METODOLOGII PENTRU DEZVOLTAREA SISTEMELOR BAZATE PE AGENŢI INTELIGENŢI

prof. dr. ing. Mihaela M. Oprea

m_oprea@yahoo.com

Universitatea Petrol-Gaze din Ploieşti, Catedra de Informatică

Rezumat: Tehnologia agenţilor inteligenţi s-a impus în ultimii ani ca o soluţie eficientă pentru dezvoltarea sistemelor software complexe, distribuite, care lucrează în medii dinamice şi deschise. Pentru o bună inginerie a acestor sisteme bazate pe agenţi inteligenţi s-au creat diferite metodologii, majoritatea specifice anumitor tipuri de aplicaţii. Lucrarea prezintă o sinteză a principalelor metodologii bazate pe agenţi care s-au impus pe plan mondial.

Cuvinte cheie: agenţi inteligenţi, sisteme multiagent, inginerie software, metodologii bazate pe agenţi.

Abstract: The intelligent agents technology became in the last years one of the most efficient software technology that provide solutions for the development of complex, distributed software systems that work in dynamic and open environments. Different methodologies were created for a good engineering of these agent-based systems, most of them being specific to certain types of applications. The paper presents a synthesis and a brief analysis study of the main agent-based methodologies that were reported so far in the literature.

Key words: intelligent agents, multiagent systems, software engineering, agent-based methodologies.

1. Introducere

Multe dintre sistemele software complexe, distribuite (geografic), dezvoltate în ultimii ani utilizează tehnologia agenţilor inteligenţi. Aplicaţiile variază de la cele industriale (sisteme de monitorizare şi control, sisteme de diagnoză, sisteme de modelare şi simulare a lanţului de furnizori, sisteme de fabricaţie) până la cele virtuale, din Internet (sisteme de comerţ electronic, licitaţii electronice, afaceri electronice, organizaţii virtuale, web semantic). O bună inginerie software a acestor sisteme bazate pe agenţi inteligenţi presupune utilizarea unor metodologii, instrumente, metode şi tehnici (de modelare, de notaţie, de verificare) specifice tehnologiei agenţilor inteligenţi. Ingineria software dispune de metodologii, instrumente, metode şi tehnici bine puse la punct, care pot fi extinse pentru a fi aplicate la dezvoltarea sistemelor bazate pe agenţi inteligenţi. De exemplu, ingineria software orientată pe obiecte este utilizată ca punct de pornire, fiind adaptată cerinţelor specifice sistemelor bazate pe agenţi. Un alt exemplu este dat de ingineria software orientată pe componente. Cu toate acestea, experienţa acumulată la nivel mondial în analiza, proiectarea şi implementarea sistemelor bazate pe agenţi inteligenţi a condus la dezvoltarea unei inginerii software bazate pe agenţi inteligenţi. Există la ora actuală numeroase cercetări care au drept scop fundamentarea, standardizarea şi unificarea diferitelor metodologii, instrumente, metode şi tehnici ale ingineriei software bazate pe agenţi, pentru a oferi o teorie cât mai completă şi eficientă în dezvoltarea sistemelor bazate pe agenţi inteligenţi. În literatura de specialitate au fost prezentate deja o parte din rezultatele acestor cercetări. Astfel, a apărut un manual al ingineriei software bazate pe agenţi inteligenţi [1], în care sunt prezentate sintetic concepte şi abstractizări, metodologii, instrumente, infrastructuri şi metode noi de inginerie a software-ului bazat pe agenţi.

Întrucât sistemele bazate pe agenţi inteligenţi au o serie de caracteristici specifice (autonomia, cooperarea, mobilitatea, scalabilitatea, comunicarea, dinamica, coordonarea, complexitatea, securitatea, adaptabilitatea, deschiderea, etc.), a apărut necesitatea creării de noi metodologii, dedicate sistemelor software bazate pe agenţi, metodologii orientate pe agenţi, mult mai potrivite decât metodologiile orientate pe obiecte. Lucrarea prezintă o sinteză a principalelor metodologii orientate pe agenţi care s-au impus în literatura de specialitate: Gaia, Tropos, MaSE, Prometheus, Zeus. Sunt descrise etapele de bază şi caracteristicile acestor metodologii.

Lucrarea cuprinde patru secţiuni. După secţiunea introductivă, în secţiunea 2 sunt prezentate elementele fundamentale referitoare la sistemele multiagent. Secţiunea 3 prezintă aspectele definitorii ale ingineriei software bazate pe agenţi în raport cu ingineria software bazată pe

obiecte. De asemenea, sunt descrise etapele principale ale dezvoltării unui sistem software bazat pe agenţi. În secţiunea 4 sunt prezentate succint şi analizate cele mai importante metodologii orientate pe agenţi care s-au impus pe plan mondial. În ultima secţiune sunt sintetizate concluziile lucrării.

2. Sisteme multiagent

Modelarea sistemelor software complexe, distribuite, se poate realiza cu ajutorul sistemelor multiagent. Un sistem multiagent este un sistem bazat pe agenţi inteligenţi, alcătuit din minimum doi agenţi inteligenţi care au un scop global de îndeplinit, eventual, scopuri locale şi care au abilităţi pentru rezolvarea scopului global. Altfel spus, un sistem multiagent grupează agenţi inteligenţi cu abilităţi similare şi / sau diferite care pot rezolva probleme complexe, lucrând în mod eficient, în special, în cazul sistemelor distribuite. Practic, rezolvarea scopului global se realizează prin distribuirea subscopurilor acestuia local, către agenţii din sistem, care vor rezolva subscopurile pe baza abilităţilor lor. Ideea care stă la baza modelării multiagent este dezvoltarea de module separate ce furnizează o soluţie şi cooperarea modulelor pentru a rezolva probleme complexe prin schimb de informaţie. Modularitatea şi cooperarea permit o administrare mai facilă a rezolvării problemelor complexe. Figura 1 prezintă schematic ecuaţia unui sistem multiagent, ce cuprinde agenţii inteligenţi şi comunicarea, ca modalitate de interacţiune între agenţi în vederea realizării task-urilor asociate sistemului multiagent.

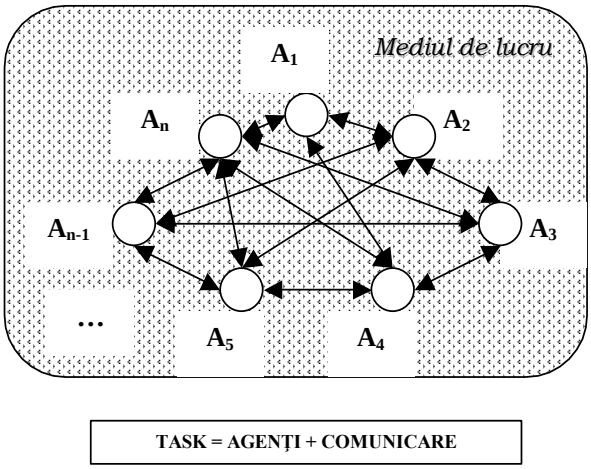

Figura 1. Ecuaţia unui sistem multiagent

Un agent inteligent este o entitate autonomă, virtuală sau fizică, care percepe mediul de lucru, are iniţiativă, comunică cu alte entităţi similare şi realizează acţiuni asupra mediului de lucru. Agenţii dintr-un sistem bazat pe agenţi partajează anumite resurse şi informaţii referitoare la activităţile lor. Gruparea agenţilor inteligenţi în sisteme multiagent implică utilizarea unor mecanisme de coordonare ce trebuie să conducă la o comportare coerentă a sistemului, conformă cu scopul global al acestuia.

Agenţii inteligenţi pot utiliza diferite categorii de arhitecturi agent: deliberative, reactive şi hibride. Arhitecturile deliberative sunt arhitecturi simbolice prin care agenţii păstrează un model complet de reprezentare a mediului lor de lucru. Un exemplu de astfel de arhitectură este arhitectura BDI [2], care are un suport teoretic foarte bun, bazat pe un formalism matematic bine pus la punct, dar care nu este o arhitectură practică pentru sistemele complexe, ce lucrează în timp real. Întrucât un agent bazat pe arhitectura BDI păstrează un model complet al lumii reale, timpul lui de răspuns este relativ mare. Arhitecturile reactive au la bază modelul de raţionare fără reprezentare, ele fiind specifice agenţilor inteligenţi de tip roboţi mobili. Agenţii iau decizii la momentul execuţiei, pe baza unei cantităţi limitate de informaţie şi a unor reguli simple de tip situaţie-acţiune. Deciziile agenţilor reactivi sunt bazate direct pe intrările de la senzori, timpul de răspuns fiind mic. Arhitecturile hibride combină avantajele celor două

arhitecturi, deliberativă şi reactivă şi evită dezavantajele lor. Exemple de arhitecturi hibride sunt PRS, COSY, TOURING MACHINES, INTERRAP [3].

O componentă esenţială a unui sistem multiagent este mediul de lucru în care îşi desfăşoară activitatea agenţii inteligenţi. Structura mediului de lucru depinde de domeniul de aplicaţie şi poate cuprinde alţi agenţi şi alte sisteme sau module non-agent. Mediul de lucru poate fi accesibil sau inaccesibil, determinist sau nedeterminist, episodic sau ne-episodic, static sau dinamic, discret sau continuu. În dezvoltarea sistemelor bazate pe agenţi inteligenţi trebuie să se ţină cont atât de caracteristicile mediului de lucru, cât şi de caracteristicile pe care agenţii trebuie să le aibă în mediul respectiv de lucru, pentru a rezolva problemele din domeniul lor de expertiză. Pe lângă caracteristicile de bază, autonomie, reactivitate, proactivitate şi abilitate socială (comunicare), agenţii inteligenţi pot avea şi alte caracteristici suplimentare: mobilitate, sinceritate, raţionalitate, bunăvoinţă, cooperare, flexibilitate, adaptare, învăţare, etc.

Pentru realizarea task-urilor care le sunt asignate, agenţii trebuie să comunice între ei într-un limbaj de comunicare agent (de exemplu, FIPA ACL, KIF, KQML). De asemenea, agenţii trebuie să utilizeze aceeaşi ontologie (un vocabular cu termeni specifici domeniului de aplicaţie) pentru a înţelege corect mesajele primite sau transmise şi să folosească protocoale de comunicaţie (de exemplu, TCP/IP, SMTP, HTTP).

Sistemele multiagent pot fi sisteme închise sau deschise. Într-un sistem închis agenţii nu părăsesc sistemul şi nici alţi agenţi din exteriorul sistemului multiagent nu pot intra în sistem. Într-un sistem deschis agenţii pot ieşi/reintra din/în sistemul multiagent şi de asemenea, alţi agenţi din exterior pot intra/ieşi în/din sistem.

Dezvoltarea unui sistem multiagent presupune rezolvarea unor probleme importante: coordonarea agenţilor, comunicarea între agenţi, cooperarea agenţilor, planificarea task-urilor etc. O soluţie eficientă este dată de utilizarea unor infrastructuri specifice pentru coordonare, comunicare, care pot fi oferite de platformele agent. Standardul FIPA [4] furnizează astfel de facilităţi prin platforma agent FIPA, care a fost preluată de majoritatea mediilor de dezvoltare a sistemelor multiagent.

3. Ingineria software bazată pe agenţi

De-a lungul timpului au fost propuse o serie de modalităţi de dezvoltare a sistemelor software, structurale, orientate pe componente şi orientate pe obiecte. Dezvoltarea primelor sisteme multiagent a fost realizată prin utilizarea metodologiilor orientate pe obiecte. Implementarea acestor sisteme cu ajutorul obiectelor nu a fost cea mai potrivită soluţie, întrucât obiectele nu au proprietăţile specifice agenţilor inteligenţi, proactivitate, autonomie, componentă socială (de comunicare inter-agent). Astfel, analiza, proiectarea şi implementarea sistemelor bazate pe agenţi inteligenţi a condus în ultimii ani la apariţia unei noi paradigme a ingineriei software, ingineria software bazată pe agenţi, care cuprinde abstractizări (ale agenţilor, protocoalelor de interacţiune, mediului de lucru, contextului, etc.), metodologii şi instrumente specifice sistemelor multiagent.

În general, dezvoltarea sistemelor multiagent se realizează cu ajutorul unei anumite metodologii. În prezent, există două categorii de metodologii pentru dezvoltarea sistemelor bazate pe agenţi: metodologii bazate pe ingineria cunoaşterii (de exemplu, DESIRE, MAS-CommonKADS) şi metodologii ale ingineriei software (de exemplu, Gaia, Prometheus, Tropos). Aceste metodologii acoperă etapele de analiză şi proiectare a unui sistem multiagent, iar unele dintre ele oferă şi suport software pentru implementare.

Implementarea agenţilor inteligenţi şi a sistemelor multiagent se poate realiza cu ajutorul unor medii de dezvoltare care sunt disponibile gratuit pentru mediul academic (FIPA-OS, JADE, ZEUS) sau a unor sisteme comerciale (JACK, AgentBuilder). Majoritatea acestor pachete software utilizează limbajul Java şi modelează agenţii utilizând paradigma programării orientată pe obiecte, la care sunt adăugate componentele specifice agenţilor inteligenţi (comunicare, coordonare, reprezentarea noţiunilor mentale, a cunoaşterii, etc.).

Ingineria software presupune parcurgerea a cinci etape principale în dezvoltarea unui sistem software: analiza, proiectarea, implementarea, testarea şi întreţinerea sistemului. Înlănţuirea acestor etape se realizează conform modelului adoptat pentru dezvoltarea software-ului. Dintre modelele cele mai cunoscute, menţionăm: modelul cascadă, modelul cascadă cu reacţie înapoi şi modelul spirală. Întrucât modelul cascadă este cel mai utilizat model, ne vom referi doar la acesta. Figura 2 prezintă schematic etapele dezvoltării unui sistem software în cadrul modelului cascadă. Etapa de analiză a fost împărţită în două sub-etape: stabilire cerinţe (ale utilizatorilor – analiza timpurie a cerinţelor) şi specificaţii (cerinţe din punctul de vedere al dezvoltatorului sistemului software – analiza târzie a cerinţelor). În unele metodologii, etapa de proiectare este şi ea împărţită în două sub-etape: proiectare arhitecturală şi proiectare detaliată.

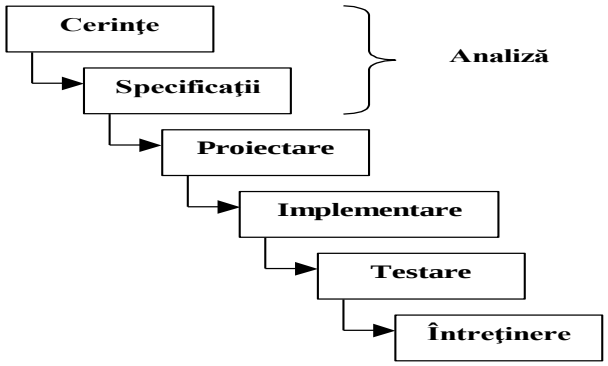

Figura 2. Modelul cascadă pentru dezvoltarea software-ului

Prezentăm succint câteva consideraţii referitoare la etapele de analiză, proiectare, implementare şi testare a sistemelor bazate pe agenţi inteligenţi.

Etapa de analiză specifică ce anume trebuie să facă sistemul bazat pe agenţi, pornind de la analiza cerinţelor utilizatorului. Astfel, sunt analizate şi cerinţele sistemului, care cuprind cerinţe funcţionale şi ne-funcţionale. Această etapă poate fi divizată în două sub-etape: cerinţe şi specificaţii sau analiza timpurie a cerinţelor şi analiza târzie a cerinţelor. În urma parcurgerii acestei etape se generează o reprezentare preliminară a sistemului bazat pe agenţi şi a unor scenarii de utilizare, fie sub forma unor specificaţii formale, fie sub forma unor modele şi diagrame, fie printr-o combinaţie a acestora. Dintre metodele formale utilizate în cadrul acestei etape menţionăm metoda i* [5] (care utilizează actori, scopuri, cunoştinţe, angajamente pentru a modela medii organizaţionale şi sistemele lor de informare) şi limbajele pure de specificaţie. Dintre metodele grafice amintim metode bazate pe UML (AUML [6]) şi reţelele Petri.

Proiectarea unui sistem bazat pe agenţi presupune rafinarea reprezentărilor obţinute în cadrul etapei de analiză. Astfel, sunt rafinate diagramele şi/sau se creează o bibliotecă de componente. Etapa de proiectare se poate realiza cu sau fără ajutorul unui mediu de dezvoltare, care traduce diagramele în cod sursă. Exemple de medii de dezvoltare a sistemelor multiagent sunt date de ZEUS [7] şi AgentBuilder [8]. Utilizarea unui mediu de dezvoltare are avantajul dezvoltării rapide a unor prototipuri ale sistemului. Dezavantajul major este faptul că, de obicei, codul nu este documentat şi este riscantă modificarea unei componente predefinite. În plus, dezvoltarea rapidă a unui prototip nu garantează îndeplinirea cerinţelor iniţiale. Proiectarea sistemului bazat pe agenţi fără ajutorul unui mediu de dezvoltare implică alegerea unor teorii, metode şi instrumente software corespunzătoare. Astfel, se realizează selecţia unui model de agent, a unei arhitecturi agent corespunzătoare modelului de agent, se proiectează caracteristicile agentului şi se alege o platformă agent (adică, un set de servicii pentru managementul agenţilor şi comunicare). Pentru o bună inginerie software este bine să se urmărească aplicarea standardelor existente (de exemplu, FIPA).

Etapa de implementare a sistemului bazat pe agenţi constă în translatarea conceptelor proiectării în programe ce pot fi executate. Implementarea se poate realiza fie într-un limbaj convenţional (Java, C, Prolog), fie într-un limbaj orientat pe agenţi (Agent0, AgentSpeak(L)).

Testarea unui sistem bazat pe agenţi constă în identificarea problemelor şi verificarea

codului conform specificaţiei sistemului. Există o serie de metode de verificare a unui sistem multiagent, verificarea formală, metoda blackbox şi metoda whitebox [5]. De asemenea, sunt utilizate o serie de metode de depanare a sistemului, metode ce păstrează urma execuţiei sistemului multiagent, respectiv mesajele care sunt schimbate între agenţi.

4. Metodologii orientate pe agenţi

Principalele metodologii orientate pe agenţi, care au fost dezvoltate până în prezent, includ: Gaia, Tropos, MaSE, Prometheus, Zeus, ADELFE, MAS-CommonKADS, DESIRE, MESSAGE, SADDE, PASSI, INGENIAS. În această secţiune vom prezenta pe scurt o parte din aceste metodologii.

4.1 Gaia

Metodologia Gaia [9] este prima metodologie propusă pentru analiza şi proiectarea sistemelor multiagent. În cadrul acestei metodologii, un sistem multiagent este considerat ca fiind o organizaţie alcătuită din agenţi, fiecare agent având roluri specifice, care cooperează între ei pentru realizarea scopului lor comun. Prima versiune a acestei metodologii este limitată datorită faptului că ea poate fi aplicată doar sistemelor multiagent închise, în care agenţii doresc să coopereze şi datorită nerespectării standardelor ingineriei software pentru notaţiile utilizate. Metodologia a fost extinsă şi astfel au apărut variantele Gaia v.2 [10] şi ROADMAP [11]. Metodologia este aplicată după stabilirea cerinţelor şi constă în construirea unor modele care descriu sistemul multiagent la nivel macro (al societăţii de agenţi) şi la nivel micro (intra-agent).

Prezentăm succint etapele metodologiei Gaia în varianta iniţială.

În etapa de analiză se construiesc două modele: modelul rol şi modelul interacţiune, care structurează sistemul multiagent sub forma unei mulţimi de roluri abstracte ce interacţionează. În Gaia, rolurile sunt construcţii abstracte, atomice, utilizate pentru a conceptualiza sistemul, fără a avea corespondenţe concrete în sistemul implementat. O schemă de rol este o descriere semi-formală a comportamentului unui agent. Modelul rol este definit de toate schemele rol identificate. Pentru fiecare rol se defineşte o schemă rol prin specificarea a patru atribute: responsabilităţi, activităţi, permisii (resursele disponibile) şi protocoale. Modelul interacţiune constă în definirea unui protocol pentru fiecare protocol corespunzător unui rol din sistem.

În etapa de proiectare se definesc trei modele: modelul agent, modelul servicii şi modelul cunoştinţe. Aceste modele formează o specificaţie completă ce va fi utilizată în etapa de implementare. Rolurile şi protocoalele sunt mapate în construcţii concrete, tipuri de agenţi, care vor fi instanţiate la momentul rulării. Modelul agent se obţine prin asignarea de roluri fiecărui tip de agent. În Gaia, un serviciu este un bloc coerent de funcţionalitate, neutru în raport cu detaliile de implementare. Serviciile sunt determinate din activităţile şi protocoalele rolurilor. Pentru fiecare serviciu se specifică patru atribute: intrări, ieşiri, pre-condiţii şi post-condiţii. Modelul cunoştinţe este un graf directat între tipuri de agenţi, care exprimă gradul de cuplare între aceştia.

Prima variantă îmbunătăţită a metodologiei Gaia este metodologia ROADMAP [11], care a inclus o ierarhie dinamică a rolurilor (pentru sisteme multiagent deschise), modele pentru descrierea mediului de lucru şi cunoaşterea agenţilor (care a fost neglijată în versiunea iniţială a metodologiei Gaia).

A doua variantă îmbunătăţită a metodologiei Gaia este Gaia v.2 [10] în care s-au introdus: un model al mediului de lucru (ce descrie entităţi şi resurse cu care interacţionează sistemul multiagent), reguli organizaţionale (restricţii ale organizaţiei la nivel global, între roluri şi protocoale) şi structuri organizaţionale (pentru arhitectura globală a sistemului).

În figura 3 este prezentată schematic metodologia Gaia, cu completarea adusă de variantele îmbunătăţite, prin includerea unui model al mediului de lucru.

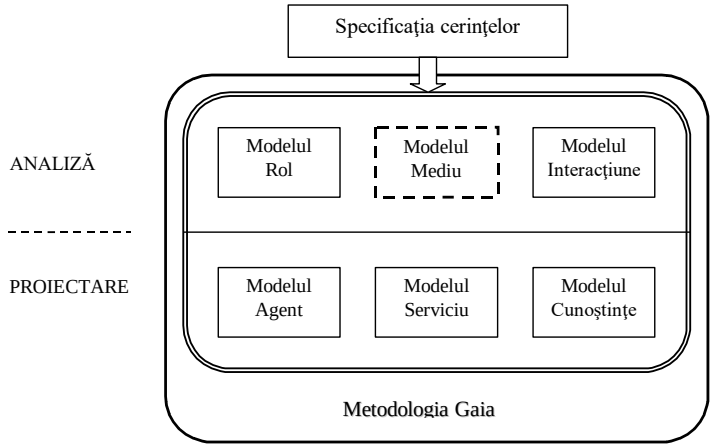

Figura 3. Metodologia Gaia – reprezentare schematică

Dezavantajul major al acestei metodologii este lipsa etapei de modelare a cerinţelor.

Principalele concepte utilizate în metodologia Gaia sunt: rol, interacţiune, agent, serviciu, cunoştinţe, activităţi, protocoale, responsabilităţi, resurse, mediu (doar în variantele ROADMAP şi Gaia v.2).

4.2 Tropos

Metodologia Tropos [12] se bazează pe ideea construirii unui model al sistemului dezvoltat şi al mediului de lucru, care este rafinat şi extins în mod incremental. Tropos cuprinde toate activităţile de analiză şi proiectare ale procesului de dezvoltare a software-ului, de la analiza domeniului de aplicaţie până la implementarea sistemului.

Etapele metodologiei Tropos sunt următoarele:

- analiza timpurie a cerinţelor;
- analiza târzie a cerinţelor;
- proiectarea arhitecturală;
- proiectarea detaliată;
- implementarea.

Toate etapele sunt bine definite în literatura de specialitate a ingineriei software. Analiza cerinţelor a fost împărţită în două etape, analiza timpurie (etapă acceptată de comunitatea cercetătorilor, dar care nu este aplicată pe scară largă) şi analiza târzie (etapă care este inclusă în majoritatea metodologiilor).

Analiza timpurie a cerinţelor presupune înţelegerea problemei prin studierea configuraţiei organizaţionale. Intenţiile utilizatorului sunt modelate sub formă de scopuri. Pornind de la scopurile iniţiale sunt determinate cerinţele funcţionale şi ne-funcţionale ale sistemului. În cadrul acestei etape se construieşte un model i* [5] al dependenţelor strategice între actori, scopuri. Un actor este o entitate care are scopuri strategice şi intenţii în cadrul sistemului sau organizaţiei. Un actor poate fi un agent, un rol sau o poziţie (o funcţie), din punct de vedere al modelării multiagent. Din punctul de vedere al aplicaţiei, un actor poate fi văzut ca fiind un sub-sistem. Un rol este o caracterizare abstractă a comportamentului unui agent. O poziţie este un set de roluri jucate de un agent. Un agent poate ocupa o anumită poziţie. Agenţii utilizează resurse (entităţi fizice sau informaţionale). Un scop este dat de interesele strategice ale actorilor. Scopurile pot fi de tip hard (cerinţe funcţionale) şi soft (cerinţe ne-funcţionale).

Analiza târzie a cerinţelor constă în descrierea sistemului sub forma unor specificaţii ale cerinţelor funcţionale şi ne-funcţionale ale sistemului. În cadrul acestei etape se construieşte

diagrama scopurilor.

Proiectarea arhitecturală defineşte arhitectura globală a sistemului în termeni de sub-sisteme (actori) interconectate prin fluxuri de date şi control (dependenţe). În cadrul acestei etape se construieşte diagrama actorilor (actori şi dependenţe între actori).

Proiectarea detaliată a sistemului multiagent constă în specificarea capabilităţilor agenţilor şi a interacţiunilor dintre agenţi. De obicei, atunci când se realizează această etapă, platforma de implementare este deja aleasă, astfel că proiectarea detaliată va fi mapată direct în codul sursă. În cadrul acestei etape se construiesc: diagrama capabilităţilor şi diagrama planurilor cu ajutorul diagramelor UML [13] şi AUML [6]. În Tropos, un plan este definit ca o modalitate de realizare a unui obiectiv (scop). Capabilitatea reprezintă abilitatea unui actor de definire, alegere şi execuţie a unui plan pentru atingerea scopului.

Implementarea sistemului presupune parcurgerea pas cu pas a specificaţiei proiectării detaliate şi utilizarea pachetului software JACK [14], de dezvoltare a sistemelor multiagent.

În figura 4 este prezentată schematic metodologia Tropos.

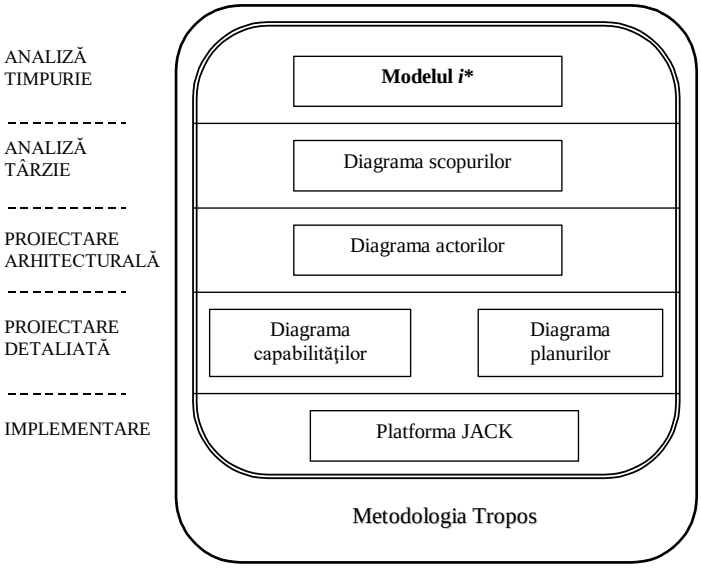

Figura 4. Metodologia Tropos – reprezentare schematică

Metodologia Tropos utilizează arhitectura BDI [2] de modelare a agenţilor.

Avantajul major al metodologiei Tropos este dat de acoperirea integrală a etapelor de analiză timpurie şi târzie a cerinţelor, precum şi a etapelor de proiectare arhitecturală şi detaliată. Din păcate, nu există, deocamdată, instrumente software care să permită tranziţia între etape. De asemenea, metodologia nu este adaptată pentru sisteme multiagent complexe, de dimensiuni mari şi nu poate fi aplicată oricărui tip de aplicaţie.

Principalele concepte utilizate în metodologia Tropos sunt: actor, dependenţă, scop (hard şi soft), plan, resursă, capabilitate.

4.3 MaSE

Metodologia MaSE [15] este o metodologie completă pentru analiza, proiectarea şi implementarea sistemelor multiagent eterogene, care utilizează modele grafice derivate din modele standard UML pentru a descrie tipurile de agenţi dintr-un sistem şi interfeţele lor cu alţi agenţi. De asemenea, MaSE furnizează o descriere detaliată a proiectării interne a agenţilor, independent de arhitectura agent utilizată.

Un sistem multiagent este văzut ca fiind o abstractizare a paradigmei orientate pe obiecte, în

care agenţii sunt obiecte specializate. Coordonarea agenţilor se realizează prin conversaţii şi acţiuni pro-active de realizare a scopurilor individuale sau globale.

Prezentăm succint etapele de analiză şi proiectare. Etapa de implementare este facilitată de utilizarea platformei agentTool [15].

În cadrul etapei de analiză se stabilesc scopurile, se identifică şi se analizează diferite studii de caz şi se rafinează rolurile. Un scop este o abstractizare a unor cerinţe detaliate. Un rol descrie o entitate care execută o anumită funcţie în cadrul sistemului. Scopurile sunt îndeplinite de roluri. Pentru fiecare rol se definesc task-urile concurente pe care trebuie să le execute. Se construiesc următoarele modele grafice: diagrama ierarhiei scopurilor, modelul rol, modelul task-uri concurente şi diagrama secvenţelor pentru studiile de caz (diagrame de interacţiune).

Proiectarea sistemului multiagent presupune asignarea de roluri claselor agent, identificarea conversaţiilor, construirea conversaţiilor, adăugarea de mesaje şi stări pentru robusteţe, definirea arhitecturii interne a fiecărui agent şi definirea structurii finale a sistemului cu ajutorul diagramelor de dezvoltare. Se construiesc următoarele modele grafice: diagrama claselor agent, diagrame de conversaţie, diagrame de arhitecturi agent, diagrame de dezvoltare.

În figura 5 este prezentată schematic metodologia MaSE.

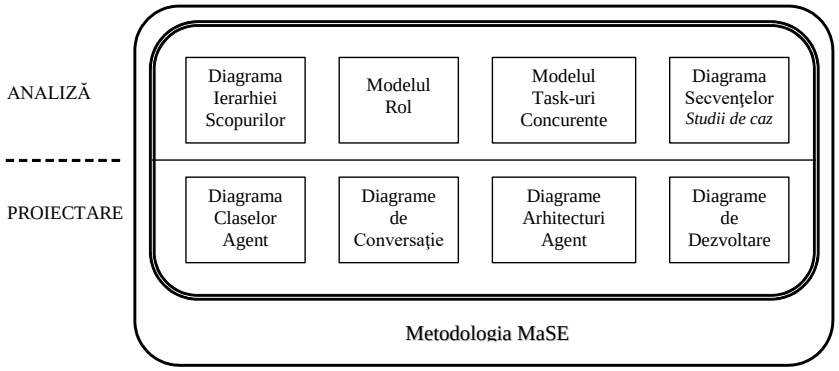

Figura 5. Metodologia MaSE – reprezentare schematică

Avantajul major al metodologiei MaSE este dat de proiectarea detaliată pentru care sunt furnizate modele şi suport tehnic.

Principalele concepte utilizate în metodologia MaSE sunt: scop, rol, agent, tip agent, conversaţie, task, reguli de organizare.

4.4 Prometheus

Metodologia Prometheus [1], [16] a fost realizată cu scopul de a facilita dezvoltarea sistemelor bazate pe agenţi atât în mediul academic, cât mai ales în mediul industrial, de către persoane fără cunoştinţe anterioare în domeniul agenţilor inteligenţi. Prometheus este o metodologie detaliată şi completă, fiind în acelaşi timp şi generală, cu un suport tehnic bine pus la punct. Astfel, instrumentul de proiectare PDT [16] asistă dezvoltatorul sistemului multiagent în toate etapele de analiză şi proiectare.

Etapele metodologiei Prometheus sunt: analiza (specificaţia sistemului), proiectarea arhitecturală şi proiectarea detaliată. Implementarea se poate realiza pe orice platformă agent. În particular, dezvoltatorii metodologiei Prometheus au utilizat JACK.

Etapa de analiză constă în determinarea specificaţiei sistemului. În cadrul acestei etape sunt definite scopuri, funcţionalităţi, scenarii şi interfeţe cu mediul de lucru sub formă de acţiuni şi percepţii.

Proiectarea arhitecturală utilizează elementele definite în cadrul etapei anterioare pentru a determina care sunt agenţii din sistem şi care sunt interacţiunile dintre agenţi. Se construiesc

următoarele diagrame: diagrama de cuplare a datelor, diagrama cunoştinţelor fiecărui agent (conţinând relaţiile cu agenţii cunoscuţi de respectivul agent), diagrama generală a sistemului, diagrame şi protocoale de interacţiune.

În cadrul proiectării detaliate se determină structura internă a fiecărui agent din sistem şi se specifică modul în care agenţii îşi îndeplinesc task-urile. Fiecare agent este descris în termeni de capabilităţi, evenimente interne, planuri şi structuri de date. Se construiesc diagrame generale ale agenţilor şi ale capabilităţilor. De asemenea, sunt furnizaţi descriptori de plan, date, evenimente şi capabilităţi şi specificaţiile procesului (prin rafinarea protocoalelor de interacţiune).

În figura 6 este prezentată schematic metodologia Prometheus.

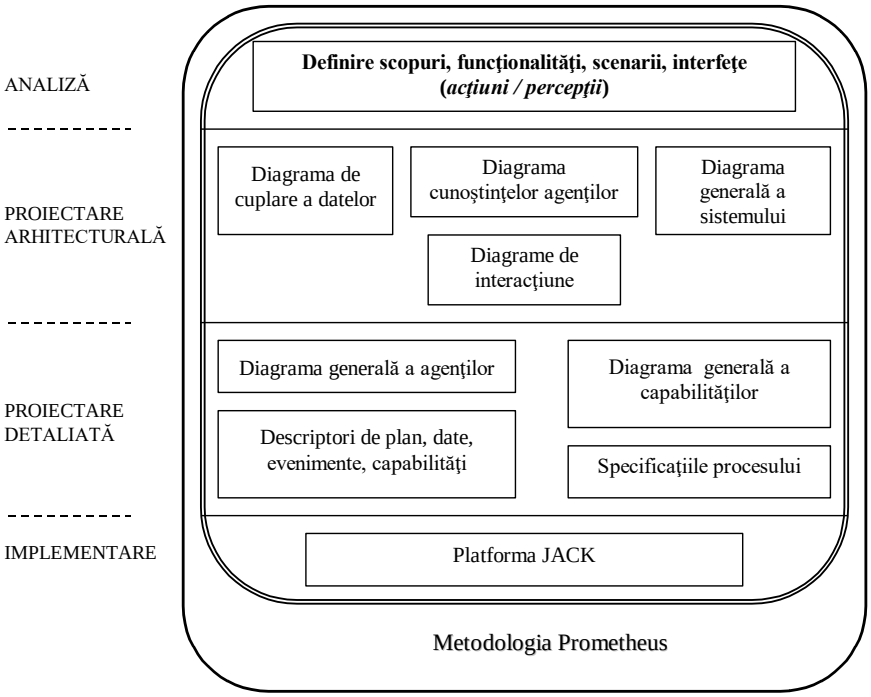

Figura 6. Metodologia Prometheus – reprezentare schematică

Metodologia Prometheus este similară cu metodologia Gaia în ceea ce priveşte realizarea etapelor de analiză târzie şi proiectare arhitecturală. Dezvoltatorii metodologiei şi-au propus includerea unei etape de analiză timpurie a cerinţelor şi utilizarea unor instrumente software performante pentru verificarea funcţionării sistemului bazat pe agenţi.

Principalele concepte utilizate în metodologia Prometheus sunt: percepţii, acţiuni, scopuri, funcţionalităţi (echivalentul rolurilor din alte metodologii), agent, plan, task, tipuri agent, capabilităţi, protocol.

4.5 ZEUS

Metodologia ZEUS [17], [18] este o metodologie specifică mediului ZEUS de dezvoltare a sistemelor multiagent [7]. În general, ZEUS este utilizat pentru dezvoltarea rapidă a unor sisteme prototip.

Etapele metodologiei sunt: analiza domeniului, proiectarea, implementarea şi testarea sistemului.

Scopul etapei de analiză a domeniului de aplicaţie este de a înţelege corect problema şi de a o modela corespunzător. Metodologia nu impune utilizarea unei anumite metode de analiză. Metoda recomandată este modelarea rolurilor [18], care este independentă de alegerea software-

ului pentru implementarea sistemului bazat pe agenţi. Problema este gândită în termeni de roluri pe care le poate avea un agent şi de responsabilităţi asociate fiecărui rol. Un model de rol este un şablon al celei mai simple soluţii pentru o anumită aplicaţie. Un rol descrie poziţia şi lista de responsabilităţi ale agentului în cadrul unui anumit context sau model de rol. Rolurile asociate agenţilor şi modelele de roluri furnizează un vocabular pentru descrierea sistemelor multiagent.

Prima activitate din etapa de proiectare este alocarea rolurilor identificate la etapa anterioară unor agenţi din sistemul multiagent. Dezvoltatorul sistemului transpune problema din termeni de roluri şi responsabilităţi în termeni de agenţi şi task-uri / servicii, modelând problema şi soluţia acesteia sub forma unui sistem multiagent. A doua activitate a etapei de proiectare a aplicaţiei constă în modelarea cunoaşterii declarative care va fi utilizată de rolurile agenţilor. Se vor identifica şi defini conceptele cheie ale domeniului aplicaţiei. Conceptele ZEUS sunt denumite fapte ZEUS şi sunt de două categorii: abstracte şi entitate, fiind organizate sub forma unei ierarhii de fapte. Fiecare concept are un nume şi o listă de atribute identificate prin nume, tip de date, restricţii şi valoare implicită. Conceptele cheie definite de către dezvoltatorul aplicaţiei vor forma ontologia sistemului multiagent, adică vocabularul de termeni cunoscuţi de agenţi (termeni care se vor regăsi în mesajele schimbate între agenţi, ale căror valori vor fi modificate eventual de task-uri sau în funcţie de valorile cărora se vor declanşa anumite task-uri).

Implementarea sistemului bazat pe agenţi presupune parcurgerea următoarelor sub-etape: creare ontologie, crearea fiecărui agent task (definire agent, descriere task-uri, organizare agent, coordonare agent), configurare agenţi utilitari, configurare agenţi task, implementare agenţi (scrierea codului sursă Java specific aplicaţiei). Toate aceste sub-etape sunt realizate cu ajutorul Generatorului de Agenţi, un instrument software încorporat în ZEUS. Dezvoltatorul sistemului bazat pe agenţi va implementa doar task-urile specifice aplicaţiei, resursele externe (de obicei, baze de date) şi programele externe (de exemplu, interfeţe utilizator). Exceptând programele care trebuie scrise manual, codul sursă este generat automat în ZEUS.

Testarea sistemului bazat pe agenţi este realizată cu ajutorul unui agent facilitator predefinit, Visualiser, care are rolul de depanare a sistemului, permiţând analiza mesajelor transmise, analiza evoluţiei agenţilor, analiza executării task-urilor şi a îndeplinirii scopurilor etc.

În figura 7 este prezentată schematic metodologia ZEUS.

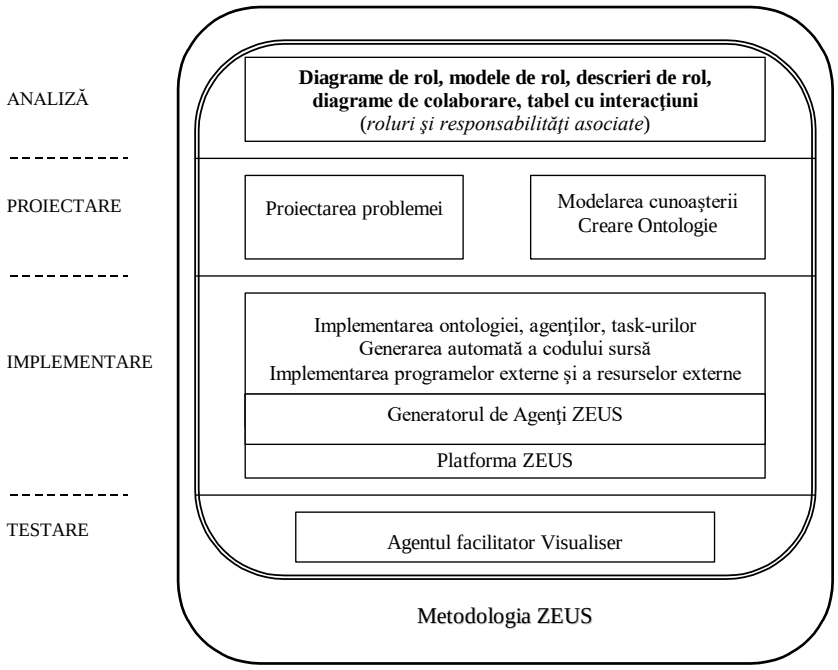

Figura 7. Metodologia ZEUS – reprezentare schematică

Dezavantajul major al acestei metodologii este dat de riscul modificării unor componente predefinite, în condiţiile în care codul nu este documentat.

Principalele concepte utilizate în metodologia ZEUS sunt: faptă, scop, agent, task.

5. Concluzii

Dezvoltarea sistemelor bazate pe agenţi inteligenţi presupune utilizarea unei metodologii specifice, orientate pe agenţi, mult mai potrivită decât o metodologie orientată pe obiecte. O astfel de metodologie utilizează concepte specifice agenţilor inteligenţi şi modelării multiagent. În literatura de specialitate au fost prezentate diferite analize comparative ale metodologiilor orientate pe agenţi (de exemplu, [1], [19], [20]), care au evidenţiat avantaje şi dezavantaje şi au formulat direcţii viitoare de dezvoltare a acestora.

Scopul studiului realizat şi prezentat în această lucrare a fost de a sintetiza şi analiza succint un număr de metodologii reprezentative, orientate pe agenţi, care s-au impus în ultimii ani pe plan mondial, Gaia, Tropos, MaSE, Prometheus şi ZEUS. Astfel, metodologiile selectate au fost descrise succint şi reprezentate schematic într-o formă unitară, structurată pe etapele modelului cascadă de dezvoltare a unui produs software. De asemenea, s-au identificat conceptele utilizate în cadrul fiecărei metodologii.

Din studiul prezentat în lucrare se desprind următoarele concluzii:

- nici una din metodologii nu este generală şi nici completă (nu acoperă toate etapele dezvoltării software-ului, pornind de la analiza timpurie a cerinţelor şi până la testarea şi întreţinerea acestuia);
- fiecare metodologie utilizează un număr de concepte a căror semnificaţie sau denumire poate diferi;
- toate metodologiile analizate utilizează modelul cascadă de dezvoltare a unui produs software.

Alegerea unei metodologii potrivite dezvoltării unei anumite aplicaţii depinde de domeniul aplicaţiei, de complexitatea acesteia şi de caracteristicile specifice ce pot fi furnizate de anumite metodologii orientate pe agenţi. Metodologiile Gaia (variantele îmbunătăţite, ROADMAP şi Gaia v.2), Tropos şi Prometheus sunt printre cele mai utilizate metodologii orientate pe agenţi.

BIBLIOGRAFIE

1. **BERGENTI, F., M.-P. GLEIZES, F. ZAMBONELLI**: Methodologies and Software Engineering for Agent Systems – The Agent-Oriented Software Engineering Handbook, Kluwer Academic Publishers, 2004.

2. **RAO, A. S., M. P. GEORGEFF**: Modelling rational agents within a BDI-architecture. În: Proc. of Knowledge Representation and Reasoning Conference KRR-91, San Mateo, CA, 1991.

3. **WEISS, G.**: Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence, The MIT Press, Cambridge, 1999.

4. **FIPA**: http://www.fipa.org, 1997.

5. **YU, E.**: Modelling Strategic Relationships for Process Reengineering, PhD Thesis, University of Toronto, 1995.

6. **AUML**: http://www.auml.org, 2005.

7. **ZEUS**: http://sourceforge.net/projects/zeusagent, 2005.

8. **AGENTBUILDER**: http://www.agentbuilder.com, 2006.

9. **WOOLDRIDGE, M., N. R. JENNINGS, D. KINNY**: The Gaia Methodology for Agent-Oriented Analysis and Design. În: International Journal of Autonomous Agents and MultiAgent Systems, 3(3), 2000, pp. 285-312.

10. **ZAMBONELLI, F., N. R. JENNINGS, M. WOOLDRIDGE.**: Developing Multiagent Systems: The Gaia methodology. În: ACM Transactions on Software Engineering and Methodology, 12(3), 2003, pp. 417-470.

11. **JUAN, T., A. PIERCE, L. STERLING**: ROADMAP: Extending the Gaia methodology for Complex Open Systems. În: Proc. of the 1st ACM Joint Conference on Autonomous Agents and Multi-Agent Systems, ACM Press, 2002.

12. **BRESCIANI, P., A. PERINI, P. GIORGINI, F. GIUNCHIGLIA, J. MYLOPOULOS**: Tropos: An Agent-Oriented Software Development Methodology. În: Autonomous Agents and Multi-Agent Systems, 8(3), 2004, pp. 203-236.

13. **UML**: http://www.omg.org/uml, 2001.

14. **JACK**: http://www.agent-software.com, 2005.

15. **DELOACH, S.**: Analysis and design using MaSE and agentTool. În: Proc. of the 12th Midwest Artificial Intelligence and Cognitive Science Conference MAICS-01, Miami University Press, 2001.

16. **PADGHAM, L., M. WINIKOFF**: Developing Intelligent Agent Systems: A Practical Guide, John Wiley and Sons Ltd, 2004.

17. **COLLIS, J., D. NDUMU**: The Application Realisation Guide – ZEUS Methodology Documentation Part III, Intelligent Systems Research Group, BT Labs, 1999.

18. **COLLIS, J., D. NDUMU**: The Role Modelling Guide – ZEUS Methodology Documentation Part I, Intelligent Systems Research Group, BT Labs, 1999.

19. **DAM, K. H., M. WINIKOFF**: Comparing Agent-Oriented Methodologies. În: P. Giorgini et al. (Eds), AOIS-03, LNAI 3030, Springer-Verlag, 2004, pp. 78-93.

20. **OPREA, M.**: Agent-Oriented Software Engineering. În: Proc. of the 24th IASTED International Multi-Conference Software Engineering SE-06, Innsbruck, Austria, 2006, pp. 1-6.