodă de rezoluție în logica predicatelor (metoda lui Robinson) de construire a unui arbore (AND/OR) folosind căutarea înapoi (de tip *goal driven*), spre deosebire de CLIPS [136] (deci şi de Jess [29]) care înlănţuie deducţiile ţinând cont de regulile de producţie obținând soluţia prin căutare înainte (de tip *data driven*). De obicei, mediile Prolog nu sunt prevăzute cu module pentru realizarea de interfețe grafice cu utilizatorul (cu excepţia Visual Prolog). În acest caz se utilizează pachete externe specializate care comunică cu mediul Prolog prin fire (en. pipes). Pentru SWI-Prolog este recomandată utilizarea pachetului XPCE.

2.7.5. Sisteme RPC în educație

Folosind [61, 62, 89], un sistem RPC pentru educația artificială (EA) ar trebui să țină cont de patru elemente: **cunoștințele, rezolvarea problemelor**, Componente dependente de **dezvoltatori** și **administratori**, cu accent pe **Sistemele Inteligente de Tutoring**.

Conform Horvitz [36], *o interfață inteligentă de utilizator* ar trebui să ia în considerare aspectele de *imprecizie* și *incertitudine* în timpul "run-time". Acest lucru este mai important în EA datorită naturii întrebărilor formulate de către cursanți. După cum se menționează în [79], **interfețele in limbaj natural** (ILN) vor fi următoarea generație de interfețe pentru a îmbunătăți experiențele utilizatorilor. Propunerea noastră din [61, 62] se bazează pe tehnici de inteligență artificială pentru a face față impreciziei / incertitudinii și a aspectelor legate de limbajul natural, cu înțelegerea și restructurarea cunoștințelor pentru sistemele rapide de răspuns.

3. METODE PENTRU REPREZENTAREA ŞI PROCESAREA CUNOAŞTERII

3.1. Relaţii, baze de cunoştinţe, interogări

Reprezentarea cunoaşterii cu ajutorul relaţiilor poate fi privită ca transferul expertizei umane către sisteme de procesare a bazelor de cunoştinţe. Exprimarea relaţiilor dintre concepte se realizează folosind *harta conceptelor* (eng. *concept map*) şi a apărut în domeniul educaţiei, mai precis în cadrul activităţilor de învățare şi evaluare, conform Novak şi Canas [69]. *Harta conceptelor* nu trebuie confundată cu *Harta minții* (eng. *mind map*). Harta minții utilizează imagini, simboluri, coduri și alte entități pentru a forma o diagramă utilă în generarea, vizualizarea și structurarea ideilor privind un anumit subiect supus atenției [140].

Elaborarea hărților conceptuale poate fi realizată folosind aplicaţia software CMAP [124]: http://cmap.ihmc.us/cmaptools/cmaptools-download/. Hărţile conceptuale beneficiază de reprezentări grafice variate: a) *pânza de paianjăn* b) *structura arborescentă* c) *hărţi conceptuale generale* cu intrări şi ieşiri.

Există două moduri de a privi legătura dintre ***sistemele pentru reprezentarea şi procesarea cunoaşterii*** (***sisteme RPC***) şi Sistemele de Gestiune a Bazelor de Date (**SGBD**), conform Borgida [10].

Transformarea reprezentărilor relaționale în structuri care să permită interogarea inteligentă este posibilă prin regândirea informaţiei ca fiind parte a unui predicat (logica de ordinul 1). Se va obţine o bază de date îmbogăţită care constă din [65]: O *mulţime P de predicate*. Fiecare predicat este definit **true** pentru faptele asociate şi este **false**, în caz contrar; O mulţime R de predicate predefinite; O mulţime S de predicate, pentru fiecare predicat fiind disponibilă o mulţime de reguli; Mulţimile P, R şi S sunt disjuncte două câte două.

Întreaga bază de date poate fi gândită ca o mulţime de axiome, iar regula de deducţie are la bază principiul rezoluţiei. Astfel, putem avea în vedere o extindere a conceptului de *baza de date* (eng. *database:* **DB**) care devine *baza de cunoştinţe* (eng. *knowledge database*: **KDB**).

Reuniunea mulţimilor P şi R formează **componenta extensională** a bazei de cunoştinţe, iar mulţimea S, cea care permite *interogarea inteligentă prin reguli*, este **componenta intensională.**

3.2. Metode RPC orientate-obiect

În vederea proiectării şi implementării unui sistem RPC orientat-obiect trebuie parcurse următoarele patru faze: caracterizarea domeniului, modelarea orientată-obiect, formalizarea şi implementarea propriu-zisă. Odată ce a fost aprofundat domeniul problemei, modelarea orientată-obiect poate utiliza metoda **CRC** – *Clase, Responsabilităţi*, și *Colaborări*.

Structura unei cartele CRC trebuie să precizeze: `Numele clasei`, `Câmpurile` (date şi metode) clasei, precum şi `Tipurile` de entităţi cu care această clasă colaborează: tipuri simple de date, clase predefinite, clase ale proiectului. Cartelele CRC sunt portabile, nu necesită utilizarea calculatorului (dacă se lucrează cu cartonaşe), permit experimentarea abordărilor (prin utilizarea gumei şi a creionului / sau opţiunii **`Del/Ins`** în versiunea computerizată [142]).

Cea mai importantă abordare aplicabilă actualelor modelări orientate obiect este bazată pe **`UML – Unified Modelling Language`**. În UML, toate lucrurile universului în discuţie sunt modelate folosind clase. Fiecare clasă este caracterizată printr-o mulţime de operaţii (Comportament generic descris prin cod organizat sub formă de funcţii/metode) şi atribute (care descriu starea unei entităţi). Componentele unei clase pot fi *publice* (+), *private* (-) sau *protejate* (#). Codul poate fi *modificat* în clase derivate din cele iniţiale, poate fi *moştenit* sau poate fi *final* (nu mai este permisă rescrierea metodei în clasele derivate).

Asocierile dintre clase pot indica: *interacţiunea dintre clase* (un obiect al clasei **`A`** utilizează un obiect al clasei **`B`**; de exemplu un **`Abonat`** împrumută o **`Carte`**; aici clasele sunt **`Abonat`** şi **`Carte`**), *multiplicităţi* (o **`Carte`** poate avea mai multe ediţii sau copii), *roluri, direcţii de comunicare a mesajelor*, *agregare* şi *compunere* etc.

Agregarea este metoda prin care obiectelor li se permite a incorpora alte obiecte, adică implementarea "**`xPy: r`**`elaţia x este parte a lui y`". Prin urmare, componentele unui obiect pot fi atât entități primitive, cât și obiecte ale universului de discurs. Astfel se pot crea lumi obiectuale structurate folosind elemente laticiale din cele mai diverse.

Moştenirea este un mecanism specific modelului orientat obiect care permite unei clase **`A`** să moştenească câmpurile (date şi metode ale) unei clase **`B`**. Rezultă, de fapt, o ierarhizare a claselor. Considerăm că dacă **`B`** moşteneşte proprietăţile clasei **`A`**, atunci **`A`** este superclasă pentru **`B`**, iar **`B`** este o subclasă a lui **`A`**. Acest şir poate continua până la specializarea efectivă. Trecerea

de la clasă la subclasă se face prin specializare. O tehnică specifică procesării obiectuale este **polimorfismul**, care permite realizarea aceleiaşi operaţii prin algoritmi diferiţi în clase diferite.

3. 3. Concepte şi grafuri asociate

Cunoaşterea se poate înregistra în momentul în care au fost identificate conceptele, relaţiile dintre acestea şi există soluţia tehnică de implementare computerizată. Sistemele RPC bazate pe *grafuri* folosesc modele de tipul G = (V, E) unde V și E sunt mulțimi finite, nevide, de vârfuri (noduri), respectiv muchii sau arce pentru redarea legăturilor dintre noduri.

Principalele noţiuni cu care vom opera sunt: *nod*, *legătură*, *nod special* (comportament contextual), *legătură contextuală*, *reţea semantică* (simplă), *graf conceptual*, *reţele semantice multinivel* şi *nod de tip proces*.

Un *nod* este o piesă elementară (indivizibilă) a cunoaşterii (termen, entitate, concept) care este unică în cadrul modelului. Nodurile pot reprezenta obiecte generice, lucruri, evenimente, acţiuni, idei, oameni, concepte de nivel înalt (ca abstractizare) precum: persoana, eveniment sportiv, învăţare, sentimente, activităţi, figuri geometrice etc. Unităţile de cunoaştere mai cuprinzătoare sunt reprezentate prin grupuri de noduri interconectate. Cea mai mică diferenţă semantică între doi termeni va duce la existenţa a două noduri distincte. Nodurile au nume (monosemantice) care le etichetează. În afara numelui, nodurile pot avea şi alte atribute. Astfel, nodurile pot fi de mai multe categorii: *nod* (nod ordinar, nod static, entitate, concept, termen), *nod contextual* (abstract, grup, mulţime, clasă, cadru, tip), *noduri de tip data* (descriu depozite de date) şi *noduri proces* (legare, înrudire, dinamice, funcţionale, de acţiune, de condiţionare).

Legătura are rolul de a conecta două noduri. Există numai *legături binare*. De exemplu, pentru propoziţia "`Universitatea din Piteşti`" se vor crea două noduri: unul pentru "`Universitate`" şi al doilea pentru "`Piteşti`". Între aceste noduri se va plasa o legătură (de exemplu cu săgeată de la nodul "`Universitate`" la nodul "`Piteşti`" pentru a elimina ambiguitatea "`Piteşti Universitate`". O *legătură contextuală* apare între un nod contextual şi un nod specializat, de exemplu de la nodul "**Om**" la nodul "**Bărbat**" sau de la nodul "**Om**" la nodul "**Femeie**".

Grafurile conceptuale au fost iniţial introduse, în [90] și dezvoltate în [91, 92], de Sowa. pentru a descrie scheme utilizate în baze de date, dar apoi au fost utilizate în aplicaţii inteligente (inteligenţă artificială, ştiinţe cognitive etc.). În [125] este descris un standard privind grafurile conceptuale elaborat de Comitetul NCITS.T2 pentru Interschimbarea şi Interpretarea Informaţiei.

Grafurile conceptuale pot fi descrise în trei moduri: **GDF** (Graphical Display Form), **CGIF** (Conceptual Graph Interchange Format) sau **CLIF** (Common Logic Interchange Format) şi **LF** (Linear Form), oricare dintre acestea putând fi translatate în **KIF** (Knowledge Interchange Format). Putem considera că această abordare a lui Sowa [90, 91, 92] este bazată pe logica formală, mai precis pe logica predicatelor.

Reprezentarea GDF foloseşte dreptunghiuri pentru a prezenta conceptele, cercuri sau elipse pentru a introduce relaţia dintre concepte. Între relaţii şi concepte sunt inserate arce cu săgeţi. Dacă o relaţie are mai mult de două arce atunci acestea vor fi numerotate. În ***reprezentarea LF***, conceptele sunt prezentate între paranteze drepte [ ], iar relaţiile între paranteze rotunde ( ). În locul arcelor din reprezentarea grafică, în reprezentarea LF se utilizează un simbol special, de exemplu ®, ca în exemplul: [Carte] ® (Pe) ® [Masa], care modelează parţial activitatea din sala de lectură a unei biblioteci, în care locurile cititorilor reprezintă informaţii relevante în cadrul unui anumit studiu. Reprezentările GDF şi LF nu sunt potrivite pentru comunicarea dintre calculatoare, ci doar pentru comunicare interumană sau pentru

comunicare între om şi calculator. ***Reprezentarea CGIF*** utilizează etichete de identificare a conceptelor folosite în descrierea relaţiei folosind procedeul legării [125]: [Carte: *x] [Masa: *y] (Pe ?x ?y), sau mai simplu (Pe [Carte] [Masa]) fără specificarea unor instanţe generice. Este evident că reprezentările LF şi CGIF sunt echivalente din punct de vedere logic, fiind diferite doar ca stil de reprezentare.

Pentru situaţia în care reprezentarea internă a cunoaşterii nu este graful conceptual, ci este dată prin structuri de date primare, precum struct/class din limbajul C++, se poate utiliza formalismul extins **KIF**:

(există ((?x Carte) (?y Masa) (Pe ?x ?y)).

Observăm că toate descrierile de mai sus pot fi interpretare folosind calculul predicatelor:

(∃x) Carte(x) (∃y) Masa(y) Pe(x,y).

Grafurile conceptuale au fost intens studiate nu numai de Sowa [90, 91, 92], dar și de Școala de grafuri conceptuale de la Montpelier [6]. A fost arătat că structurile conceptuale introduse la Montpelier sunt reprezentări formale pentru conceptele introduse de Sowa. Lucrarea [39] prezintă o aplicație concretă a grafurilor conceptuale pentru modelarea cunoașterii medicale în timp ce [6] studiază grafurile conceptuale din perspectivă teoretică.

Definiția 3.3.1. Fie TC o mulțime nevidă de tipuri de concepte, parțial ordonată, cu c_{max} cel mai mare element. Fie TR o mulțime nevidă de simboluri relaționale, parțial ordonată, partiționată în funcție de aritatea relației în TR_1, TR_2, ..., TR_k. Aritatea unui simbol din TR_i este i. Fie Ω o mulțime nevidă de marcaje. Se presupune că * este un marcaj special. Un graf conceptual este un ansamblu GC = (C, R, E, λ):

- (C, R, E) este un graf multi-bipartit (între două noduri pot exista mai multe muchii), E este mulțimea muchiilor;
- $\lambda : C \cup R \rightarrow TC \times (\Omega \cup \{*\}) \cup TR \cup \{1, 2, ..., k\}$ este o aplicație de etichetare care acționează astfel:
  - $(\forall x)\ (\ (x \in C) \rightarrow \lambda(x) \in TC \times (\Omega \cup \{*\}))$.
  - $(\forall x)\ (\ (x \in R) \rightarrow \lambda(x) \in TR)$.
  - $(\forall x)\ ((x \in R) \wedge \text{aritatea}(x) = n \wedge ((\exists y_i \in R), (x, y_i) \in E, i = 1, 2, ...n) \rightarrow \lambda(y_i) = i)$.

Exemplul 3.3.1. TC = {A_1, A_2, B_1, B_2, C, D, c_{max}}, unde $A_1 \le A_2 \le c_{max}$, $B_1 \le B_2 \le c_{max}$, A_1 și B_1 sunt incomparabile, A_2, B_2, C și D sunt incomparabile două câte două. Fie TR = {R_1, R_2, S, ≡}, unde $R_1 \le R_2$, iar R_2, S și ≡ sunt incomparabile două cîte două. Fie Ω = {a, b, c, d}. Atunci graful conceptual Dau [19] este:

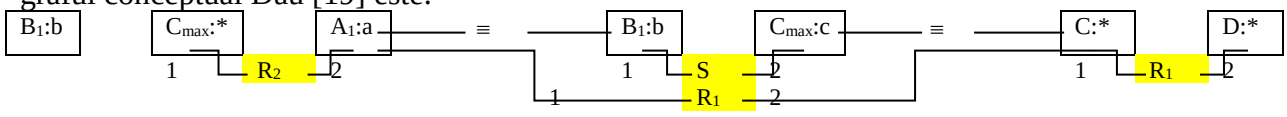


Figura 3. Reprezentare GDF cu noduri etichetate folosind aplicația λ

3.4. Reţele semantice

Rețelele semantice (RS) reprezintă o alternativă la reprezentarea cunoștințelor folosind logica predicatelor. Introduse, inițial, pentru modelarea semantică în cadrul procesării specifice limbajului natural (perioada anilor 1965-1970) au fost formalizate cu ajutorul grafurilor orientate [93].

Definiția 3.4.1. Un model RS integrează mulțimea obiectelor și mulțimea relațiilor inter-obiecte și este reprezentat prin ansamblul (N, E, R, λ, M, Σ, σ), unde:

- $N = (\Omega, A_0, R_0, M_0)$ este infrastructura statică a rețelei (Net) modelată ca un graf orientat etichetat [93], unde Ω este mulţimea etichetelor nodurilor, A_0 este o mulţime distinctă de Ω şi este mulţimea etichetelor arcelor, R_0 este o mulţime nevidă de relaţii binare pe Ω, iar M_0 este o funcţie surjectivă de la A_0 la R_0. Proprietatea remarcabilă a infrastructurii N este "Nodurile etichetate cu *u* şi *v* sunt legate prin arcul etichetat cu *w* dacă şi numai dacă $(u, v) \in M_0(w)$)";
- E este o supramulţime a lui A_0;
- R este o supramulţime a lui R_0 inclusă în $<R_0>$ (închiderea tranzitivă a mulțimii R_0), adică cuprinde acele relaţii care se formează, pornind de la relaţii din R_0, numai cu ajutorul operaţiei de compunere a relaţiilor;
- λ: ExE → E este o aplicaţie parţial definită care stabilește eticheta e_3 a relației obținută prin compunerea relațiilor cu etichetele e_1 și e_2, adică $e_3=\lambda(e_1, e_2)$;
- M : E → R este o extensie surjectivă (eng. *map*) a lui R_0 astfel încât $M(\lambda (e_1, e_2)) = M(e_1)°M(e_2)$;
- Σ este o mulţime care reprezintă spaţiul semantic;
- σ: Ω x E x Ω → Σ este o aplicaţie semantică şi σ (u,e,v) reprezintă "semantica" asociată faptului că nodurile u şi v sunt legate prin arcul etichetat cu e.

Definiția 3.4.2. Pentru două noduri etichetate cu u şi v, răspunsul la o interogare relativ la u şi v poate fi formulat numai dacă există un *drum* de la u la v, notat prin $u = u_1, u_2, ..., u_n = v$, cu arcele etichetate prin $e_1, e_2, ..., e_{n-1}$, semantica asociată fiind $\sigma (u, \lambda(...\lambda(e_1, e_2), ..., e_{n-1}), v)$.

Propoziția 3.4.1. Implementarea în PROLOG a unei interogări privind două noduri, face apel la predicatul asociat funcţiei σ.

3.5. Procesarea interogărilor

Interesul pentru grafuri asociate cunoştinţelor a crescut foarte mult, în ultima perioadă [37]. Exemplele includ, fără a se limita doar la acestea: DBpedia [7], YAGO [94], Freebase [9] şi Probase [116]. În continuare vom ilustra mai multe moduri de procesare a interogărilor în grafuri.

3.5.1. Metoda subgrafului vecinătate

Vom considera multigrafuri. Un nod x este etichetat cu $\lambda(x)$. O muchie e dintre două noduri este etichetată cu $\lambda(e)$. Muchiile multiple pot avea aceeași etichetă. Vom diferenţia între exemplul de interogare şi răspuns, acestea fiind descrise prin tupluri, un tuplu fiind un ansamblu de entităţi. Fiind dat un multigraf G şi un tuplu-interogare t (succesiune de *dim(t)* noduri), dorim să identificăm k-tupluri răspuns cu cea mai mare similaritate cu *t* (evident în ordine descrescătoare). Prin rezolvarea acestei probleme putem comunica cu un sistem RPC bazat pe multigrafuri, dându-i un exemplu de ceea ce dorim, iar sistemul RPC să ne furnizeze răspunsurile găsite după ce vizitează baza de cunoştinţe stocată în graful dat.

Graful vecinătate de prag d asociat unui tuplu t, notat prin H_t, cu nodurile $V(H_t)$ şi legăturile $E(H_t)$, fiind un subgraf slab conex al grafului dat ((în care există lanţ între oricare două noduri (indiferent de sensul legăturii)), iar înlănțuirea este realizată prin cel mult d entități (noduri). Graful interogare este asociat intenţiei utilizatorului fiind un subgraf slab conex asociat lui H_t, conţine toate instanţele de tip interogare care pot fi obținute prin maxim d entități (noduri, muchii). El este notat cu Q_t.

Un graf răspuns A asociat unui graf interogare Q este un graf izomorf cu Q (au același număr de noduri puse în corespondență biunivocă care menține și consistența și completitudinea

la nivelul muchiilor. Evident, pentru un graf Q pot exista mai multe grafuri răspuns din mulţimea notată A(Q).

Este util să găsim graful maximal de parametru *m* dat. Parametrul *m* poate avea ca valoarea maximă, *diametrul grafului*. Intuitiv, pentru un graf reprezentabil pe niveluri, acesta poate reprezenta *numărul nivelurilor*.

O metodă pentru rezolvarea acestei probleme este descrisă prin Algoritmul 3.5.1.

Algoritmul 3.5.1

Intrare: Interogarea (tuplul) t , Graful vecinătate H_t, Un întreg r (cel mult cardinalul lui H_t).

Ieşire: Graful maximal MQH_t, cu *m* muchii.

Etape:

1. m = r/(dim(t)+1); $\Omega(MQH_t) = \emptyset$, $E(MQH_t) = \emptyset$; G = ∅;
2. Pentru fiecare v_i din t execută
   2.1. G_{vi} = subgraful vârfurilor conectate prin v_i în t (și cu muchiile incidente) obținut prin parcurgerea în adâncime;
   2.2. Adaugă G_{vi} la G.
3. G_0= graful obţinut prin exploare în adâncime asociat entităţilor interogării. Adaugă G_0 la G.
4. Pentru fiecatre graf g din G execută
   4.1. p = 1; h = 1; s = m;
   4.2. Cât timp s > 0 execută
      4.2.1 M_s = componenta slab conexă a grafului G care conţine muchiile lui G de nivel s, relativ la interogările date;
      4.2.2 Dacă Ms nevid atunci
        4.2.2.1 Dacă Ms are exact m legături atunci continua cu pasul 4.3.
        4.2.2.2 Dacă Ms are mai puţin de m legături atunci [h = s; Daca p = - 1 atunci continuă cu pasul 4.3]
        4.2.2.3 Dacă Ms are mai mult de m legături atunci a) dacă h > 0 atunci {s = h; continuă cu pasul 4.3} b) {f = s; p = -1}.
      4.2.3 s = s+p;
   4.3 Dacă s = 0 atunci s = f; Adaugă $\Omega(M_s)$ la $\Omega(MQG_t)$; Adaugă $E(M_s)$ la $E(MQG_t)$.

Observăm că pentru procesarea informaţiei stocate în grafuri de concepte este necesară *explorarea grafurilor în adâncime*. În plus, utilizarea strategiei *greedy* în determinarea subgrafurilor slab conexe este o primă alegere din punct de vedere algoritmic. Optimizarea algoritmilor reprezintă, de asemenea, o cerinţa importantă.

Propoziția 3.5.1. Dacă n = dim(t), m = r/(n+1), iar identificarea valorii s necesită, în medie, c iterații, atunci complexitatea algoritmului 3.5.1 este O(n+1)(εlog(ε)+c(m+c)).

3.5.2. Metoda dominanță-similaritate

Presupunem că graful de cunoștințe are vârfurile (nodurile) etichetate cu mulțimi de date (etichetă fiecărui nod este o mulțime ale cărei valori se actualizează dinamic). Componentele etichetelor pot fi specificate în cadrul unor interogări.

Determinarea similarității a două mulțimi depinde de tipul elementelor, fiind utilizate metode bazate pe atributele comune sau pe distanța dintre elemente. Fie G un graf, $u \in V(G)$, S(u) o mulțime de caracteristici ale vârfului u. Fie W o funcție de evaluare a caracteristicilor.

Atunci, similaritatea a două mulțimi S(u), S(v), unde u și v sunt două vârfuri ale lui G, se poate calcula folosind metoda lui Jaccard [35, 111]:

$$Sim(S(u), S(v)) = \Sigma\{W(x): x \in S(u) \cap S(v)\} / \Sigma\{W(x): x \in S(u) \cup S(v)\}.$$

În cele ce urmează ne interesează rezolvarea următoarei instanțe: Fiind dat un graf G = (V(G), E(G)) de dimensiune mare, un graf interogare Q = (V(Q), E(Q)), cu n vârfuri v_1, v_2, ..., v_n și un prag μ. Se cere determinarea tuturor subgrafurilor X cu n vârfuri ale lui G astfel încât:

a) X este structural izomorfic cu Q;
b) Nodurile subgrafului X sunt $x_i = f(v_i)$, unde f este o bijecție f: $V(Q) \rightarrow V(X)$;
c) Pentru orice $(v_i, v_j) \in E(Q)$, $(f(v_i), f(v_j)) \in E(X) \subset E(G)$;
d) Sim $(S(v_i), S(x_i)) \geq \mu$, i = 1, 2, ..., n.

Pentru rezolvarea acestei probleme, Hong et al [35] au propus utilizarea conceptului de dominanță.

Definiția 3.5.1. Fie G = (V, E) un graf neorientat, fără bucle. O mulțime de vârfuri ale lui G, notată D(G), se numește mulțime dominantă pentru G dacă oricare vârf al lui V este fie în D(G), fie este direct legat (adiacent) de un vârf din D(G).

Propoziția 3.5.1. Presupunem că u este un vârf din mulțimea dominantă a grafului G. Dacă D(G) are cel puțin două elemente ($|D(G)| \geq 2$), atunci există cel puțin un vârf $v \in D(G)$ astfel încât numărul de treceri de la u la v nu depășește 3.

Definiția 3.5.2. Dacă $u \in D(G)$, atunci orice vârf v cu proprietatea din Propoziția 3.5.1 se numește vârf dominant al lui u.

Definiția 3.5.3. Fie $u \in V(G)$. Notăm $N_1(u) = \{v \in V(G)$, există drum de lungime 1 între u și v$\}$ și $N_2(u) = \{v \in V(G)$, există drum de lungime 2 între u și v$\}$.

Observația 3.5.1. Fie G = (V, E) un graf neorientat, fără bucle. Topologiile posibile pentru o mulțime dominantă D(G) cu două vârfuri u și v sunt: 1) $(u, v) \in E$; 2) $\exists$ $w \in V\text{-}D(G)$ astfel încât $(u, w) \in E$ și $(w, v) \in E$; 3) $\exists$ $x, y \in V\text{-}D(G)$, asfel încât $(x,y) \in E$, $(x, u) \in E$ și $(y, v) \in E$; 4) $\exists$ $x, y \in V\text{-}D(G)$, asfel încât $(x,y) \in E$, $(y, u) \in E$ și $(x, v) \in E$.

Definiția 3.5.4. Fie G = (V, E) un graf interogare neorientat, fără bucle. Fie D(G) o mulțime dominantă a lui G. Graful interogare dominant QD este definit prin (V(QD), E(QD)), cu $(u, v) \in E(QD)$ dacă este verificată cel puțin una din următoarele condiții:

1) $(u, v) \in G$, $u \in D(G)$, $v \in D(G)$;
2) $|N_1(u) \cap N_1(v)| > 0$;
3) $|N_1(u) \cap N_2(v)| > 0$;
4) $|N_2(u) \cap N_1(v)| > 0$.

Algoritmul 3.5.2.

Intrare: Un graf interogare Q, QD - un graf dominant al lui Q, un indice de stare s (starea inițială este s_0), $f(s_0) = \varnothing$ (mulțimea vidă)

Ieșire: Aplicația f dintre graful interogare Q și subgraful similar identificat.

Etape:

1. Dacă $f(s_0) = \varnothing$ atunci
    1.1. Determină corespondența f pentru graful dominant QD
    1.2. f(s) = f(DQ)
2. Dacă f(s) este determinat pentru toate vârfurile lui Q atunci
    2.1. f(Q) = f(s)
    2.2. Finalizare f(Q)

altfel pentru fiecare $x \in V(Q) - V(DQ)$ execută

pentru fiecare $(u, v) \in N_1(x) \times N_2(x)$ execută

pentru $y \in (\cup\{N_1(w), w \in Candidat(u)\}) \cap (\cup\{N_2(w), w \in Candidat(v)\}$ execută:

a) Dacă (x, y) satisfac condiția de similaritate a mulțimilor S(x) și S(y) atunci f(s) = f(s) ∪ {(x, y, (x, y))} și actualizează starea s'.

b) s := s', Reia de la etapa 2.

Observația 3.5.2. Mulțimea candidaților în raport cu vârfurile considerate se determină ținând cont de conservarea distanței. Mai precis, dacă (u, v) este o muchie a grafului DQ, $v \in$ Candidat (u) dacă au loc următoarele condiții: 1) dacă ponderea muchiei este 1 atunci nodurile sunt adiacente; 2) dacă ponderea muchiei este 2 atunci $|N_1(u) \cap N_1(v)|>0$; 3) dacă ponderea muchiei este 3 atunci fie $|N_1(u) \cap N_2(v)|>0$, fie $|N_2(u) \cap N_1(v)|>0$.

Observația 3.5.3. Dacă condiția de similaritate a vârfurilor consideră structura topologică (matricea de adiacență) și toleranța este zero, atunci algoritmul 3.5.2 determină structuri izomorfe cu graful Q.

Observația 3.5.4. Algoritmul 3.5.2 este un algoritm de identificare aproximativă a subgrafurilor cu anumite proprietăți, datorită pragului introdus prin intermediul similarității.

4. METODE SOFT COMPUTING ÎN REPREZENTAREA CUNOȘTINȚELOR

4.1. Introducere

Orice sistem de reprezentare și procesare a cunoștințelor (RPC) care funcționează cu date imprecise trebuie să proceseze fluxuri de intrare provenind din mai multe surse: experți care își descriu cunoștințele despre un anumit domeniu în limbaj natural sub subiectivitatea și ipotezele de ambiguitate, senzori și instrumente care pot măsura eronat, respectiv modelare aproximativă. Nu numai datele incomplete trebuie să fie analizate înainte de a fi stocate în baza de cunoştinţe a sistemului RPC, dar și datele inconsistente, datele neclare, datele ambigue, datele fuzzy sau vagi trebuie să fie considerate şi preprocesate înainte de identificarea şi extragerea cunoştinţelor.

Tratarea impreciziei în timpul dobândirii, reprezentării și procesării cunoștințelor este una dintre cele mai dificile sarcini atunci când proiectăm un sistem RPC. Din punct de vedere numeric, au fost propuse de cercetători, începând cu 1950, următoarele abordări: metoda intervalelor [64], numere fuzzy [15, 122], intervale fuzzy [12, 122], numere intuitionistic - fuzzy (Atanssov [3, 4, 5, 15]) și numere neutrosofice (Smarandache [83, 84, 85]). Potrivit lui Moore [64], pentru a lucra cu un număr incert *w*, este mai bine să lucrăm cu un interval *[x, y]* care conține *w*. De asemenea, dat fiind o funcție *f*, valoarea *f(w)* va aparține unui interval *[u, v]* fiind, de asemenea, imprecisă. Dacă # este un operator generic care reprezintă oricare din operațiile adunare, scădere, înmulțire sau împărțire a numerelor reale și notăm operatorul corespunzător în aritmetică interval prin <#> atunci se va aplica următoarea regulă [64]: [a, b] <#> [c, d] = [min (a#c, a#d, b#c, b#d), max (a#c, a#d, b#c, b#d)], dacă și numai dacă x#y este definit pentru $x \in$ [a, b] și $y \in$ [c, d]. În cazul divizării, este necesară o regulă care să ia în considerare numărul zero: 1 / [c, 0] = [-∞, 1 / c] și 1 / [0, d] = [1 / d, ∞]. Combinarea numerelor cu intervale este ușoară deoarece orice număr μ este același cu intervalul [μ, μ].

Definiția 4.1. Fie X o mulţime nevidă reprezentând universul discursului. O mulţime fuzzy A al lui X este definită de toate elementele aparținând lui X în conformitate cu gradul de apartenență prin funcția f_A definită pe X cu valori în [0,1]. În acest caz, să notăm cu A următorul ansamblu: A = {(x, f_A(x)), x ∈ X}. Acest tip de mulţime a fost introdus de Zadeh [120], care a definit gradul de neapartenenţă prin 1 - f_A(x), pentru orice x ∈ X. Când cineva schimbă funcția de apartenenţă, atunci se va obține o nouă mulţime fuzzy.

Exemplul 4.1. Fie X mulțimea persoanelor care sosesc în unitatea de primiri urgențe (UPU). La sosire fiecare persoană vine cu un indicator al gravității stării sale. Acest indicator este unul calitativ, are la bază doar analize sumare și permite o primă clasificare (un cod): roșu (foarte grav), galben (critic), verde (urgent), albastru (non-urgent), alb (în afara oricărui pericol). Aceste coduri se obțin pe baza unor reguli. De exemplu: Dacă "pacientul manifestă insuficiență respiratorie severă" sau "pacientul necesită resuscitare volemică" sau ... atunci "pacientul se află în stare foarte gravă". Dacă semnele vitale sunt în limitele normale pentru vârsta pacientului, dar necesită intervenție atunci i se atribuie codul verde cu un anumit grad. Dacă la un moment dat sunt mai mulți pacienți cu acelasi cod, dar cu grade diferite de încadrare, atunci au prioritate la primirea asistenței medicale de urgență cei cu gradul de încadrare mai mare. Dacă împărțim intervalul [0, 100] în cinci regiuni: alb = [0, 19], albastru = [20, 39], verde = [40, 59], galben = [60, 79], roșu = [80, 100], atunci fiecare pacient x ∈ X primește un grad de apartenență f(x) ∈ [0, 1]. Spunem că are loc transformarea numită nuanțare (fuzzificare) prin trecere de la (x, 1) la (x, f(x)).

Exemplul 4.2. Imprecizia exprimării unei măsurători *x* se poate face prin intermediul definirii unui grad de precizie *f(x)*. De exemplul, folosind patru valori a_1, a_2, a_3, a_4 putem defini un model trapezoidal astfel [111, 122]:

Dacă x < a_1 sau x > a4 atunci f(x) = 0. Dacă x ∈ [a_2, a_3] atunci f(x) = 1. Dacă x ∈ [a_1, a_2] atunci f(x) = (x-a_1)/(a_2-a_1). Dacă x ∈ [a_3, a_4] atunci f(x) = (x-a_4)/(a_3-a_4).

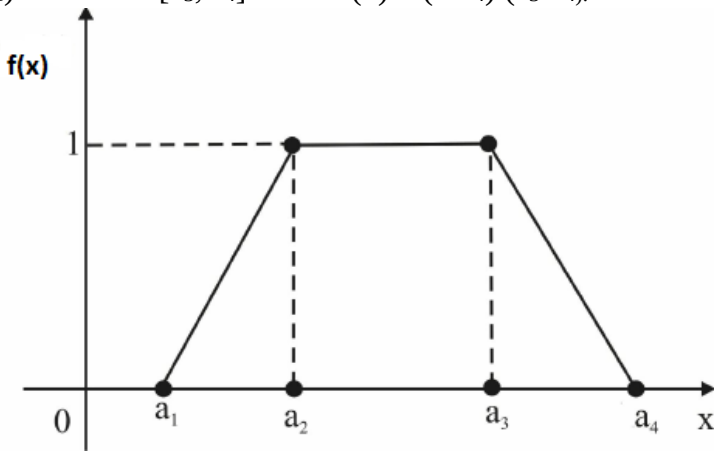


Figura 4.1. Numere fuzzy trapezoidale

Definiția 4.2. Fie X, A și f_A ca mai sus. Dacă, pentru orice x ∈ X, gradul de neapartenenţă este definit prin funcția g_A (g_A: X → [0,1]), cu f_A(x) + g_A(x) fiind mai mic sau egal cu 1), atunci se definesc mulţimile intuitionistic-fuzzy [3, 4, 5]. În acest caz, valoarea 1-f_A(x) - g_A(x) se numește *gradul de nedeterminare* și exprimă lipsa cunoașterii privind apartenența lui x la mulţimea A.

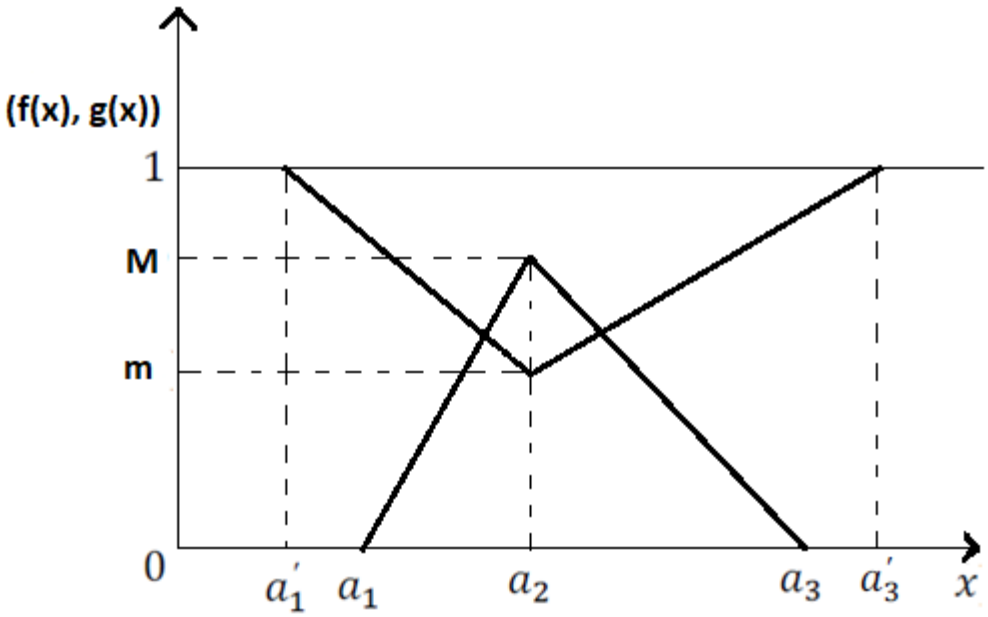


Figura 4.2. Număr intuiționistic-fuzzy de tip triunghiular

Exemplul 4.3. În figura 4.2, cu ajutorul a 5 numere se descrie un număr intuiționistic fuzzy triunghiular, unde funcția de apartenență se formează cu ajutorul numerelor a_1, a_2 și a_3, iar funcția de neapartență cu ajutorul numerelor a'_1, a_2 și a'_3. Se observă că cel mai mare grad de apartenență este M, iar cel mai mic grad de neapartenență este m. În contextul modelării intuiționistic fuzzy $m + M < 1$.

Observația 4.1. Pentru mulţimi fuzzy, nu există nedeterminare.

O nouă extindere, la o reprezentare cu trei valori a fost considerată de Smarandache [83, 84, 85].

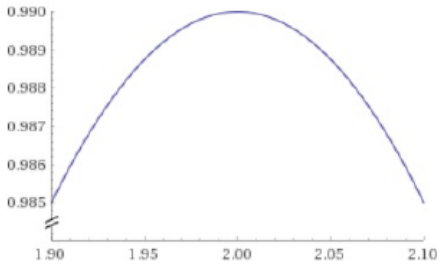
Definiția 4.3. Dacă $T_A(x)$ este gradul de apartenență, $F_A(x)$ este gradul de neapartenență, și $I_A(x)$ este gradul de nedeterminare, atunci *reprezentarea neutrosofică a unei mulţimi A* este definită prin [83, 84, 85]:

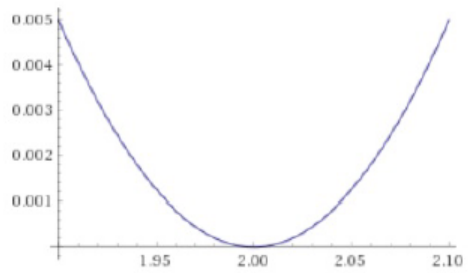
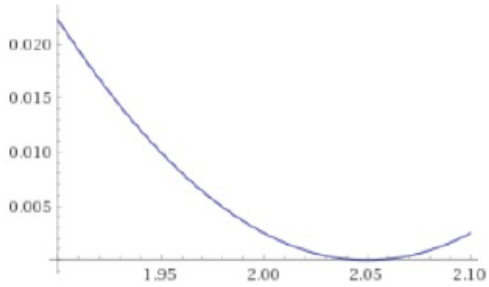
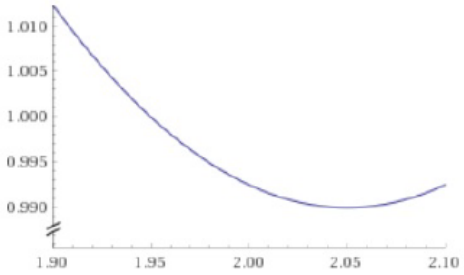
$A = \{(T_A, I_A, F_A), \forall x \in X, 0 \leq T_A(x), F_A(x), I_A(x) \leq 1 \text{ și } 0 \leq T_A(x) + F_A(x) + I_A(x) \leq 3^+\}$,

unde pentru orice număr real ^-a, și a^+ sunt numere speciale definite în analiza nestandard [76]: $^-a = \inf\{a-\varepsilon, \varepsilon \in R^*, \varepsilon \text{ infinitezimal}\}$ și $a^+ = \sup\{a+\varepsilon, \varepsilon \in R^*, \varepsilon \text{ infinitezimal}\}$.

Exemplul 4.4. Fie A mulțimea neutrosofică definită prin tabelul 4.1:

Tabelul 4.1. Exemplu de mulțime neutrosofică

Funcția de apartenență	Reprezentare grafică
$T_A(x) = \exp\left(-\frac{(x-2)^2)}{2}\right) - 0.01$	

$F_A(x) = 1 - \exp\left(-\frac{(x-2)^2)}{2}\right)$	
$I_A(x) = \exp\left(-(x-2.05)^2\right)$	
$T_A + F_A + I_A$	

Se observă că suma gradelor depășește valoarea 1. Aceasta face ca modelul lui Smarandache să fie o extensie a modelelor Zadeh și Atanassov.

Următoarea secțiune consideră sistemele RPC fuzzy, a treia se referă la sistemele RPC intuitionistic-fuzzy, iar a patra secțiune este dedicată sistemelor RPC neutrosofice.

4.2. Modelarea fuzzy și reprezentarea cunoștințelor lingvistice

Un sistem RPC fuzzy are reguli de forma "în cazul în care A este *adevărat* cu un *grad mare*, atunci B este *adevărat* cu un *grad ridicat*", sau "dacă x este *mare*, y este *mediu*, iar z este *scăzut* atunci t este *bine"*, pentru a menționa câteva exemple. Astfel de sisteme RPC pot fi utilizate ca sisteme de control (*dacă A este cald într-un anumit grad, atunci B ar trebui coborât până la o anumită valoare*) sau se utilizează pentru a calcula calitatea generală a unui experiment studiat (dacă A este *mare* și B este *mare* atunci rezultatul este *foarte bun*, dacă C este *mare* și B *nu* este *mare*, atunci rezultatul este *slab*).

Nu există nici o pierdere a generalității dacă presupunem regulile de forma

"Dacă X este A și Y este B atunci Z este C".

Regulile date ca exemplu utilizează expresiile de limbaj natural cum ar fi "mari", "foarte bune", "săraci" etc. Astfel de elemente sunt numite valori ale unor *variabilele lingvistice* [105].

Pentru a oferi răspunsuri la întrebările formulate de orice utilizator final, motorul inferențial al sistemului RPC ar trebui să poată aplica cele mai potrivite proceduri de implicare

fuzzy. După Atanassov și colab. [5], un operator de implicare *IF(a, b)* ar trebui să respecte axiome din următoarea listă:

A1) (∀a) (∀b) (a ≤ b → (∀c) (IF(a, c) ≥ IF(b, c);
A2) (∀a) (∀b) (a ≤ b → (∀c) (IF(c, a) ≥ IF(c, b));
A3) (∀t) (IF(0, t) = 1);
A4) (∀t) (IF(1, t) = t);
A5) (∀t) (IF(t, t) = 1);
A6) (∀a) (∀b) (∀c) (IF(a, IF(b, c)) = IF(b, IF(a, c));
A7) (∀a) (∀b) (IF(a, b) = 1 dacă și numai dacă a ≤ b);
A8) (∀a) (∀b) (IF(a, b) = IF(N(b), N(a))), unde N este operatorul de negare;
A9) IF este o funcție continuă.

Observații 1) Toate axiomele sunt verificate de operatorul de implicare Lukasiewicz [122]:

IF(x, y) = min (1,1 - x + y).

2) Doar primele șapte axiome sunt verificate de implicația lui Godel [122]:

IF (x, y) = 1 dacă x ≤ y și IF (x, y) = y, dacă x> y.

Prin urmare, un sistem RPC fuzzy este compus din: *Modul de intrare* (date crisp), *Unitatea de fuzzificare*, *Baza de cunoștințe fuzzy*, *Motorul de inferență fuzzy*, *Unitate de defuzzificare*, *Modul de ieșire* (date clare).

Deoarece motorul inferențial produce cunoaștere exprimată nuanțat, este necesar ca Modulul M_3 sa apeleze unitatea de defuzificare prin intermediul căreia informatia nuanțată se transformă în informație crisp.

Cele mai uzuale metode de defuzificare sunt bazate pe graficul funcției de apartenență. Se poate reține ca valoare crisp acceptata abscisa care fie aparține centrului de greutate al plăcii determinate de funcția de apartenență (figura 4.3), fie abscisa care împarte aria subgraficului funcției de apartenență în două părți egale (figura 4.4). Se poate alege și valoarea modală, dacă maximul este unic identificat.

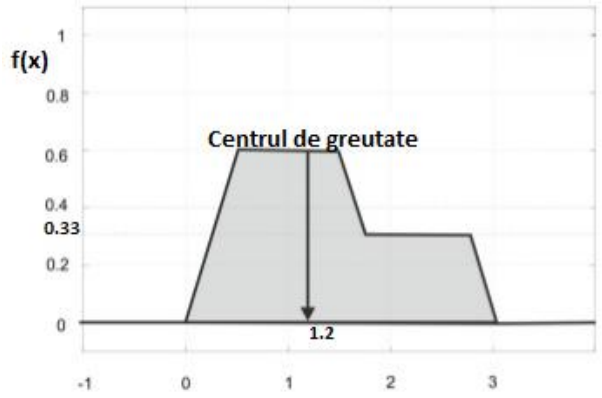


Figura 4.3. Metoda centrului de greutate

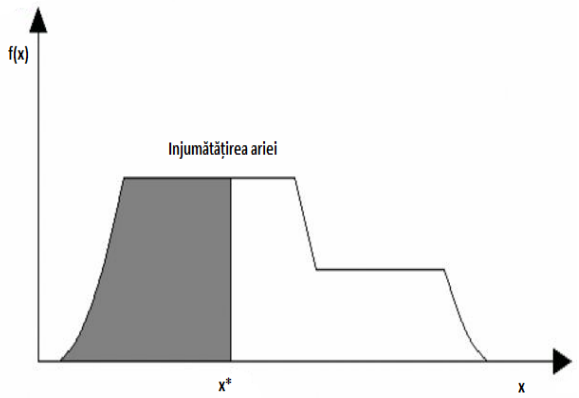


Figura 4.4. Metoda înjumătățirii ariei

Evident, modulele de intrare și ieșire vor fi conectate la interfața cu utilizatorul. Tipul de interfață depinde de decizia analistului de sistem care formulează cerințele software.

4.3. Reprezentarea cunoștințelor intuiționistic-fuzzy

Fie X un set oarecare de obiecte stocate într-un sistem RPC. Într-un cadru intuiționistic-fuzzy A, pentru fiecare x din mulțimea univers va fi definit atât $f_A(x)$ ca și $g_A(x)$, reprezentând

gradul de apartenență și gradul de neapartenență [3], [12]. Este posibil să se introducă și alte caracteristici:

- *gradul de nedeterminare* $h_A(x) = 1 - f_A(x) - g_A(x)$,
- *gradul de favorizare* a lui x prin $m_A(x) = f_A(x)(1 + h_A(x))$, precum și
- *gradul de împotrivire* asupra lui x cum ar fi $n_A(x) = g_A(x)(1 + h_A(x))$.

Din motive teoretice, dar și aplicative, sunt utile următoarele definiții [3, 24, 59]:

Definiția 4.4. Două mulțimi intuitionistic-fuzzy ale lui X, notate prin A și B, sunt *similare* dacă există $x \in X$ astfel încât $f_A(x) = f_B(x)$ și $g_A(x) = g_B(x)$. Dacă aceste proprietăți se aplică pentru orice $x \in X$, atunci A și B sunt *egale* sau *comparabile*.

Definiția 4.5. Două mulțimi intuitionistic-fuzzy A și B sunt *echivalente* dacă și numai dacă există o bijecție h pentru a avea: $f_B(x) = h(f_A(x))$ și: $g_B(x) = h(g_A(x))$, pentru orice $x \in X$. Prin urmare, $f_A(x) = h^{-1}(f_B(x))$ și $g_A(x) = h^{-1}(g_B(x))$.

Conceptele de"subset", respectiv "subsetul propriu" sunt definite prin relația de ordine directă dintre gradele de membru, iar ordinea inversă prin relația dintre gradele care nu sunt membre, și anume:

$$A \subseteq B \textit{ dacă și numai} \text{ dacă } f_A(x) \leq f_B(x)$$

și

$$g_A(x) \geq g_B(x),$$

$$\text{și } A \subset B \textit{ dacă și numai dacă } f_A(x) < f_B(x), \text{ și } g_A(x) > g_B(x),$$

respectiv.

Definiția 4.6. Fie x o propoziție modelată de logica fuzzy intuiționistă. Prin urmare, valoarea ei de adevăr este dată de $V(x) = \langle a, b \rangle$, unde a și b sunt gradele de valabilitate și nonvalabilitate pentru x, folosind notațiile lui Atanassov et al. [5]. Propoziția fuzzy intuiționistă x este o *tautologie fuzzy intuiționistă* dacă și numai dacă $a \geq b$.

Fie x și y două propoziții fuzzy intuiționiste, date prin $V(x) = \langle a, b \rangle$ și $V(y) = \langle c, d \rangle$. Fie IIF(a, b; c, d) operatorul de **i**mplicare **i**ntuitionistic- **f**uzzy. Cu axiomele introduse în secțiunile anterioare, operatorul IIF al lui Zadeh definit prin

$$IIF(a, b, c, d) = \langle \max(b, \min(a, c)), \min(a, d) \rangle$$

satisface axiomele A2/3/4/5/7/9, în timp ce operatorul IIF a lui Kleene-Dienes dat de

$$IIF(a, b; c, d) = \langle \max(b, c), \min(a, d) \rangle$$

satisface toate axiomele cu excepția lui A7.

Regula de compunere inferențială bazată pe logica intuitionistic-fuzzy (RCILIF), conform lui Cornelis & Deschrijver [16], utilizează două universuri de discurs U și V, două variabile X și Y presupuse cu valori din U, respectiv V și o relație intuiționistic-fuzzy R între U și V. RCILIF poate fi exprimată ca o schema de deducție

$$\{X \text{ este } A', \quad (X, Y) \text{ este } R; \quad Y \text{ este } B' = R \circ A'\},$$

unde A' este o multime intuiționistic-fuzzy de univers U.

RCILIF definește *schema intuitionistic-fuzzy a raționamentului modus ponens generalizat*. Este important, atunci când se construiește un sistem RPC bazat pe concepte intuiționistic-fuzzy, să se utilizeze numai operatori de implicare care să asigure validitatea schemei modus ponens.

Atât numerele fuzzy cât și numerele intuiționistic-fuzzy pot fi utilizate în contextul sistemelor RPC care suportă imprecizia. Prin urmare, un sistem RPC Intuitionistic-Fuzzy este compus din: *Modul de intrare* (date clare), *unitatea de fuzzificare intuiționistă*, *baza de cunoștinte modelate intuitionistic-fuzzy*, *motorul de inferență fuzzy intuiționist bazat pe operatori*

adecvati de implicare cu respectarea schemei modus ponens generalizat in context intuitionistic-fuzzy, *unitatea de defuzzificare intuiționistă*, *Modul de ieșire* (date clare).

Cu excepția unității de defuzzificare intuiționistă, celelalte module au o descriere similară corespondentelor din cadrul sistemelor RPC fuzzy.

Metoda funcției indicator. Unitatea de defuzzificare necesită o componentă suplimentară care pornind de la funcțiile f_A și g_A, determină o *funcție indicator* h_A care va fi interpretată ca un grad de apartenență nuanțat ($h_A(x) = \alpha f_A(x)+\beta(1-g_A(x))$ și apoi va avea loc o defuzzificare printr-o metodă, precum cele descrise mai sus. În expresia de mai sus, α și β sunt *parametri de importanță* aleși de analist pentru a obține funcția indicator [63].

4.4. Modele neutrosofice în reprezentarea cunoștințelor

Mulțimile neutrosofice, din punct de vedere filosofic, sunt modelate cu ajutorul valorilor reale sau sunt submulțimi ale intervalului non-standard $]^{-}0, 1^{+}[$.

Deoarece intervalul nestandard $]^{-}0,1^{+}[$ este dificil de utilizat pentru situații reale din inginerie și probleme științifice se folosește intervalul unitar standard [0,1].

În cele ce urmează folosim reprezentarea TIF a mulțimilor neutrosofice, reprezentare introdusă de Smarandache [83, 84, 85].

Definiția 4.7. Un element x din universul discursului X se numește *semnificativ* în ceea ce privește mulțimea neutrosofică A în cazul în care următoarele grade sunt semnificative: *gradul de adevăr/ membru/ apartenență* sau *gradul de falsitate/non-membru* sau *nederminarea*, adică $T_A(x)$ sau $I_A(x)$ sau $F_A(x)$ este mai mare de 0.5 [63].

Definiția 4.8. O multime neutrosofic-intuiționistă este definită prin [2]:

$$\tilde{A} = \langle x, T_A(x), I_A(x), F_A(x)\rangle,$$

unde min $\{T_A(x), F_A(x)\} \leq 0.5$, min $\{T_A(x), I_A(x)\} \leq 0{,}5$ și min $\{F_A(x), I_A(x)\} \leq 0.5$, $\forall x \in X$, atunci când $0 \leq T_A(x) + I_A(x) + F_A(x) \leq 2$.

Operatorii din logica neutrosofică pot fi definiți în mai multe moduri cu privire la nevoile aplicațiilor sau la rezolvarea problemelor [83, 84, 85]. Pot fi utilizate N-normele (N_n) și N-conormele (N_c).

N_n reprezintă operatorul *and* (și) în logica neutrosofică și operatorul de *intersecție* în teoria mulţimilor neutrosofice.

N_c reprezintă operatorul *or* (sau) operatorul în logică neutrosofică și operatorul *reuniune* în teoria multimilor neutrosofice.

Ambele N_n și N_c trebuie să satisfacă următoarele axiome [63]:

a) *Condiţii limită* pentru N_n: $N_n(t, 0) = 0$, $N_n(t, 1) = t$;

b) *Condițiile la limită* pentru N_c: $N_c(t, 1) = 1$, $N_c(t, 0) = t$;

c) *Commutativitate*: $N(u, v) = N(v, u)$, unde $N \in \{N_n, N_c\}$;

d) *Monotonie*: dacă $u \leq v$ atunci $N(u, w) \leq N(v, w)$, unde $N \in \{N_n, N_c\}$;

e) *Asociativitate*: $N(N(u, v), w) = N(u, N(v, w))$, unde $N \in \{N_n, N_c\}$.

Un exemplu general de N-normă este construit ca

$$N_n(u, v) = (T_1 \wedge T_2, I_1 \vee I_2, F_1 \vee F_2),$$

unde $u(T_1, I_1, F_1)$, $v(T_2, I_2, F_2)$, $\wedge$ este o N-normă, și $\vee$ este orice N-conormă.

În mod similar, un exemplu general de o N-conormă este construit ca

$$N_c(u, v) = (T_1 \vee T_2, I_1 \wedge I_2, F_1 \wedge F_2).$$

De exemplu, se pot utiliza următoarele definiții:

$$a \wedge b = ab \text{ (ca produs) } \textit{și } a \vee b = a + b - ab.$$

Un operator negaţie poate fi ușor definit de $n(u) = v$, unde $u(T, I, F)$ și $v(F, I, T)$.

Un operator de implicare neutrosofică (NI) poate fi introdus prin NI (u, v) = N_c (n(u), v). În orice caz, în funcție de constituenții folosiți pentru a defini o N-normă (respectiv N-conormă) se pot obține mai multe modele definitorii pentru operatorul de implicare.

Definiția 4.9. [63] Formal, un *număr neutrosofonic* este un obiect de forma a+ bI, unde a și b sunt numere reale sau complexe, iar I este operația prin care I^2 = I, I-I=0, I+I=2I, 0I=0, dar 1/I și I/I nu sunt definite. Notăm R(I) numerele neutrosofice cu componente reale, respectiv cu C(I) numerele neutrosofice cu componente complexe.

Remarca 4.1. [63] I^n=I (n>0), xI+yI=(x+y)I.

Definiția 4.10. [63] Dacă x = a+bI și y = c+dI sunt numere neutrosofice de tip R(I) or C(I) atunci:

a) x + y = (a+c) + (b+d)I;

b) x - y = (a-c) + (b-d)I;

c) xy = (ac) + (ad + bc + bd)I;

d) αx = (αa) + (αb)I;

e) x/y = u+vI (nd este posibil), unde u = a/c, iar v = (bc-ad)/(c^2+cd).

Remarca 4.2. $(a+bI)^2 = a^2 + (2ab+b^2)I$; $(a+bI)^n = a^n+I\{\Sigma Comb(n,k)a^{n-k}b^k$; k= 1, 2, ..., n}, unde Comb(n,k) = n!/k!/(n-k)!, iar n! = 1 x 2 x ... x n (factorialul numărului n).

Remarca 4.3. [63] Dacă a ≥ 0 și a+b ≥ 0 atunci $\sqrt{}$(a+bI) = u+vI ∈ {s_1, s_2, s_3, s_4}, unde s_1 = ($\sqrt{}$a, -$\sqrt{}$a + $\sqrt{}$(a+b)), s_2 = ($\sqrt{}$a, -$\sqrt{}$a - $\sqrt{}$(a+b)), s_3 = (-$\sqrt{}$a, -$\sqrt{}$a + $\sqrt{}$(a+b)) și s_4 = (-$\sqrt{}$a, -$\sqrt{}$a - $\sqrt{}$(a+b)).

Definiția 4.11. [63] Dacă x = a+bI și y = c+dI sunt numere neutrosofice de tip R(I) atunci definim o relație de ordine parțială pe mulțimea numerelor neutrosofice astfel:

a) a + bI ≡ c + dI dacă și numai dacă a = c și b = d;

b) Dacă a ≤ c atunci a + bI ≤ c+dI;

c) Dacă a = c și b ≤ d atunci a + bI ≤ c+dI.

Un alt model util al numerelor neutrosofice este cel inspirat din notația TIF și va fi specificat prin trei componente numerice care reprezintă gradul de apartenență, gradul de nedeterminare și gradul de neapartenență. Cele trei componente au valori în intervalul [0, 1]. Deoarece vom lucra cu valori și nu cu intervale sau submulțimi de numere pentru fiecare din cele trei componente, aceste numere (t, i, f) vor fi numite numere neutrosofice simple.

Definiția 4.12. [63] Fie A = (t_1, i_1, f_1) și B = (t_2, i_2, f_2) numere neutrosofice simple. Fie ∧ operatorul asociat intersecției (respectiv conjuncției logice) și ∨ operatorul asociat reuniunii (respectiv disjuncției logice). Atunci, operațiile aritmetice de bază, când pot fi definite, au următorul mod de lucru:

a) adunarea: A + B = ($t_1 \vee t_2$, $i_1 \wedge i_2$, $f_1 \wedge f_2$) (de exemplu = (t_1+t_2-t_1t_2, i_1i_2, f_1f_2));

b) înmulțirea: AB = ($t_1 \wedge t_2$, $i_1 \vee i_2$, $f_1 \vee f_2$) (de exemplu = (t_1t_2, i_1+i_2-i_1i_2, f_1+f_2-f_1f_2));

c) scăderea: A - B = (t_3, i_3, f_3) unde t_3 = (t_1 - t_2)/(1 - t_2), i_3 = i_1/i_2 și f_3 = f_1/f_2.

d) împărțirea: A / B = (t_3, i_3, f_3) unde t_3 = t_1/t_2, i_3 = (i_1 - i_2) / (1 - i_2) și f_3 = (f_1-f_2)/(1-f_2).

e) ridicarea la putere: C^λ = (1-(1-t)$^\lambda$, i^λ, f^λ), dacă C = (t, i, f) este un număr neutrosofic simplu.

Un altfel de model util, numit număr neutrosofic quadruplu este o entitate de forma *(a, bT, cI, dF)*, este scris ca *a+bT+cI+dF*, unde *T, I, F* au semnificația obișnuită, iar a, b, c, d sunt numere reale. Componenta *a* reprezintă partea determinată (cunoscută sigur), iar *bT+cI+dF* este partea necunoscută. Conform lui Smarandache (2015), într-un model pesimist (*formal: T<I<F*), putem presupune că *TI = IT = I, TF = FT = F, IF = FI = F, TT = T^2 = T, II = I^2 = I, FF = F^2 = F, 0T = 0I = 0F = 0.*

Următoarele proprietăți sunt necesare pentru a permite procesarea cunoștințelor neutrosofice reprezentate cu ajutorul numerelor quadruple [18]:

- Dacă $x = a_1 + b_1T + c_1I + d_1F$ și $y = a_2 + b_2T + c_2I + d_2F$, atunci $x + y = (a_1+a_2) + (b_1+b_2)T + (c_1+c_2)I + (d_1+d_2)F$, $x - y = (a_1-a_2) + (b_1-b_2)T + (c_1-c_2)I + (d_1-d_2)F$, $xy = a_1a_2 + (a_1b_2 + a_2b_1 + b_1b_2)T + (a_1c_2 + a_2c_1 + c_1c_2 + c_1b_2 + c_2b_1)I + .(a_1d_2 + a_2d_1 + b_1d_2 + b_2d_1 + c_1d_2 + c_2d_1 + d_1d_2)F$.
- Dacă $x = a_1 + b_1T + c_1I + d_1F$ și m este un număr real (scalar) atunci $mx = ma_1 + mb_1T + mc_1I + md_1F$.
- Dacă $x = a_1 + b_1T + c_1I + d_1F$, $y = a_2 + b_2T + c_2I + d_2F$, iar m și n sunt numere reale (scalari), atunci $m(x + y) = mx + my$, $(m + n)x = mx + nx$, $(mn)x = m(nx)$, $-x = -a_1 - b_1T - c_1I - d_1F$.
- Într-un model optimist (*formal:* $T>I>F$) modelul numerelor neutrosofice quadruple se bazează pe proprietățile: $TI = IT = T$, $TF = FT = T$ și $IF = FI = I$. Dacă $x = a_1 + b_1T + c_1I + d_1F$ și $y = a_2 + b_2T + c_2I + d_2F$ atunci $xy = a_1a_2 + (a_1b_2 + a_2b_1 + b_1b_2+ c_1b_2 + c_2b_1+ d_1b_2 + d_2b_1)T + (a_1c_2 + a_2c_1 + c_1c_2+ c_1d_2 + c_2d_1)I + .(a_1d_2 + a_2d_1 + d_1d_2)F$. Se poate alege modelarea optimistă sau cea pesimistă în funcție de situația reală de modelat.

O extensie mai interesantă este reprezentată de grafurile neutrosofice, cu nedeterminare la nivel de vertex (nod, vârf), sau la nivel de muchie (sau arc). Componenta supusă nedeterminării (nod sau conexiune) este utilizată atunci când există informații incomplete despre astfel de entități. Matricea de adiacenţă a unui graf neutrosopic [59, 63, 108] constă din elemente aparținând setului {0 - *nici o conexiune*, 1 - *conexiune sigură*, I - *conexiune nedeterminată între noduri*}, precum și numere neutrosofice.

Hărțile cognitive neutrosofice și hărțile relaționale neutrosofice sunt generalizări ale hărților cognitive fuzzy și respectiv hărților relaționale fuzzy. O hartă cognitivă neutrosofică reprezintă relația cauzală dintre concepte, adiacenţă neutrosofică. Matricea corespunzătoare relației neutrosofice a unei hărți cognitive neutrosofică are valori aparținând mulțimii {0, 1, -1, I}, unde 0, 1 și I sunt cele de mai sus, iar -1 descrie o conexiune proporțională inversă [108].

Cele de mai sus ne permit să afirmăm că sistemele RPC de tip neutrosofic pot fi implementate în mai multe moduri, decizia fiind a analistului problemei de rezolvat.

Asemănător sistemelor RPC descrise mai sus, un sistem RPC neutrosophic este compus din: *modul de intrare* (cu date clare), *unitate de neutrosoficare*, *baza de cunoștințe neutrosofice*, *Motor inferential de tip neutrosofic*, *unitate de deneutrosoficare*, *modul de ieșire* (date clare).

Unitatea de neutrosoficare preprocesează datele de intrare clare pentru a identifica cazuri valabile, cazuri invalide, cazuri ambigue.

Odată identificați operatorii neutrosofici pentru conectorii logici utilizați la implementarea RPC, baza cu reguli neutrosofice poate fi utilizată precum în cadrul clasic al sistemelor expert implementate prin programare logică sau reguli de producție. Unitatea de deneutrosoficare este răspunzătoare cu informațiile privind calitatea de membru / valabilitate pentru a furniza un centru de greutate sau o valoare particulară a datelor (medie, mediană etc.).

În situația în care rezultatul este de forma $A = \{(T_A(x), I_A(x), F_A(x)) \mid x \in X\}$, procesul de deneutroficare are doi pași. Mai întâi se determină o funcție indicator (**Metoda funcției indicator**) cu ajutorul a trei parametri α, β și γ, astfel:

$$h_A(x) = \alpha T_A(x) + \beta I_A(x) + \gamma(1-F_A(x)),\ x \in X.$$

În al doilea pas se aplică o metodă de defuzificare pentru a determina efectiv rezultatul.

Un sistem medical KRP implementează algoritmi medicali (bazați pe "arborii de decizie") care sunt folositori pentru un diagnostic parțial bazat pe cele mai potrivite teste de laborator. În continuare, să folosim numerele neutrosofice într-un format a+bI asociate valorilor normale ale elementelor CBC: Hemoglobină (g / dl) = 13,5 + 3I (masculin) și 12 + 3I (feminin), Hematocrit (%) = 41 + 9I (masculin) și 36 + 8I (feminin), RBC (x106 / ml) = 4,5 + I (masculin) și 4 + 0,9I (feminin). Când se iau în considerare electroliții, valorile comune pot fi convertite în calciu (mg / dl) = 8,8 + 1,5I, clorură (mEq / L) = 95 + 12I, magneziu (mEq / L) = 1.6 + 0.8I, fosfat / dL) = 2,5 + 2 I, potasiu (mEq / L) = 3,5 + 1,7 I, sodiu (mEq / L) = 135 + 12I.

În cazul sistemelor expert medicale, care lucrează cu numere neutrosofice, valori robuste ale indicatorilor se pot obține cu ajutorul medianei e = a+b, sau distanței intercuartile m = a+3b/4+c/2, dacă se folosește o reprezentare quadruplă (a, b, c, d).

5. MODELE ŞI TEHNOLOGII *MARKUP* UTILIZATE ÎN REPREZENTAREA ŞI PROCESAREA CUNOAŞTERII

5.1. Introducere

Prin modele şi tehnologii Markup, în contextul acestei lucrări, înţelegem acele modele şi tehnologii derivate din SGML (Standard Generalized Markup Language [14]; ISO 8879:1986). Metalimbajul XML a fost proiectat pentru a se obține o versiune simplificată a limbajului SGML, dar pentru a permite transferul structurilor de date între aplicațiile software bazate pe comunicare folosind protocolul Internet.

O aplicaţie în limbajul RPC care utilizează tehnologiile XML trebuie să identifice şi să traducă obiectele descrise în fişierul XML. Traducerea se face pe baza schemelor şi a DTD-urilor, care sunt definite pentru acel document. Se pot utiliza diverse tehnologii pentru a analiza un document XML.

În contextul reprezentării cunoaşterii cu ajutorul regulilor putem considera **<Cunoaștere>** ca element rădăcină care poate include unul sau mai multe elemente de tip **Regula**. Fiecare element **Regula** are un identificator unic *rid*. O regulă de asociere (A => B) este formată din **Ipoteza A** și **Consecința B**. Fiecare **Ipoteză** și **Consecință** trebuie să aibă unul sau mai mulţi itemi ipoteză și elementele consecinţă. Fiecare item **Ipoteză** și **Consecinţă** are un nume reprezentat printr-un șir. Redăm acest model în Tabelul A.

Tabelul A. Exemplu de model XML pentru reguli de asociere [56, 61, 62]

Document XML	DTD-ul modelului
<Cunoastere> <Regula rid="1"> <Ipoteza> <ItemIpoteza> <Name>A</Name> </ItemIpoteza> </Ipoteza> <Consecinta> <ItemConsecinta> <Name>B</Name> </ItemConsecinta> </Consecinta> </Regula> </Cunoastere>	<!ELEMENT Cunoastere (Regula+)> <!ELEMENT Regula (Ipoteza Consecinta)> <!ATTLIST Regula rid CDATA> <!ELEMENT Ipoteza (ItemIpoteza+)> <!ELEMENT ItemIpoteza (Nume)> <!ELEMENT Consecinta (ItemConsecinta+)> <!ELEMENT ItemConsecinta (Nume)> <!ELEMENT Nume (#PCDATA)>

Pentru a descrie structuri utile procesării cunoaşterii din domeniul ştiinţei solului pentru agricultură, Meenakshi et. al. [52] au convertit declaraţiile XML într-un format util aplicaţiei, numit KBML (Knowledge Based Markup Language). Toate meta-informațiile sunt stocate într-un fișier KBML, în timp ce datele reale pot fi disponibile în orice sursă de date (distribuite, mobile etc.). Totuşi KBML nu este un limbaj Markup, ci doar o aplicaţie XML.

5.2. Extensii RDF

În contextul modelării şi procesării cunoaşterii au fost dezvoltate notaţii Markup specializate precum: RDF/XML (model sintactic pentru exprimarea serializată a grafurilor RDF ca documente XML), CKML (Conceptual Knowledge Markup Language, [42]), OML (Ontology Markup Language [134]), DLML (Description Logic Markup Language [129]). OML este o extensie a lui SHOE şi suportă expresiile lambda. OML şi CKML sunt bazate pe grafurile conceptuale introduse de Sowa [90, 91, 92].

RDF (*The Resource Description Framework*) este cadrul de lucru pentru definirea resurselor și furnizează un model şi o sintaxă pentru metadate pentru procesare şi transfer inter aplicații [11]. Mai precis, RDF este un util pentru procesarea metadatelor și asigură interoperabilitate între diverse aplicaţii din mediul Internet. RDF foloseşte limbajul XML [141]. Specificarea datelor RDF necesită un proiect CRC sau UML pentru a descrie asocierile dintre clase. Setul de clase este numit schema RDF. Extensibilitatea se obține prin mecanismul de derivare/specializare a claselor identificate în timpul analizei problemei subsecvente implementării sistemului RPC.

Primele versiuni CKML au avut la bază filosofia prelucrării cunoaşterii conceptuale (*CKP – Conceptual Knowledge Processing*). Noile versiuni CKML au inclus idei şi tehnici privind fluxul informaţional (*IF – Information Flow*) şi proiectarea logică a sistemelor distribuite. Versiunea finală CKML este atât un limbaj bazat pe logică descriptivă cât şi un limbaj bazat pe cadre. Conform Kent [42], "în CKML conceptualizarea de specificat necesită utilizarea noțiunii matematice de *latice conceptuală* sau cea mai practică noțiune de *spațiu conceptual*". Deducem, de aici, clarificarea noţiunii de latice conceptuală prin intermediul "conceptelor formale" ca elemente ale unei *structuri laticiale* a conceptelor. Baza teoretică a practicii bazate pe CKML o constituie teorema CKP care afirmă *echivalenţa dintre structurile de date de tip latice conceptuală şi context formal (clasificare)* [42]. Pentru nuanţare se utilizează scale conceptuale de tip nominal sau ordinal.

OML asigură trei niveluri de specificare a restricţiilor [134]: **superior** – secvenţe (corespunzătoare fluxului informaţional); **intermediar** – calculul relaţiilor binare; **inferior** – expresii logice (corespunzătoare grafurilor conceptuale). Înainte de discutarea acestor scheme de marcare, vom dezbate câteva idei privind ontologiile, resursele digitale etc.

Exprimarea unei ontologii este posibilă cu ajutorul limbajelor de specificare precum [11, 22, 70, 118]: KIF (Knowledge Interchange Format), CL (Common Logic), OIL, DAML+OIL [128] şi OWL [11, 134].

KIF se bazează pe logica predicatelor, dar oferă o sintaxă orientată Lisp pentru aceasta [129]. Din punct de vedere semantic, există patru categorii de constante în KIF: constante de tip obiect, constante de tip funcție, constante de tip relație și constante logice.

OIL (*Ontology Inference Layer*) extinde schema RDF pentru a oferi o sintaxă intuitivă şi o mare putere de expresie şi o semantică mai bine definită cu facilitarea utilizării logicii

descriptive în cadrul schemelor de raţionament. Astfel OIL reuneşte şi unifică trei direcţii: logica descriptivă, modelarea bazată pe cadre şi modelarea RDF/XML [11].

DAML (*DARPA Agent Markup Language*) + OIL are o sintaxă de tip schema RDF, moşteneşte primitivele ontologice ale RDF (sublase, domeniu, interval) şi adaugă primitive suplimentare precum tranzitivitate, cardinalitate etc. Schema DAML+OIL este orientată pe obiecte în care conceptele sunt abstractizate prin clase, iar rolurile prin proprietăţi ale obiectelor. Astfel, modelul ontologic DAML+OIL are la bază o mulţime de axiome despre clase şi proprietăţi, precum şi un set de constructori foarte utili din perspectiva sistemelor RPC [11, 128].

Din cele de mai sus rezultă următorul proces evolutiv: 1) OIL extinde RDF; 2) DAML extinde RDF; 3) DAML+OIL integrează DAML şi OIL şi extinde RDF; 4) OWL extinde DAML+OIL şi RDF.

Rezultatul final al cercetărilor privind modelarea ontologică folosind RDF/XML a condus la specificarea OWL, în trei variante: OWL lite (ierarhie simplă ierarhizare de clase cu constrângeri simple), OWL DL (expresivitate maximă) şi OWL Full (foarte expresivă). Ca aplicabilitate practică OWL DL se remarcă prin garantarea proceselor de raţionament. OWL permite obţinerea unor clase complexe prin utilizarea operaţiilor: *intersecţie* (owl:intersectionOf), *reuniune* (owl:unionOf), *complementară* (owl:complementOf), *enumerare* (owl:oneOf) şi *disjuncţia claselor* (owl:disjointWith). De asemenea, OWL pune la dispoziție restricţii asupra proprietăţilor. Aceste restricţii se dovedesc utile în schemele de raţionament specifice sistemelor RPC: allValuesFrom, hasValue, someValuesFrom, cardinality, minCardinality, maxCardinality. Alte caracteristici OWL se referă la nume şi date. Numele OWL sunt referințe RDF URI, iar datele aparţin tipului de dată definit de schema XML..

5.3. SCXML

Modelul stărilor unei maşini pentru abstractizarea controlului poate fi descris, în XML, folosind un limbaj proiectat pentru îndeplinirea unui astfel de obiectiv: SCXML (*State Chart XML*) [147, 148]. Conceptele fundamentale SCXML sunt [147]: stare, tranziție, eveniment. O stare este definită de o mulțime de tranziții posibile la apariția unor evenimente generate fie din interior, fie din exterior.

Un scurt exemplu SCXML descrie un automat finit determinist cu trei stări: q1 - stare inițială, q2 - stare intermediară, q3 - stare finală, evenimentele e1 și e2, care realizează tranziții conform Tabelului 5.3.

Tabelul 5.3. Funcția de tranziție

T(q,e)	q1	q2	q3
e1	q1	q3	q2
e2	q2	q1	q3

Următorul text SCXML descrie modul de acțiune a automatului descris în tabel:

```
<scxml initial state="q1">
<state id= "q1">
<transition event= "e1" target= "q1"/>
<transition event= "e2" target= "q2"/>
</state>
<state id= "q2">
<transition event= "e1" target= "q3"/>
<transition event= "e2" target= "q1"/>
</state>
```

```
<state id= "q3">
<transition event= "e1" target= "q2"/>
<transition event= "e2" target= "q2"/>
</state>
</scxml>
```

Dacă e1 descrie situația în care intrarea este cifra binară 0, iar e2 cazul specific cifrei binare 1, atunci automatul recunoaște numerele întregi din mulțimea {x | x = 3k+2, k = 0, 1, 2,...}.

5.4. Voice XML

VoiceXML este un limbaj de marcare pentru specificarea dialogului vocal dintre un om și o aplicaţie informatică, de exemplu un sistem RPC. Astfel, folosind VoiceXML 2.0 se pot realiza aplicaţii RPC care asigură recunoaşterea automată a vorbirii (*ASR - Automated Speech Recognition*) şi răspunsul interactiv vocal (*IVR - Interactive Voice Response*).

Aplicaţiile vxml pot fi de tip uni - sau multi - document. O aplicaţie multi - document permite definirea unui document rădăcină care defineşte toate entităţile vizibile şi valorificate de documentele fiu. Exemplul de mai jos descrie un schelet de tip interogare folosind voce. Acest model poate fi extins pentru implementarea interfeţelor bazate pe voce în cadrul sistemelor RPC.

VXML 3.0 [149] aduce elemente noi din perspectiva realizării interfețelor prin voce în cadrul sistemelor RPC atât prin extinderile aduse versiunii 2.0, dar și prin noile facilități introduse. În plus, specificația VXML 3.0 se bazează pe modelul SCXML.

Orice aplicație RPC cu interfață VoiceXML are în spate (backend) atât baza de cunoștințe cât și codul pentru modulele sistemului RPC. Partea de interfață poate fi organizată pe niveluri: 1) nivelul PREZENTARE; 2) nivelul CONTROL și 3) nivelul UIDM - User Interface Data Model. Astfel, se respectă tehnica MVC - Model_View_Controller de implementare a interfețelor cu utilizatorul.

Pentru recunoaștere se utilizează inferența gramaticală bazată pe SRGS [150]. În final, descriem codul VXML pentru traducerea datelor dintr-un tabel [62]:

```
<?xml version="1.0"?>
<vxml version="2.0">
..
<block> Urmatoarea structura este matricea 1 </block>
<block> Numele matricei este </block>
// Cod ...
<block> Linia 1 </block>
<block> Coloana 1 </block>
<block> E[1][1] </block>
<block> Coloana 2 </block>
<block> E[1][2] </block>
<block> Finalul liniei 1 </block>
<block> Linia 2 </block>
<block> Coloana 1 </block>
<block> E[2][1] </block>
<block> Coloana 2 </block>
<block> E[2][2] </block>
<block> Finalul liniei 2 </block>
<block> Acesta este finalul matricei 1 </block>
</vxml>
```

5.5. Tehnologii Java pentru procesarea documentelor Markup

Pentru implementarea sistemelor RPC în Java [131], soluţia cea mai potrivită o constituie utilizarea JAXB (*Java Architecture for XML Binding*).

5.5.1. Arhitectura JAXB

Din punct de vedere al procesului de legare XML-JAVA implementat de JAXB, se remarcă existenţa a două componente majore: un generator de scheme şi un compilator de scheme, iar procesul efectiv de legare implică şapte acţiuni [57, 62, 131, 141]:

1. **Generarea claselor**: Compilatorul de scheme JAXB preia, ca intrare, o schema xml si generează clasele Java asociate descrierii.
2. **Compilarea claselor**: Toate clasele generate, alte surse Java şi codul Java completat vor fi compilate.
3. ***Unmarshal***: Documentele XML care îndeplinesc restricţiile din schema sursă sunt procesate de către JAXB. De asemenea, JAXB permite transferarea datelor XML din alte surse decât fișiere și documente XML, cum ar fi nodurile DOM, tablouri de șiruri de caractere, surse SAX și așa mai departe.
4. **Generarea arborelui** care descrie conţinutul documentului XML: Procesul *unmarshal* generează un arbore al obiectelor instanțiate din clasele JAXB generate; Acest arbore reprezintă structura și conținutul documentelor sursă XML.
5. **Validarea**: Procesul *unmarshal* presupune validarea sursei înainte de generarea arborelui care descrie conținutul. Dacă se modifică arborele în pasul următor, se poate utiliza operația de validare JAXB pentru a valida modificările înainte de a transforma conținutul într-un document XML.
6. Aplicația client poate **modifica** datele XML reprezentate de arborele JAXB utilizând interfețele generate de compilatorul JAXB.
7. ***Marshal*:** Arborele care descrie conţinutul este convertit în document XML. Conţinutul poate fi validat înainte de conversie. Procesul numit "Marshalling" oferă unei aplicații client capacitatea de a converti un arbore Java derivat din JAXB în date XML.

Regulile care stau la baza proceselor descrise mai sus sunt (figura 5.1) :

R1: Documentul XML, notat cu D, respectă Schema XML, notată cu S
R2: Legare(S) => Mulţimea claselor JAXB = M
R3: Obiectele sunt instanţe ale claselor JAXB din M.
R4: Marshal (Obiecte) => Document XML
R5: Unmarshal (Document XML) => Obiecte

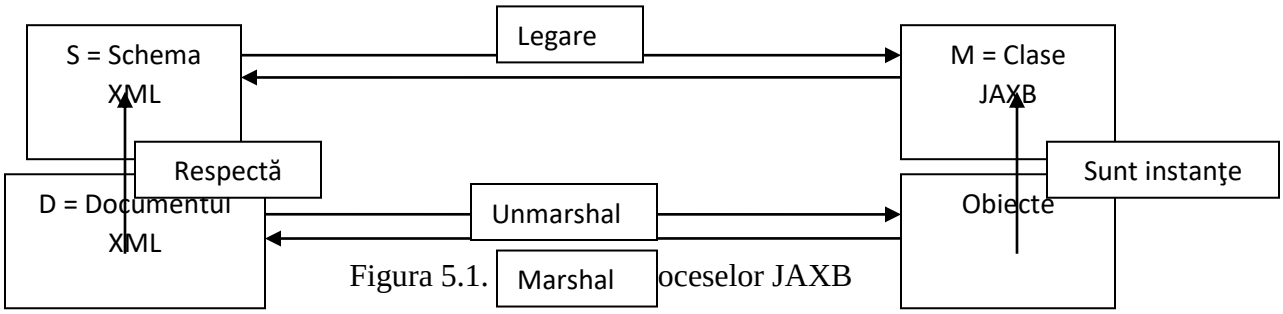


Figura 5.1. oceselor JAXB

5.5.2. Procesarea bazelor de cunoştinţe mari folosind JAXP

O tehnologie alternativă, dar pentru a cărei aplicare este necesar un efort mai mare este JAXP (Java API for XML Procesing) [131, 141], care utilizează DOM (Document Object Model) şi se bazează pe SAX (Simple API for XML Parsing).

În timpul operatiei de "parsing" bazată pe SAX se generează evenimente care anunţă elementele identificate, iar aplicaţia Java trebuie să trateze evenimentele cu metode *callback* (pentru construirea structurii de date). Operatia de parsing DOM construieşte în memorie o reprezentare arborescentă a datelor din documentul XML. Tehnologia JAXP permite transformarea documentelor XML folosind tehnologia XSLT (*Extensible Stylesheet Language Transformation*).

SAX poate fi folosit în următoarele situaţii: 1) procesarea unor documente XML de mari dimensiuni; 2) aplicaţia necesită abandonarea procesării (procesorul SAX poate fi oprit oricând); 3) se doreşte extragerea unor informații de dimensiune mică; 4) în contextul unor capacităţi de calcul reduse (memorie limitată, lărgime de bandă îngustă, etc.)

5.5.3. Tehnologia XMLBeans

Această tehnologie de compilare a schemei XML cu obţinerea, în memorie, a arborelui claselor, a fost dezvoltată în perioada 2003-2014 de Apache Software Foundation pentru a permite procesarea unor scheme de mari dimensiuni. XMLBeans pentru Java are drept corespondent C++ mediul de procesate XML denumit xmlbeansxx [131, 141].

Din cele de mai sus rezultă că pentru procesarea bazelor de cunoştinţe structurate care respectă o schemă şi sunt stocate în fişiere XML, se pot utiliza tehnologiile: 1) JAXB - orientată pe schema XML şi 2) JAXP - orientată pe procesarea directă a documentelor XML. JAXP este o bună alegere pentru baze de cunoştinţe mari ce urmează a fi procesate în condiţii de capacitate computaţională scăzută.

6. TEHNOLOGII JAVA PENTRU INTEROGAREA BAZELOR DE CUNOŞTINŢE

6.1 Introducere

În acest capitol descriem o implementare a procesului de interogare pentru o bază de cunoştinţe extinse moştenite. Implementarea utilizează tehnologia client-server (Figura 6.1) oferită de platforma Java. Sunt descrise atât partea de server cât şi partea de client a aplicaţiei. Aplicaţia permite utilizarea unui server şi a mai multor clienţi şi a fost testată într-o reţea wireless privată. Sunt prezentate etapele de prelucrare şi rezultatele date de către server.

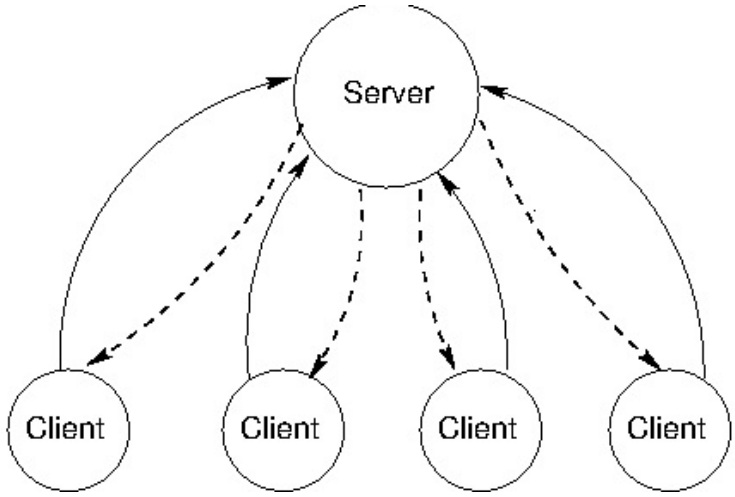


Figura 6.1: Conexiunea client-server

6.2 Descrierea bazei de cunoştinţe

Modelul propus are următoarele caracteristici: 1) Modelul permite folosirea moştenirii multiple pentru calculul valorii unui atribut. Mai exact, valoarea unui atribut poate fi moştenită, de la mai multe obiecte. În acest caz, o funcţie de decizie este necesară pentru a stabili strategia de alegere a unui atribut din mulţimea atributelor moştenite. Această funcţie poate specifica o nouă valoare pentru un atribut care este obţinută din combinarea valorilor selectate. Astfel, valoare unui atribut poate fi o valoare moştenită, dar, la fel de bine, poate fi o valoare nouă. 2) Valoarea unui atribut poate specifica anumiţi parametri care pot reprezenta cunoştinţe nesigure (fuzzy sau probabilistice) sau un factor de risc. 3) Un obiect poate specifica mai multe valori pentru un atribut. În acest caz, un obiect poate moşteni anumite atribute sau poate obţine noi valori pornind de la cele moştenite. Un obiect este caracterizat prin: Numele obiectului; identifică în mod unic un obiect; O mulţime finită de nume simbolice. Fiecare din acestea reprezintă un alt obiect care este numit părintele unui obiect. Un anume obiect poate avea zero părinţi sau mai mulţi părinţi; O mulţime finită de sloturi. Un slot este o pereche ordonată de forma *(atrib; val)*, unde *atrib* este un nume simbolic şi *val* este valoarea corespunzătoare acestuia.

Obiectele sunt virtual conectate printr-o relaţie părinte-copil. Fiecare caracteristică a unui obiect este dată printr-un atribut. Un obiect poate specifica valori imediate (de exemplu temperatura, lungimea, înălţimea, culoarea, greutatea, etc.) sau poate identifca numele unuei metode care calculează valoarea unui atribut.

Presupunem că fiecărei valori a unui atribut îi este atribuit un parametru. Acest parametru stabileşte nesiguranţa valorii, reprezentând un coeficient de risc sau o valoare de cost. Valoarea V a unui atribut A al unui obiect F este calculată după cum urmează:

1. Presupunem că descrierea lui F conţine cel puţin un slot de forma (A; V; p) sau (A; P; p), unde V este o valoare imediată, P este un nume de procedură şi p este un parametru asociat lui (A; V). Notăm cu Slot(F) mulţimea acestor sloturi.
2. Presupunem că avem o strategie de decizie dată printr-o funcţie *Alegere* astfel încât *Alegere*(Slot(F)) = (A; T; p) este slotul selectat. Dacă T = V atunci valoarea lui A este V

şi p este un parametru asociat lui (A; V). Dacă T = P atunci valoarea V este returnată de procedura P pentru anumite valori ale argumentelor sale şi p este parametru asociat acestei valori.

3. Presupunem că descrierea lui F nu conţine sloturi a căror primă componentă să fie A. În acest caz valoarea atributului este obţinută pornind de la părinţii lui F. Cu alte cuvinte, valoarea atributului este moştenită de la părinţi.

Pentru exemplificarea conceptelor este prezentat un obiect ca un sistem de trei entităţi:

(n1; {q1; : : : ; qs}; {(a1; v1; p1); : : : ; (ak; vk; pk)})

unde

- n1 este numele obiectului;
- {q1; : : : ; qs} defineşte mulţimea tuturor părinţilor lui n1;
- {(a1; v1; p1); : : : ; (ak; vk; pk)} reprezintă mulţimea de atribute pentru valorile lor şi valorile parametrilor corespunzători.

Se consideră următoarele obiecte:

(s1; {s2; s3}; {(a; 3; 2); (b; 1; 7); (c; 8; 14)})
(s2;{s4}; {(d; 5; 15); (e; 8; 21)})
(s3; {s5}; {(f; 40; 10); (g; 6; 20)})
(s4; {}; {(h; 20; 10); (x; 50; 30)})
(s5; {s6}; {(h; 30; 20); (y; 4; 3)})
(s6; {}; {(x; 35; 40); (a; 4; 1)})

Mulţimea {s1; s2; s3; s4; s5; s6} reprezintă o bază de cunoştinţe extinsă bazată pe moştenire. Relaţia părinte-copil permite reprezentarea acestei baze de cunoştinţe ca în figura 6.2. Folosim strategia valorii maximale pentru determinarea valorii parametrilor şi notăm cu V (s; atr) valoarea unui atribut atr pentru obiectul s şi obţinem următoarele valori:

- V (s1; h) = 30 deoarece mulţimea celor mai aproape predecesori ai lui s1 care conţin atributul h este mulţimea {s4; s5} şi parametrul maximal este 20;
- V (s1; x) = 50 deoarece s4 este cel mai apropiat predecesor al lui s1 care conţine atributul x.
-

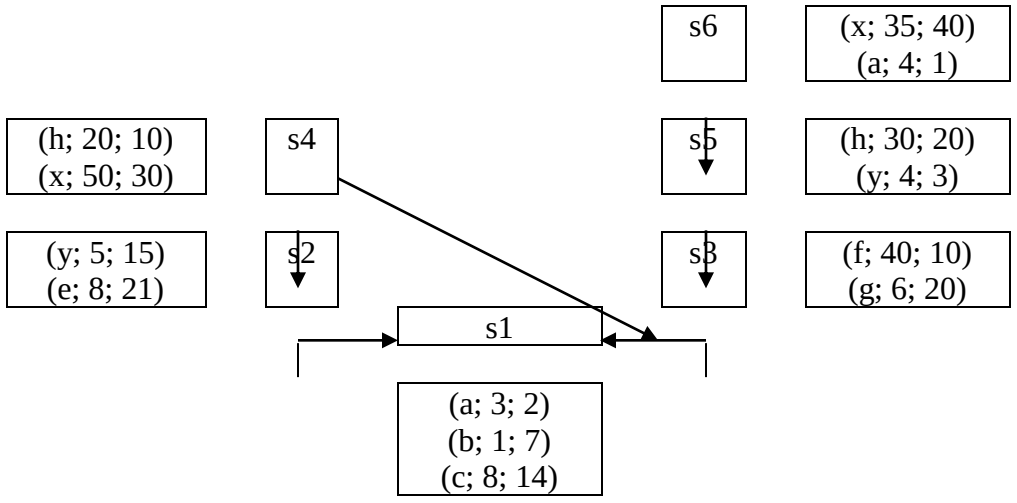


Figura 6.2: Un exemplu de bază de cunoştinţe

6.3 Implementarea interogării bazelor de cunoştinţe

O bază de cunoştinţe bazată pe moştenire este un şir finit L de entităţi de forma (n1; {q1; : : : ; qs}; {(a1; v1; p1); : : : ; (ak; vk; pk)}) aşa cum a fost specificat în secțiunea anterioară. O interogare este definită ca o pereche (ob; ar), unde ob este un nume de obiect din L şi ar este un nume de atribut.

În continuare considerăm că valoarea *pi* este factorul de certitudine al valorii *vi* a atributului *ai*. Răspunsul la interogarea (ob; atr) este o propoziţie de forma "*Valoarea atributului atr pentru obiectul ob este vs cu factorul de certitudine pr*" sau "*Valoarea atributului atr pentru obiectul ob este necunoscută*".

Notăm cu Raspuns(ob; atr) o astfel de propoziţie răspuns. Pentru exemplul prezentat anterior se obţin următoarele rezultate:

- *Raspuns(s1; h) = Valoarea atributului h pentru obiectul s1 este 30 cu factorul de certitudine 20.*
- *Raspuns(s3; x)=Valoarea atributului x pentru obiectul s3 este 50 cu factorul de certitudine 30.*
- *Raspuns(s2; a)=Valoarea atributului a pentru obiectul s2 este necunoscută.*

Din punct de vedere al implementării, într-un program Java, baza de cunoştinţe considerată este structurată ca o clasă Java, fără membri de tip metodă:

```
private Object[][] data3 = {
{"s1", "s2 s3","atr(a,3,2) atr(b,1,7) atr(c,8,14)"},
{"s2", "s4","atr(y,5,15) atr(e,8,21)"},
{"s3","s4 s5","atr(f,40,10) atr(g,6,20)"},
{"s4","","atr(h,20,10) atr(x,50,30)"},
{"s5","s6","atr(h,30,20) atr(y,4,3)"},
{"s6","","atr(x,35,40) atr(a,4,1)"}};
```

Fiecare linie defineşte un obiect. De exemplu, prima linie reprezintă obiectul s1 care conţine:

- părinţii s2 şi s3;
- atributele a, b şi c şi valorile corespunzătoare acestor atribute 3, 1 şi 8; al treilea argument defineşte parametrul valorii atributului.

Pentru simplificarea procesării şi pentru o mai bună vizualizare a etapelor de procesare, aplicaţia client conţine o interfaţa utilizator care include:

1. modul de dialog specific mesajele de la server; un mesaj de bun venit de la server este afișat după realizarea primei conexiuni a serverului cu clientul; rezultatele interogării sunt afisate de acest modul;
2. un modul specific informaţiilor auxiliare privind etapele procesării: numele bazei de cunoştinţe aleasă de către utilizator; numele obiectului şi numele atributului selectat de către utilizator; confirmarea schimbărilor efectuate în baza de cunoştinţe;
3. modul de confirmare privind:
    - numele bazei de cunoştinţe selectate de către utilizator;
    - numele obiectului selectat;
    - numele atributului selectat.
4. modul pentru declanşarea acţiunilor:
    - specificarea unei noi interogări asupra aceleiaşi baze de cunoştinţe;

- specificarea intenţiei utilizatorului de schimbare a bazei de cunoştinţe;
- schimbarea bazei de cunoştinţe;
- terminarea etapelor de procesare ale clientului.

6.4 Rezultatele aplicaţiei Java

Serverul este inclus în fişierul KBServer:java şi clientul în fişierul KBClient:java. Pentru exemplificarea aplicaţiei folosim un server şi doi clienţi, client1 şi client2, conetaţi într-o reţea wireless. Se execută următorii paşi:

1. Se lansează serverul:
   java KBServer
2. Pentru clienţii, client1 şi client2, se execută comanda:
   java KBClient

7. CONCLUZII ŞI DEZVOLTĂRI VIITOARE

7.1. Concluzii

Această lucrare a investigat stadiul cercetărilor în domeniului sistemelor RPC şi a identificat noi modalităţi arhitecturale pentru a procesa baze de cunoştinţe modelate prin tehnici soft computing. Urmărind structura lucrării pot fi formulate următoarele concluzii:

1. Reprezentarea și prelucrarea cunoașterii (RPC) joacă un rol important în realizarea sistemelor expert și se bazează pe următoarele aspecte:
   - Cunoaşterea este compusă din piese elementare, numite concepte.
   - Conceptele se află în relaţii / asociaţii unele cu altele.
   - Conceptele şi asociaţiile sunt părţi ale unor structuri a căror dinamică se stabilizează în timp.
   - Cunoaşterea poate fi factuală (prin descrierea atributelor conceptelor), respectiv procedurală (prin descrierea modului de evoluţie a structurilor).
   - Unui sistem pentru reprezentarea și procesarea cunoașterii (RPC) trebuie să-i fie evaluate următoarele proprietăţi: *reprezentabilitatea* – abilitatea de a permite reprezentarea tuturor formelor de cunoaştere specifice domeniului şi necesare rezolvării problemei; *deductibilitatea* – abilitatea de a permite evoluţia structurilor (obţinerea de noi structuri pornind de la cele existente); *eficacitatea* – abilitatea de a permite alegerea celui mai potrivit mecanism evolutiv/deductiv pentru a obţine soluţia cu cel mai mic efort; *autoînvăţarea* – abilitatea de a produce cunoaştere nouă prin metode automate ori de câte ori este posibil.
   - Un sistem RPC poate fi de tip D, I, K sau W. Încadrarea în una din categorii se face prin evaluarea sistemului RPC și aplicarea regulilor de încadrare cu pragul de 90%.
2. Arhitectura unui sistem RPC este constituită din mai multe module care permit utilizatorului interogarea (folosind interfețe utilizator prietenoase) bazei de cunoştinţe în legătură cu anumite întrebări specificate în limbaj natural, sau într-un limbaj artificial (program – script, într-un limbaj de programare).
   - Modelul unui sistem RPC este S = (Q, $\{M_i\}i=1, 2, ...5$, $\{P_i\}i=1, 2, ..., 5$, $\{I_i\}i=1, 2, ..., 5$, $\{C_{ij}\}i=1, 2, ..., 5$; j=1, 2, ..., 5), unde Q este o mulțime de variabile de

stare ale sistemului, M_i, i=1, 2, ...5 sunt modulele prezentate mai sus, P_i este o mulțime de metode de procesare specifice componentei M_i, I_i este o mulțime de interfețe/protocoale care permit interconectarea modulului M_i în structura sistemului S, iar C_{ij} sunt mulțimi de componente care asigură buna funcționare a conexiunii dintre modulele M_i și M_j.(Figura 2.1). Un sistem expert E este o instanță a unui model S de sistem RPC.

- Cunoaşterea poate fi identificată prin tehnici specifice ştiinţelor sociale, dar şi prin tehnici specifice inteligenţei computaţionale.
- Achiziţia cunoaşterii în contextul calităţii impactului produs de sistemul RPC presupune existența unei metodologii cu cel puţin patru faze: (1) planificare, (2) identificarea cunoaşterii, (3) analiza cunoaşterii identificate, (4) verificarea ipotezelor.
- Custodele bazei de cunoştinţe asigură disponibilitatea cunoaşterii, fiabilitatea sistemului RPC, precum şi securitatea bazei de cunoştinţe. Analistul bazei de cunoştinţe menţine piesele cunoaşterii la nivel calitativ din punct de vedere al consistenţei şi integrităţii (ca urmare a aplicării unor transformări sau operaţii de actualizare);
- "Purificarea" cunoaşterii constă în activități de detectare și corectare a pieselor dintr-o bază de cunoştinţe incorecte, incomplete, formatate (modelate) în mod necorespunzător sau redundante. În plus, se va asigura consistenţa cu piesele stocate în alte baze de cunoştinţe. Această sarcină devine dificilă în contextul bazelor de cunoştinţe distribuite şi a noilor abordări bazate pe concepte Big Data. "Purificarea" cunoaşterii se realizează folosind algoritmi de "reparare" ce au la bază reguli specifice dependente de domeniul problemei de rezolvat, dificultatea implementării fiind dată de dependenţa de context a utilizării pieselor cunoaşterii stocate în baza de cunoştinţe. Validarea cunoaşterii este diferită de "purificarea" cunoaşterii şi se referă doar la corectitudinea înregistrării pieselor cunoaşterii în sistemul RPC.
- Există o mare varietate de metode şi unelte software pentru "purificarea" şi validarea datelor, dar acestea pot fi utilizate doar parţial în contextul bazelor de cunoştinţe şi in mod special doar pentru descoperirea cunoştinţelor. Tehnicile de filtrare şi vizualizare joacă un rol important în preprocesarea datelor, iar prin extensie, a cunoaşterii.
- Integrarea cunoaşterii devine dificilă atunci când piesele cunoaşterii provin din mai multe surse. În acest caz trebuie eliminate redundanţele, trebuie identificate eventualele contradicţii precum şi acoperirea tuturor cazurilor (când sunt definite reguli pe porţiuni).
- Regăsirea cunoaşterii are la bază potrivirea parţială, cea mai bună potrivire, respectiv grade de potrivire (în modele fuzzy, intuitionistic-fuzzy, neutrosofice).
- Din punct de vedere al procesului inferenţial se utilizează mecanisme complexe care au la bază inferența deductivă, inferenţa inductivă, raţionamente asociative, respectiv raţionamente prin analogie. Pentru creşterea eficienţei motorului inferenţial, sistemele RPC pot utiliza o agendă de reguli.
- Interfaţa cu utilizatorul poate fi de tip text, interfaţă grafică, dar şi interfeţe audio-video care sunt în dezvoltare şi se bazează pe tehnici specifice de sinteză şi recunoaştere audio-video.

- Principalele modele conceptuale şi tehnologiile existente pentru proiectarea unui sistem RPC sunt ilustrate cu exemple provenind din studii de caz asupra Cyc/CycL, CLIPS, Jess şi Prolog.

3. Cunoaşterea se poate înregistra în momentul în care au fost identificate conceptele, relaţiile dintre acestea şi există soluţia tehnică de implementare computerizată. Reprezentarea cunoaşterii cu ajutorul relaţiilor poate fi privită ca transferul expertizei umane către sisteme de procesare a bazelor de cunoştinţe. Exprimarea relaţiilor dintre concepte se realizează folosind harta conceptelor.
    - Harta conceptuală este un graf orientat, în care nodurile reprezintă conceptele (unităţile semantice importante), iar arcele sunt legături de determinare între acestea (de la general spre particular) care exprimă relaţia dintre două concepte sau noduri. Elaborarea hărților conceptuale poate fi realizată folosind aplicaţia software CMAP.
    - Există două moduri de a privi legătura dintre sistemele pentru reprezentarea şi procesarea cunoaşterii (sisteme RPC) şi Sistemele de Gestiune a Bazelor de Date (SGBD): 1) aplicarea noţiunilor de baze de date şi de procesare a datelor stocate în baze de date la sistemele RPC; 2) aplicarea ideilor specifice RPC (de exemplu logica descriptivă) pentru a proiecta baze de date tradiţionale folosind diagrame entitate-relaţie bine formate.
    - În vederea proiectării şi implementării unui sistem RPC orientat-obiect trebuie parcurse următoarele patru faze: caracterizarea domeniului, modelarea orientată-obiect, formalizarea şi implementarea propriu-zisă.
    - Transformarea unei baze de date orientate obiect într-o bază de cunoştinţe procesate în manieră obiectuală presupune încapsularea mai multor predicate pentru a forma un obiect abstract (un prototip).
    - Grafurile conceptuale pot fi descrise în trei moduri: GDF (Graphical Display Form), CGIF (Conceptual Graph Interchange Format) sau CLIF (Common Logic Interchange Format) şi LF (Linear Form), oricare dintre acestea putând fi translatate în KIF (Knowledge Interchange Format).Reţelele semantice evidenţiază aspecte lexicale, structurale, procedurale şi semantice. Componenta lexicală adresează simbolurile permise în formarea numelor de noduri, etichete etc. Componenta structurală se referă la eventualele restricţii de interconecare a nodurilor. Componenta procedurală adresează operaţii de construire, acces în citire, acces pentru scriere, precum şi proceduri de eliminare. Evident, componenta semantică permite deducerea interpretării drumului dintre două noduri ale reţelei.
    - Pentru procesarea informaţiei stocate în grafuri de concepte este necesară explorarea grafurilor în adâncime. În plus, utilizarea strategiei greedy în determinarea subgrafurilor slab conexe este o primă alegere din punct de vedere algoritmic. Optimizarea algoritmilor reprezintă o cerinţă importantă.
    - Reţelele semantice evidenţiază aspecte lexicale, structurale, procedurale şi semantice. Componenta lexicală adresează simbolurile permise în formarea numelor de noduri, etichete etc. Componenta structurală se referă la eventualele restricţii de interconecare a nodurilor. Componenta procedurală adresează operaţii de construire, acces în citire, acces pentru scriere, precum şi proceduri de eliminare.

- Procesarea interogărilor în grafuri se poate realiza folosind metoda subgrafului vecinătate (Algoritmul 3.5.1), respectiv metoda dominanță-similaritate (Algoritmul 3.5.2). Pentru procesarea informaţiei stocate în grafuri de concepte este necesară explorarea grafurilor în adâncime. În plus, utilizarea strategiei greedy în determinarea subgrafurilor slab conexe este o primă alegere din punct de vedere algoritmic. Metoda dominanță-similaritate este un algoritm de identificare aproximativă a subgrafurilor cu anumite proprietăți, datorită pragului introdus prin intermediul similarității.

4. Când sunt colectate seturi de date incomplete, ambigue sau imprecise atunci se pot utiliza modelele fuzzy ale lui Zadeh, propunerile intuitioniste ale lui Atanassov și extensiile neutrosofice ale lui Smarandache. Aceste modele pot fi aplicate la numere, grafuri, diferite hărți și sisteme RPC. Se obţin noi clase de sisteme RPC cu aplicabilitate în domenii care permit raţionamentul nuanțat.
    - Tratarea impreciziei în timpul dobândirii, reprezentării și procesării cunoștințelor este una dintre cele mai dificile sarcini atunci când proiectăm un sistem RPC. Se pot utiliza metode bazate pe intervale, numere fuzzy, intervale fuzzy, numere intuitionistic - fuzzy și numere neutrosofice.
    - O mulțime fuzzy A este un ansamblu: $A = \{(x, f_A(x)), x \in X\}$. Conceptul a fost introdus de Zadeh, care a definit gradul de neapartenenţă prin $1 - f_A(x)$, pentru orice $x \in X$.
    - O mulțime intuitionistic fuzzy A este un ansamblu: $A = \{(x, f_A(x), g_A(x)), x \in X\}$. Conceptul a fost introdus de Atanassov, care a definit gradul de neapartenenţă prin $g_A(x)$, iar $1 - f_A(x) - g_A(x)$ este gradul de nederminare, pentru orice $x \in X$.
    - O mulțime neutrosofică A este un ansamblu: $A = \{(x, T_A(x), I_A(x), F_A(x)), x \in X\}$. Conceptul a fost introdus de Smarandache, care extins conceptele fuzzy și intuitionistic-fuzzy prin includerea gradului de apartenență, gradului de neapartenență, respectiv a gradului de nedeterminare. Pentru modelele Smarandache se acceptă ca suma celor trei grade să fie între 0 și 3.
    - Un sistem RPC fuzzy este compus din: Modul de intrare (date crisp), Unitatea de fuzzificare, Baza de cunoștințe fuzzy, Motorul de inferență fuzzy, Unitate de defuzzificare, Modul de ieșire (date clare). Modulul de intrare este responsabil cu acele funcții ale componentei M1 a sistemului RPC care asigură achiziția cunoașterii. Acest modul operează cu date de intrare de tip crisp. Unitatea de fuzzyficare realizează, pe de o parte, activități generale de preprocesare a cunoașterii, dar, în plus, transformă reprezentările datelor pentru ca anumite componente RPC să poată opera cu entități nuanțate (mulțimi fuzzy, propoziții fuzzy, numere fuzzy). Baza de cunoștințe fuzzy este o componentă a modulului M2 al sistemului RPC și asigură stocarea și accesul la piesele cunoașterii modelate în manieră nuanțată. Motorul de inferență implementează mecanismele de raționament fuzzy bazat pe operatorii acceptați pentru operațiile și (and), sau (or), negație (not) și implicație. Acesta corespunde modulului M4. Modulul M3 al sistemului RPC preia cererile de la utilizator și cu ajutorul bazei de cunoștințe și a motorului inferențial va furniza rezultatul. Deoarece motorul inferențial produce cunoaștere exprimată nuanțat, este necesar ca Modulul M3 sa apeleze unitatea de defuzificare prin intermediul căreia informatia nuanțată se transformă în informație crisp. Cele mai uzuale metode de defuzificare sunt bazate pe graficul

funcției de apartenență. Se poate reține ca valoare crisp acceptata abscisa care fie aparține centrului de greutate al plăcii determinate de funcția de apartenență, fie abscisa care împarte aria subgraficului funcției de apartenență în două părți egale. Se poate alege și valoarea modală, dacă maximul este unic identificat.

- Un sistem RPC Intuitionistic-Fuzzy este compus din: Modul de intrare (date clare), unitatea de fuzzificare intuiționistă, baza de cunoștinte modelate intuitionistic-fuzzy, motorul de inferență fuzzy intuiționist bazat pe operatori adecvati de implicare cu respectarea schemei modus ponens generalizat in context intuitionistic-fuzzy, unitatea de defuzzificare intuiționistă, Modul de ieșire (date clare). Unitatea de defuzzificare necesită o componentă suplimentară care pornind de la funcțiile f_A și g_A, determină o *funcție indicator* h_A care va fi interpretată ca un grad de apartenență nuanțat ($h_A(x) = \alpha f_A(x)+\beta(1-g_A(x))$ și apoi va avea loc o defuzzificare printr-o metodă, precum cele descrise mai sus. În expresia de mai sus, α și β sunt *parametri de importanță* aleși de analist pentru a obține funcția indicator.
- Hărțile cognitive neutrosofice și hărțile relaționale neutrosofice sunt generalizări ale hărților cognitive fuzzy și respectiv hărților relaționale fuzzy. O hartă cognitivă neutrosofică reprezintă relația cauzală dintre concepte, adiacenţă neutrosofică. Matricea corespunzătoare relației neutrosofice a unei hărți cognitive neutrosofică are valori aparținând mulțimii {0, 1, -1, I}, unde 0, 1 și I sunt cele de mai sus, iar -1 descrie o conexiune proporțională inversă.
- Un sistem RPC neutrosophic este compus din: modul de intrare (cu date clare), unitate de neutrosoficare, baza de cunoștințe neutrosofice, Motor inferential de tip neutrosofic, unitate de deneutrosoficare, modul de ieșire (date clare). procesul de deneutroficare are doi pași. Mai întâi se determină o funcție indicator cu ajutorul a trei parametri α, β și γ, astfel: $h_A(x) = \alpha T_A(x) + \beta I_A(x) + \gamma(1-F_A(x))$, $x \in X$. În al doilea pas se aplică o metodă de defuzificare pentru a determina efectiv rezultatul.
- Principalele rezultate ale acestui capitol sunt: Tratarea unitară a sistemelor RPC indiferent de natura modelului pentru gestiunea situațiilor dominate de imprecizie; Definirea unei metode de defuzzificare intuiționistă; Definirea unei metode de deneutrosoficare; Utilizarea conceptelor staticii descriptive în neutrosoficarea șirurilor de date (rezultate din măsurători); Definirea operațiilor cu entități neutrosofice pentru a permite implementarea sistemelor RPC neutrosofice; Exemplificarea conceptelor cu situații din practică, inclusiv medicină.

5. Cea mai potrivită alegere Markup o reprezintă modelul XML, iar din punct de vedere al tehnologiilor JAVA pentru procesarea documentelor XML merită a fi reţinute pentru aplicaţii practice JAXB (obiectual, procesare în memorie) şi JAXP (liniar, procesare orientată pe fragmente cu identificarea şi tratarea evenimentelor). Efortul de programare JAXB este mai redus şi este promovată procesarea obiectuală. In plus, prin VoiceXML se pot descrie interfete inteligente ale sistemelor RPC bazate pe voce. Concluziile subsecvente urmează:
    - O aplicaţie în limbajul RPC care utilizează tehnologiile XML trebuie să identifice şi să traducă obiectele descrise în fişierul XML. Traducerea se face pe baza schemelor şi a DTD-urilor, care sunt definite pentru acel document. Se pot utiliza diverse tehnologii pentru a analiza un document XML.

- În contextul modelării şi procesării cunoaşterii au fost dezvoltate notaţii Markup specializate precum: RDF/XML (model sintactic pentru exprimarea serializată a grafurilor RDF ca documente XML), CKML (Conceptual Knowledge Markup Language), OML (Ontology Markup Language), DLML (Description Logic Markup Language). OML este o extensie a lui SHOE şi suportă expresiile lambda. OML şi CKML sunt bazate pe grafurile conceptuale introduse de Sowa, părintele hărților conceptuale.
- Modelul stărilor unei maşini pentru abstractizarea controlului poate fi descris, în XML, folosind un limbaj proiectat pentru îndeplinirea unui astfel de obiectiv: SCXML (State Chart XML). Conceptele fundamentale SCXML sunt: stare, tranziție, eveniment. Exemplu din lucrare care descrie un automat finit determinist cu trei stări: q1 - stare inițială, q2 - stare intermediară, q3 - stare finală, evenimentele e1 și e2, și realizează tranziții programate este ilustrativ pentru a reda puterea mecanismului.
- VoiceXML este un limbaj de marcare pentru specificarea dialogului vocal dintre un om și un sistem RPC. Aplicaţiile vxml pot fi de tip uni - sau multi - document. O aplicaţie multi - document permite definirea unui document rădăcină care defineşte toate entităţile vizibile şi valorificate de documentele fiu.
- Orice aplicație RPC cu interfață VoiceXML are în spate (backend) atât baza de cunoștințe cât și codul pentru modulele sistemului RPC. Partea de interfață poate fi organizată pe niveluri: 1) nivelul PREZENTARE; 2) nivelul CONTROL și 3) nivelul UIDM - User Interface Data Model. Astfel, se respectă tehnica MVC - Model_View_Controller de implementare a interfețelor cu utilizatorul.
- Tehnologiile Java JAXP și JAXB sunt eficace și potrivite pentru implementarea unor module RPC.

6. Au fost luate în considerare conceptele de bază de cunoştinţe extinsă bazată pe moştenire şi a fost implementat procesul de interogare a unei astfel de structuri. Implementarea are la bază tehnologia client-server oferită de platforma Java EE8. Cererile de interogare au fost testate într-o reţea wireless privată, folosind o aplicaţie server şi două instanţe client. Serverul conţine mai multe baze de cunoştinţe şi fiecare client poate interoga o bază de cunoştinţe distinctă. Aceeaşi tehnologie poate fi folosită pentru a interoga şi alte tipuri de baze de cunoştinţe, cu o implementare corespunzătoare a funcţiei răspuns.
7. Dezvoltarea sistemelor RPC face apel atât la modele de reprezentare a cunoașterii, scheme de raționament, dar și la tehnologii informatice care să asigure implementarea dialogului om-mașină, gestiunea faptelor și regulilor, efectuarea raționamentului, dar și scheme de pre/post procesare în contextul cunoașterii fuzzy, intuitiuonistic -fuzzy și neutrosofic.

7.2. Dezvoltări viitoare

Cercetările întreprinse în cadrul programului doctoral şi descrise în această teză pot fi continuate, respectiv aplicate în următoarele direcţii:

1. Dezvoltarea de sisteme RPC prototip pentru domenii ale cunoaşterii în care se acceptă raţionamente nuanţate: fuzzy, intuiţionistic fuzzy, respectiv neutrosofice.
2. Interfețe prin Voce pentru interogarea bazelor de cunoştinte.

3. Dezvoltarea unor tehnici specifice procesării unor baze de cunoştinţe de mare dimensiune (context BigData).
4. Extinderea aplicaţiilor RPC pentru a fi performante în context distribuit.

BIBLIOGRAFIE

1. Ackoff R. L., *From Data to Wisdom*, Journal of Applies Systems Analysis, 16, pp. 3-9, 1989.
2. Akram M., Shahzadi S., *Representation of Graphs using Intuitionistic Neutrosophic Soft Sets*, Journal of Mathematical Analysis, 7(6), pp. 31-53, 2016.
3. Atanassov K. T., *Review and New Results on Intuitionistic Fuzzy Sets*, Mathematical Foundations of Artificial Intelligence Seminar, Sofia, 1988, Preprint IM-MFAIS- 1-88. Reprinted: Int. J. Bioautomation, 20(S1), S7-S16, 2016.
4. Atanassov K. T., *On intuitionistic fuzzy graphs and intuitionistic fuzzy relations*, Proceedings of the IV IFSA World Congress, Sao Paulo, Brazil July 1995, 1, pp. 551-554, 1995.
5. Atanassov K. T., Papadopoulos B. K., Syropoulos A., *Variants of Intuitionistic Fuzzy Implications, Issues in Intuitionistic Fuzzy Sets and Generalized Nets*, Vol. 4, K. Atanassov, J. Kacprzyk, M. Krawczak, and E.Szmidt, Eds., Wydawnictwo WSISiZ, Warszawa, pp. 5-8, 2007.
6. Aubert J.F., Baget J.-F., Chein M., *Simple Conceptual Graphs and Simple Concept Graphs*, In Scharfe H., Hitzler P., Ohrstrom P. (editori): ICCS 2006, LNAI 4068, pp. 87-101, 2006.
7. Auer S., Bizer C., Kobilarov G., Lehmann J., R. Cyganiak R., Ives Z., ***DBpedia***: *A nucleus for a Web of open data*. In ISWC'07/ASWC'07, Proceedings of the 6th international The semantic web and 2nd Asian conference on Asian semantic web conference, Busan, Korea, pp. 722-735, 2007.
8. Biswas P., Pramanik S., Giri B. C., *Aggregation of triangular fuzzy neutrosophic set information and its application to multi-attribute decision making*, Neutrosophic Sets and Systems, 12, pp. 20-40, 2016.
9. Bollacker K., Evans C., Paritosh P., Sturge T., Taylor J., ***Freebase***: *a collaboratively created graph database for structuring human knowledge*. In Proceedings of the 2008 ACM SIGMOD international conference on Management of data, Vancouver, Canada, pp. 1247–1250, 2008.
10. Borgida A., *Knowledge Representation meets Databases – a view of the symbiosis*, Proceedings of the 20th International Workshop on Description Logics DL'07, Bolzano University Press, 2007, http://dl.kr.org/dl2007/dl-workshops/dl2007/invited_3.pdf.
11. Buraga S., *Semantic Web*. *Fundamente şi aplicaţii*, Matrix Rom, Bucureşti, 2004.
12. Bustince H., Barrenechea E., Pagola M., Fernandez J., Xu Z., Bedregal B., Montero J., Hagras H., Herera F., de Baets B., *A historical account of types of fuzzy sets and their relationships*, IEEE Transactions on Fuzzy Systems, DOI 10.1109/tFUZZ.2015.2451692, 2015.
13. Chein M., Mugnier M.-L., *Graph-based Knowledge Representation: Computational Foundations of Conceptual Graphs*, Springer Science & Business Media, Springer-Verlag, 2008.

14. Coleman J., Willis D., *SGML as a Framework for Digital Preservation and Access*, Commission on Preservation and Access, Washington, DC, 1997.
15. Cornelis Ch., Deschrijver G., Kerre E. E., *Implication in intuitionistic fuzzy and interval-valued fuzzy set theory: construction, classification, application*, International Journal of Approximate Reasoning, 35, pp. 5595, 2004.
16. Cornelis Ch., Deschrijver G., *The Compositional Rule of Inference in an Intuitionistic Fuzzy Logic Setting*, Proceedings of the ESSLLI2001 Student Session (13th European Summer School in Logic, Language and Information), (Ed. K. Striegnitz), pp. 83-94, 2001.
17. Craine W. L., *Fuzzy Hypergraphs and Fuzzy Intersection Graphs*, PhD Thesis, University of Idaho, 1993.
18. Dănciulescu Daniela, Dănciulescu Daniel, Pătruț Bogdan, Țugui Alexandru, **Mohammed Saeed Ali Amer**, *Towards Medical Neutrosophic KRP Systems*, SMART 2018, https://www.edusoft.ro/smart2018/the-conference/programme/, Iunie 2018, Acceptat pentru publicare în nr. 9(2018) al revistei ***BRAIN*** - *Broad Research in Artificial Intelligence and Neuroscience*, ISSN 2067-3957.
19. Dau F., *Concept Graphs without Negations: Standard models and Standard graphs*, LNCS 2746, pp. 243-256, 2003.
20. Deli I., Subas Y., *A ranking method of single valued neutrosophic numbers and its applications to multi-attribute decision making problems*, Int. J. Mach. Learn. & Cyber, DOI 10.1007/s13042-016-0505-3, 2016.
21. Despi I., Opris D., Yalcin E., *Generalised Atanassov Intuitionistic Fuzzy Sets*, eKNOW 2013: The Fifth International Conference on Information, Process, and Knowledge Management, IARIA, pp. 51-56, 2013.
22. Devadoss N., Ramakrisnan S., *Knowledge Representation using Fuzzy Ontologies - A Review*, International Journal of Computer Science and Information Technologies, 6(5), pp. 4304-4308, 2015.
23. Efe A., *Unearthing and Enhancing Intelligence and Wisdom Within the COBIT 5 Governance of Information Model*, COBIT, 2016.
24. Ejegwa P., Akowe S. O., Otene P. M., Ikyule J. M., *An Overview On Intuitionistic Fuzzy Sets*, International Journal of Scientific & Technology Research, 3(3), pp. 142-145, 2014.
25. Engelbrecht A.P., *Computational Intelligence. An Introduction*, John Wiley & Sons Ltd, 2007 (2nd ed.)
26. Ensing M., Paton R., Speel P.-H. şi Rada R., *An object-oriented approach to knowledge representation in a biomedical domain*, Artificial Intelligence in Medicine 6, pp. 459-482, 1994.
27. Feigenbaum A.E., https://exhibits.stanford.edu/feigenbaum: *The Edward A. Feigenbaum Papers*. Work in artificial intelligence and computer science at Stanford University, ultimul acces Decembrie 2017.
28. Fensel D., van Harmelen F., Horrocks I., McGuinness D.L., Patel-Schneider P.F., *OIL: An Ontology Infrastructure for the Semantic Web*, http://www.cs.man.ac.uk/ horrocks/Publications/download/2001/IEEE-IS01.pdf, 2001.
29. Friedman-Hill E., *Jess in Action: Java Rule-Based Systems*, Manning Publications, 2003.
30. Giarratano J. C., Riley G. D., *Expert Systems: Principles and Programming*, Course Technology, 2004.

31. Gray S. A., Zanre E., Gray S. R. J., *Fuzzy Cognitive Maps as Representations of Mental Models and Group Beliefs*. In: Papageorgiou E. (eds) Fuzzy Cognitive Maps for Applied Sciences and Engineering, Intelligent Systems Reference Library, vol 54. Springer, Berlin, Heidelberg, 10.1007/978-3-642-39739-42, 2014.
32. Han J., Huang Y., Cercone N., Fu Y., *Intelligent Query Answering by Knowledge Discovery Techniques*, IEEE Transactions on Knowledge and Data Engineering, 8(3), pp. 373-390, 1996.
33. Hefley W.E., Murray D., *Intelligent User Interfaces*, In Proceedings of IUI'93, Orlando, Florida, ACM Press, NY, pp. 3-10, 1993.
34. Hobbes T., *Elements of Law*, Natural and Political, Routledge, 1969.
35. Hong L., Zou L., Lian X., Yu P.S., *Subgraph Matching with Set Similarity in a Large Graph Database*, IEEE Transactions on Knowledge and Data Engineering 27(9), pp. 2507-2521, 2015.
36. Horvitz E., *Principles of Mixed-Initiative User Interfaces*, In Proc. of CHI, pp. 159-166, 1999.
37. Jayaram N., Khan A., Li Ch., Yan X., Elmasri R., *Querying Knowledge Graphs by Example Entity Tuples*, arXiv: 1311.2100v1 [cs.DB], 2013.
38. Jurafsky D., Martin J. H., *Question Answering*. In Speech and Language Processing (draft, 3rd ed), https://web.stanford.edu/~jurafsky/slp3/28.pdf (ianuarie 2017).
39. Kamsu-Foguem B., Diallo G., Foguem C., *Conceptual graph-based knowledge representation for supporting reasoning in African traditional medicine*, http://dx.doi.org/10.1016/j.engappai.2012.12.004, 2012.
40. Karmakar S., Bhunia A.K., *A Comparative Study of Different Order Relations of Intervals*, Reliable Computing, 16, pp. 3872, 2012.
41. Karunambigai M. G., Parvathi R., and Buvaneswari R., *Arcs in intuitionistic fuzzy graphs*, Notes of Intuitionistic Fuzzy Sets 18 (4), pp.48-58, 2012.
42. Kent R. E., *Conceptual Knowledge Markup Language (CKML): An introduction*, Netnomics 2, pp. 139–169, 2000.
43. Kistner G., Nuernberger Ch., *Developing User Interfaces using SCXML Statecharts*, In Proceedings of the 1st EICS Workshop on Engineering Interactive Computer Systems with SCXML, NVIDIA, Publication Rights Licensed to ACM, http://www.webmail.phrogz.net/files/Developing%20User%20Interfaces%20using%20SCXML%20Statecharts.pdf, 2014.
44. Kurfess Franz J., *Neural Networks and Structured Knowledge: Knowledge Representation and Reasoning*, Applied Intelligence 11(1), pp. 5-13, 1999.
45. Legg Sh., *Machine Super Intelligence*, PhD Thesis, Faculty of Informatics of the University of Lugano, 2008.
46. Lenzerini M., D. Nardi D., Simi M., *Inheritance Hierarchies in Knowledge Representation and Programming Languages*, John Wiley and Sons, 1991.
47. Levesque H.J., Brachman, R.J., *A fundamental tradeoff in knowledge representation and reasoning*. In: R.J. Brachman and H.J. Levesque (Eds.), Readings in knowledge representation, Morgan Kaufmann, USA, 1985.
48. Liew A., *DIKIW: Data, Information, Knowledge, Intelligence, Wisdom and their Interrelationships*, Business Management Dynamics, 2(10), pp.49-62, 2013.

49. Liu P., Chu Y., Li Y., Chen Y., *Some Generalized Neutrosophic Number Hamacher Aggregation Operators and Their Application to Group Decision Making*, International Journal of Fuzzy Systems, 16(2), pp. 242-255, 2014.
50. Lucas P. J. F., van der Gaag L. C., *Principles of Expert Systems*, Addison-Wesley, 1991.
51. Maletic I. J., Marcus A., *Data Cleansing: A Prelude to Knowledge Discovery*, In O. Maimon, L. Rokach (eds.), Data Mining and Knowledge Discovery Handbook, 2nd ed., DOI 10.1007/978-0-387-09823-4_2, Springer Science+Business Media, LLC 2010.
52. Meenakshi A., Aghila R., Suganthi P., S. Kavya S., *A Knowledge Representation Technique for Intelligent Storage and Efficient Retrieval using Knowledge based Markup Language*, Indian Journal of Science and Technology, 9(8), DOI: 10.17485/ijst/2016/v9i8/87955, pp. 1-8, 2016.
53. Merritt D., *Building Expert Systems in Prolog*, Amzi! Inc, 2000 (Springer Verlag, 1989).
54. **Mohammed Saeed Ali Amer**, Referat 1- *Metode tradiţionale pentru reprezentarea şi procesarea cunoaşterii*, Școala doctorală, Universitatea din Pitești. Materialul a stat la baza lucrării [58].
55. **Mohammed Saeed Ali Amer**, Referat 2 - *Sisteme pentru reprezentarea şi procesarea cunoaşterii*, Școala doctorală, Universitatea din Pitești. Noile dezvoltări au fost prezentate în Mohammed Saeed Ali Amer, 2017, *Systems for the representation and processing of knowledge*, The Ninth Doctoral Student Workshop in Computer Science, Universitatea din Pitești, 19-20 Mai, 2017.
56. **Mohammed Saeed Ali Amer**, Referat 3 - *Modele şi tehnologii Markup utilizate în reprezentarea şi procesarea cunoaşterii*, Școala doctorală, Universitatea din Pitești. Materialul a permis expunerea [57] și a stat la baza dezvoltărilor din [61].
57. **Mohammed Saeed Ali Amer**, *Using XML and Java technologies for knowledge base interrogation*, The Sixth Doctoral Student Workshop in Computer Science, Universitatea din Pitești, 16-17 Mai, 2014.
58. **Mohammed Saeed Ali Amer**, *Traditional Approaches in Knowledge Representation*, Analele Universității SPIRU HARET, Seria Matematică-Informatică, 11(2), pp. 5-16, 2015.
59. **Mohammed Saeed Ali Amer**, *Recent Approaches in Knowledge Representation*, Analele Universității SPIRU HARET, Seria Matematică-Informatică, 12(1), pp. 5-12, 2016.
60. **Mohammed Saeed Ali Amer,** *On Measuring the Intelligence Level of Knowledge Representation and Processing Systems*, Analele Universității SPIRU HARET, Seria Matematică-Informatică, 12(2), pp. 31-38, 2016.
61. **Mohammed Saeed Ali Amer**, *Intelligent Interfaces for Knowledge Representation and Processing Systems*, Proceedings of ICVL 2017, International Conference on Virtual Learning, pp. 370-375 (http://www.c3.icvl.eu/files/Contents.pdf, WOS).
62. **Mohammed Saeed Ali Amer,** Dănciulescu Daniela, *Modern Interfaces for Knowledge Representation and Processing Systems Based on Markup Technologies*, International Journal of Computers, Communications and Control, 13(1), pp. 117-128, WOS:000425895400009, 2018.
63. **Mohammed Saeed Ali Amer**, *Recent neutrosophic models for KRP systems*, 7th International Conference on Computers Communications and Control (ICCCC), Mai 2018 (http://dzitac.ro/files/2018/ICCCC2018_paper_58.pdf, WOS).
64. Moore R. E., *Interval Analysis*, Englewood Cliffs, New Jersey, Prentice-Hall, 1966.

65. Motro A., Yuan Q., *Querying Database Knowledge*, In Proceedings of ACM-SIGMOD International Conference on Management of Data (pp. 173-183). Atlantic City, NJ, May 23-25, 1990.
66. Mulgan G., *A machine intelligence commission for the UK: how to grow informed public trust and maximise the positive impact of smart machines*, NESTA, 2016.
67. Murray K.S., *Learning as Knowledge Integration*, PHD Disertation, University of Texas, 1995.
68. Negru V., Grigoraș Gh., Dănciulescu D., Natural Language Agreement in the Generation Mechanism based on Stratified Graphs. In Proceedings of the 7th Balkan Conference on Informatics Conference (BCI '15). ACM, New York, NY, USA, Article 36, 8 pages. DOI=http://dx.doi.org/10.1145/2801081.2801121, 2015.
69. Novak J.D., Canas A.J., *The Theory Underlying Concept Maps and How to Construct and Use Them*, Technical Report IHMC CmapTools 2006-01 Rev 2008-01, http://cmap.ihmc.us/docsţheory-of-concept-maps.
70. Noy N., McGuinness D., Amir E., Baral C., Beetz M., Bechhofer S., Boutilier C., Cohn A., de Kleer J., Dumontier M., Finin T., Forbus K., Getoor L., Gil Y., Hein J., Hitzler P., Knoblock C., Kautz H., Lierler Y., Lifschitz V., Patel-Schneider P.F., Piatko C., Riecken D., Schildhauer M., *Research Challenges and Opportunities in Knowledge Representation*, http://corescholar.libraries.wright.edu/cse/218, 2013.
71. Popirlan C.I., *A solution based on intelligent software agents to improve the data searching in the contact centers*. In: Proceedings - 2011 6th IEEE Joint International Information Technology and Artificial Intelligence Conference, ITAIC 2011 2, 1-5, 2011.
72. Popirlan C.I., Popirlan C., *A Mobile Agents approach for 2D geometrical figures recognition*. 2nd International Conference on Computer Science and Information Technology 2009 (ICCSIT 2009), August 08-11, 2009.
73. Popirlan C.I., Țăndăreanu N., *An Extension of Inheritance Knowledge Bases and Computational Properties of their Answer Functions*. Annals of the University of Craiova, Mathematics and Computer Science Series 35, 149-170, 2008.
74. Ramirez C., Valdes B., *A General Knowledge Representation Model of Concepts*, Ch3 in Ramirez C. (editor): Advances in knowledge representation, Intech, 2012.
75. Ribaric S., Pavesic N., *An Inheritance Procedure for a Knowledge Representation Scheme Based on Fuzzy Petri Nets*. In: Proceedings of the Third International Conference on Natural Computation 1, 3-9, 2007.
76. Robinson, A., *Non-Standard Analysis*, Princeton University Press, Princeton, NJ, 1996.
77. Rowley J., *The wisdom hierarchy: representation of the DIKW hierarchy*, Journal of Information and Communication Science 33(2), pp. 163-180, 2007.
78. Sandberg A., Bostrom N., *Machine Intelligence Survey*, Technical Report #2011-1, Future of Humanity Institute, Oxford University, pp. 1-2, 2011.
79. Salih D., *Natural User Interfaces*, LM Research Topics in HC, http:// www. cs.bham.ac.uk/~rjh/courses/ResearchTopicsInHCI/2014-15/Submissions/Dilman%20 Salih.pdf, 2014.
80. Siedushev O., Burov E., *Forms of fuziness in data and knowledge bases*, LPNU, http://science.lpnu.ua/sites/default/files/journal-paper/2017/may/2450/burovsedushev-en.pdf, 2017.
81. Siegel N., Goolsbey K., Kahlert R., Matthews G., *The Cyc System: Notes on Architecture*, Cycorp, 2004.

82. Simou N., Stoilos G., Tzouvaras V., Stamou G., Kollias S., *Storing and Querying Fuzzy Knowledge in the Semantic Web*, Proceedings of the *Fourth International Conference on Uncertainty Reasoning for the Semantic Web*, Volume 423, pp. 83-94, 2008.
83. Smarandache F., *Neutrosophic Set - A Generalization of the Intuitionistic Fuzzy Set*, GALLUP, University of New Mexico, http://fs.gallup.unm.edu/ifsgeneralized.pdf.
84. Smarandache F., *A unifying field in logics: neutrosophic logic. Neutrosophy, neutrosophic set, neutrosophic probability and statistics* (6th edition), Ann Arbor: ProQuest Information & Learning, 2007.
85. Smarandache F., *Neutrosophic Set - A Generalization of the Intuitionistic Fuzzy Set*, IEEE International Conference on Granular Computing, doi: 10.1109/GRC.2006.1635754, 2006.
86. Smarandache F., *(T, I, F) - Neutrosophic Structures*, Acta Electrotehnica 57(1-2), Proceedings of SISOM & ACOUSTICS 2015, pp. 21-28, 2016.
87. Smarandache F., *Subtraction and Division of Neutrosophic Numbers*, Critical Review. Volume XIII, pp. 103-110, 2016.
88. Smarandache F., *Neutrosophic Quadruple Numbers, Refined Neutrosophic Quadruple Numbers, Absorbance Law, and the Multiplication of Neutrosophic Quadruple Numbers*, Neutrosophic Sets and Systems 10, 96–98, 2015.
89. Sora J.C., Sora S.A., *Artificial Education: Expert systems used to assist and support 21st century education*, GSTF Journal on Computing (JoC), 2(3), DOI: 10.5176/2010-3043_2.3.177, 2012.
90. Sowa J.F., *Conceptual Graphs*, IBM Journal of Research and Development, 1976.
91. Sowa J.F., *Conceptual Structures: Information Processing in Mind and Machine*, Addison-Wesley, 1984.
92. Sowa J.F., *Conceptual Graphs*, In Handbook of Knowledge Representation (eds. F. van Harmelen, V. Lifschitz, and B. Porter), Elsevier, 2008, pp. 213−237.
93. Sowa J.F., *Semantic Networks*, In Encyclopedia of Artificial Intelligence (ed. S.C. Shapiro), Wiley and Sons, 1987.
94. Suchanek F. M., Kasneci G., Weikum G., *YAGO: a core of semantic knowledge unifying WordNet and Wikipedia*. In WWW2007, http://www2007.org/papers/paper391.pdf, 2007.
95. Sudsawad P., *Knowledge translation: Introduction to models, strategies, and measures*. Austin, TX: Southwest Educational Development Laboratory, National Center for the Dissemination of Disability Research, 2007.
96. Sunitha M. S., Mathew S., *Fuzzy Graph Theory: A Survey*, Annals of Pure and Applied Mathematics, 4(1), pp. 92-110, 2013.
97. Sunitha M. S., *Studies on Fuzzy Graphs*, PhD Thesis, Cochin University of Science and Technology, 2001.
98. Swetnam M., Hummel R., Mueller C., Syers P., *Intelligence Complexity*, Potomac Institute for Policy Studies, 2016.
99. Țăndăreanu N., *Communication by Voice to Interrogate an Inheritance Based Knowledge System*. Research Notes in Artificial Intelligence and Digital Communications, 7th International Conference on Artificial Intelligence and Digital Communications 107, 1-15, 2007.
100. Țăndăreanu N., *Extended Inheritance Knowledge Bases and their Interrogations by Client-Server Technology*. Annals of the University of Craiova, Mathematics and Computer Science Series 37, no. 3, 1-11, 2010.

101. Țăndăreanu N., *Inheritance-based knowledge systems and their answer functions computation using lattice theory*. Romanian Journal of Information Science and Technology 6, no. 1-2, 227-248, 2003.
102. Țăndăreanu N., Popirlan C.I.: *Factorization of an inheritance knowledge base (I).* Annals of the University of Craiova, Mathematics and Computer Science Series 37, no. 2, 62-74, 2010.
103. Țăndăreanu N., Popirlan C.I., *Factorization of an Inheritance Knowledge Base (II),* Annals of the University of Craiova, Mathematics and Computer Science Series 37, no. 4, 2010.
104. Tettamanzi A.G.B., *A Fuzzy Frame-Based Knowledge Representation Formalism*, In: Di Ges V., Masulli F., Petrosino A. (eds) Fuzzy Logic and Applications. WILF 2003. "Lecture Notes in Computer Science", vol 2955. Springer, Berlin, Heidelberg, 2006.
105. Tifie S., *Defining Medical Concepts by Linguistic Variables with Fuzzy Arden Syntax*, AMIA 2002 Anual Symposium Proceedings, pp. 796-800, 2002.
106. Tomar D., Agarwal S., *A Survey on Pre-processing and Post-processing Techniques in Data Mining*, International Journal of Database Theory and Application, 7(4), 2014, pp. 99-128.
107. Tsukamoto M., Nishio S., *Inheritance reasoning by regular sets in knowledge bases with dot notation*, Deductive and Object-Oriented Databases. Lecture Notes in Computer Science 1013, 247-264, 1995.
108. Vasantha Kandasamy W.B., Ilanthenral K., Smarandache F., *Neutrosophic Graphs: A new dimension to graph theory*, Europa Nova, 2015.
109. Vasantha Kandasamy W.B., Smarandache F., *Fuzzy Cognitive Maps and Neutrosphic Cognitive Maps*, XIQUAN, Phoenix, 2003.
110. Vasantha Kandasamy W.B., *Smarandache Neutrosophic Algebraic Structures*, Hexis, 2006.
111. Văduva I., Albeanu G., *Introducere în modelarea fuzzy*, Editura Universității din București, 2004.
112. Vohra A., Vohra D., *Pro XML Development with Java Technology*, Apress, 2006.
113. Vygotsky L., *Thought and Language*, MIT Press, 1986.
114. Walaszek-Babiszewska Anna, *Linguistic Knowledge Representation for Stochastic Systems*, Proceedings of the International Multiconference on Computer Science and Information Technology, pp. 141 150, 2007.
115. Wielemaker J., Anjewierden A., *Programming in XPCE/Prolog*, University of Amster-dam, http://www.swi-prolog.org/download/xpce/doc/userguide/userguide.pdf, 2002
116. Wu W., Li H., Wang H., Zhu K.Q., ***Probase**: a probabilistic taxonomy for text understanding*, In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, pp. 481–492, 2012.
117. Yang G., and Kifer M., *Well-Founded Optimism: Inheritance in Frame-Based Knowledge Bases*, Lecture Notes in Computer Science, in On the Move to Meaningful Internet Systems 2519, 1013-1032, 2008.
118. Yao Y., Zeng Y., Zhong N., Huang X. *Knowledge Retrieval (KR),* In: Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence, pp. 729-735, 2007.

119. Ye J., *Trapezoidal neutrosophic set and its application to multiple attribute decision-making*, Neural Comput & Applic 26, pp. 11571166, DOI 10.1007/s00521-014-1787-6, 2015.
120. Zadeh L.A., *Fuzzy sets*, Information and Control, 8, pp. 338-353, 1965.
121. Zhao J., Boley H., Dong J., *A Fuzzy Logic-Based Approach to Uncertainty Treatment in the Rule Interchange Format: From Encoding to Extension*, In: Bobillo F. et al. (eds) Uncertainty Reasoning for the Semantic Web II. Lecture Notes in Computer Science, vol 7123, Springer, Berlin, Heidelberg, 2013.
122. Zimmermann H.-J., *Fuzzy Set Theory and Its Applications*, Springer Science+Business Media, LLC, 4ed, 2001.
123. ***, CLIPS, http://clipsrules.sourceforge.net/
124. ***, CMAP: http://cmap.ihmc.us/cmaptools/cmaptools-download/
125. ***, Conceptual Graph Standard (CGS): https://www.w3.org/DesignIssues/Sowa/cgstand.htm
126. ***, CYC Developer Center (CycDC), http://dev.cyc.com/api/samples/core/query/search/
127. ***, Cyc, API Specification 1.0.0-rc5 API, http://dev.cyc.com/api/core/apidocs/overview-summary.html
128. ***, DAML tools, http://www.daml.org/tools/
129. ***, DLML, http://co4.inrialpes.fr/xml/dlml/
130. ***, EOD: *English Oxford Dictionaries*: https://en.oxforddictionaries/definition/data
131. ***, ***Java EE8*** (http://download.oracle.com/otn-pub/jcp/java_ee-8-final-eval-spec/Java EE_Platform_Spec.pdf)
132. ***, *JAVA XML parsers*, http://docs.oracle.com/javase/8/docs/api/index.html
133. ***, *MWD: Merriam-Webster Dictionary*, http://www.merriam-webster.com/dictionary/data
134. ***, *Ontology Markup Language*, http://www.ontologos.org/OML/OML%200.3.htm
135. ***, *Prolog Objects* (PO), https://sicstus.sics.se/sicstus/docs/3.7.1/html/sicstus_35.html
136. ***, *SANDIA*, Jess API, http://herzberg.ca.sandia.gov/docs/70ăpi/overview-tree.html, 2007.
137. ***, *Technopedia*, https://www.techopedia.com/definition/613/expert-system
138. ***, *VXML 2.0*, https://www.w3.org/tR/voicexml20/#dml1.4
139. ***, Wikipedia, *Bigdata*, https://en.wikipedia.org/wiki/Big_data
140. ***, Wikipedia, Mind Map, https://en.wikipedia.org/wiki/Mind_map
141. ***, *XML COVER PAGES*, http://xml.coverpages.org
142. ###, Online CRC editor: https://guidolx.github.io/simple-crc-app/
143. ###, DUET, http://grcinet.grci.com/maria/www/codipsite/Tools/Components.html (Alte proiecte DAML: http://www.daml.org/2003/08/lifecycle/webscriptercolor.html)
144. ###, UBOT, http://www.daml.org/meetings/2004/12/pi/Lockheed_Martin.pdf
145. ###, Platforma Protégé, https://protege.stanford.edu/products.php
146. ###, Platforma Ontolingua, http://www.ksl.stanford.edu/software/ontolingua/
147. ###, Specificații SCXML: https://www.w3.org/TR/scxml/
148. ###, Editor SCXML: https://doc.qt.io/qtcreator/creator-scxml.html

149. ###, VXML 3.0, https://www.w3.org/TR/2010/WD-voicexml30-20101216/
150. ###, SRGS, https://www.w3.org/TR/speech-grammar/