ANALIZA COMPARATIVĂ A LIMBAJELOR DE TIP AOPL

Vasili BRAGA

Universitatea Tehnică a Moldovei

Abstract: *În ultimii ani tehnologia agenților a devenit foarte importantă în mai multe aspecte ale tehnologiei informaționale. Sistemele de agenți și multi-agenți sunt folosite într-o mulțime de aplicații și în mai multe domenii de activitate. Se crede ca tehnologia agenților va deveni din ce în ce mai puternica și utila în anii următori. În lucrarea dată este prezentată ideea limbajelor specializate, care ar putea suporta nativ proprietățile și mediul de dezvoltare a agenților și sistemelor multi-agent și care poartă denumirea de AOP (Programarea orientată pe Agenți), iar un limbaj de tipul dat poartă denumirea de AOPL (Agent Oriented Programming Language). În lucrarea se evidențiază necesitățile, scopul, atributele și istoria acestor tipuri de limbaje de programare.*

Cuvinte cheie: *agenți, sistemele multi-agenți, AOP (Programarea orientată pe Agenți), AOPL (Agent Oriented Programming Language).*

1. Introducere

Un agent poate fi definit ca un program care are careva scopuri, pe care încearcă sa le atingă, careva capabilități pe care le poseda, unele cunoştințe pe care le are și este autonom.

Agenții pot fi considerați următorul pas în progresul gradual al dezvoltării softului. Primul pas a fost programarea imperativă, unde o serie de instrucțiuni se execută pas cu pas, una după alta. După a apărut idea programări procedurale, acest concept a fost rapid urmat de conceptul modularității și cu ideea de abstractizare a informației cât și a ascunderii informației, ceea ce a servit ca baza a OOP (Object Oriented Programming). În OOP un obiect reprezintă încapsularea și abstracția unor date cu careva metode, care poate opera cu informația data și starea lui poate fi modificată doar prin una din metodele sale.

Progresul nu s-a oprit la acest nivel și în următorii ani companii ca Sun, Borland/Imprise și Microsoft încearcă sa lucreze în jurul unui concept a unui limbaj orientat pe obiecte pe baza de componente.

Programarea bazata pe agenți poate fi considerata o extensie logica a etapelor precedente și poate fi văzuta ca o forma specializata a unui obiect, unde nimeni din afara nu poate să schimbe starea lui.

2. Limbaj de programare orientat pe agenți

Până în prezent sistemele multi-agent se proiectau în baza limbajelor OO (Object Oriented) cum ar fi C/C++ și Java, sau limbajelor de tip AI (Artificial Intelligence) ca Lisp sau Prolog. Abordarea OO fiind mai populară la moment, unde idea de baza constă în definirea agentului și adăugarea lui mai multor proprietăți. Există o mare varietate de instrumente atât în industrie cât și în mediul academic, care pot fi folosite pentru dezvoltarea sistemelor de agenți, sau sistemelor multi-agent.

Totuși, dacă se dorește de prezentat evoluția agenților cât și a sistemelor multi-agent, ar fi bine să fie făcute câteva ipoteze pentru definirea unui limbaj de programare orientat pe agenți. Deși există câteva limbaje, totuși se simte o lipsa de limbaje care ar putea fi marcate în felul dat ca un adevărat limbaj de tip AOP.

AOPL ipotetic va trebui să ofere suport pentru o gamă largă de caracteristicile agentului "cheie". Mai jos este prezentată o listă (fără a le da un clasament) de ceea ce se consideră a fi principalele caracteristici pe care AOPL ipotetic va trebui să le sprijine.

Autonomie pentru agenți: Autonomia este una dintre caracteristicile distinctive ale unui agent. Aproape toți oamenii care lucrează în domeniul tehnologiei de agent sunt de acord asupra acestui atribut. De asemenea, este acea caracteristică care distinge un agent de la un obiect în modelul OOP. Prin autonomie se referă la faptul că un agent este capabil să acționeze fără intervenția oamenilor și a altor sisteme și are controlul asupra stărilor sale interne cât și asupra comportamentului, și este condus de un set de tendințe. Aceste tendințe sunt fie scopuri individuale pe care un agent încearcă să le atingă sau funcții de supraviețuire, pe care un agent încearcă să le optimizeze. Orice limbaj OO garantează încapsularea și ascunderea informației pentru orice obiect. În mod similar, orice limbaj AO (Agent Oriented), ar trebui să garanteze autonomia agentului. Prin urmare, se crede că orice limbaj nu poate fi numit ca un adevărat AOPL, până și cu excepția cazului în care asigură autonomia agenților programați în el.

Comunicarea: Este posibil ca un agent să funcționeze în mod util singur, fără nici un fel de interacțiune cu alți agenți, dar cantitatea tot mai mare de rețele și natura distribuită atât a datelor cât și aplicațiilor face aceste situații destul de rare.

Scenariul final este un sistem multi-agent, format din mai mult de un agent, ce coordonează, cooperează, sau chiar concurează cu ceilalți agenți. În scopul de a coordona, coopera sau concura între ei, agenți trebuie să comunice unul cu altul. Aceste entități pot fi utilizatori umani sau sisteme software. Aceasta comunicare poate fi directă sau indirectă, în funcție de tipul de abordare și de arhitectură.

Un AOPL ar trebui sa permită programatorului agentului să dezvolte comunicarea între agenți fără nici o dificultate. Acesta ar trebui să ofere un nivel mare de abstracție și mijloace pentru a pune în aplicare diferite tipuri de comunicare. În mod normal pentru ca agenții să comunice între ei este folosit un limbaj de comunicare între agenți (ACL).

Există mai multe standarde și definiții ale ACL disponibile. Orice AOPL nu ar trebui să fie legat de o anumită ACL, dar ar trebui să asigure programatorului libertatea de selectare a oricărui ACL la alegerea liberă potrivit nevoilor sale.

Distribuție: Așa cum s-a descris mai sus, în mod normal, agenții activează în medii formând sisteme de tip multi-agent. Un sistem tipic multi-agent este format din mai mult de un agent și probabil este distribuit în întreaga lume. Orice AOPL ideal ar trebui să sprijine această distribuție a agenților în întreaga lume.

Simultaneitatea: sistemele multi-agent sunt adesea distribuite în natura lor. Concurența este una dintre proprietățile importante ale sistemelor distribuite. Este dificil de utilizat un limbaj de programare pentru dezvoltare și de punere în aplicare a unui MAS (Multi-Agent System), în cazul în care limbajul dat nu prevede concurența.

Atribuirea de nume: În sistemele multi-agent fiecare agent trebuie să aibă un nume unic, prin care poate fi identificat în mod unic. Acest nume unic este foarte important din mai multe motive. Mai ales, în cazul comunicării între agenți, este nevoie de un nume unic pentru destinatar și expeditorul de mesaje. De asemenea, ori de câte ori o sarcina este distribuita între agenți, este nevoie de un nume unic pentru fiecare în parte, în scopul de a păstra evidența cine și ce sarcină realizează. Așa că, orice sistem distribuit, sistemele multi-agent au nevoie de un mecanism prin care să aloce fiecărui agent un nume unic. În mod ideal, un AOPL ar trebui să ofere un așa mecanism, astfel încât fiecare agent să poată fi numit în mod unic.

Mobilitatea agentului: Un agent mobil este un tip specific de agent, care poate migra de la o mașină la alta într-o rețea eterogenă. Cu toate că nu orice agent are nevoie de a fi mobil, există unele domenii de aplicare în cazul în care aceasta este foarte util și câteodată folosirea este inevitabilă.

Reprezentarea atributelor mentale: În baza a mai multor cercetători, cum ar fi Shoham, agenții au o stare mentală [29]. Componentele unei astfel de stări mentale sunt adesea considerate că ar fi credința (Believe), obligația și capabilitatea.

Luarea deciziilor: Nu contează dacă considerăm un agent are stare mentală sau nu, el încă trebuie să ia decizii. Acest lucru, vine odată cu autonomia. Atunci când spunem că un agent este capabil să acționeze fără intervenția omului și a altor sisteme atunci avem în vedere ca el are în el capabilitatea de luare a deciziilor, astfel încât să poată decide ce să facă și când să facă.

Reprezentarea cunoștințelor: Un agent poate învăța prin diferite mijloace. Poate să învețe din experiență, el poate învăța de la alți agenți sau chiar se poate învăța prin percepție din mediul său. Informațiile pe care un agent le învață trebuie să fie reprezentate într-un fel sau altul. Este nevoie de un fel de mecanism de reprezentare a cunoștințelor, pe care un AOPL ar trebui să îl ofere.

Un succesor OO: Așa cum a fost discutat anterior orientarea pe agent poate fi considerată ca dezvoltarea progresivă a orientării pe obiecte. De asemenea, mulți dintre agenții actuali sunt bazați, dezvoltați și implementați în limbaje OO cum ar fi Java si C ++. Prin urmare, nu ar fi nerezonabil să se presupună că un AOPL, care este similar în design - ul sau unui OOL (Object Oriented Language) va avea mai multe șanse de acceptare, spre deosebire de orice alt model.

3. Unele AO Modele existente

Există mai multe limbaje dezvoltate, în special în mediul academic, pentru dezvoltarea sistemelor bazate pe agenți, dar mai jos sunt prezentate doar cinci dintre ele.

3.1 Agent-0

Termenul AOP a fost folosit pentru prima dată de Yaov Shoham în 1989. El a prezentat o nouă paradigmă de programare, care este un framework de programare care se atrage de la AI, Teoria Actului vorbirii și OOP [28,29,30]. Potrivit Shoham, un sistem AOP este format din trei componente principale: un limbaj formal

pentru a descrie starea mentală a agenților, un limbaj interpretat de programare pentru a defini agenții și un agentifier pentru a converti dispozitive neutre în agenți programabili.

Pe baza teoriei sale a fost propus un limbaj Agent-0. Agent-0 urmează o buclă de control simplă, atunci când se execută un program, care, la fiecare pas are următoarele sarcini:

1) aduna mesajele primite și actualizeze starea mentală în concordanță cu ele;

2) execute angajamente (folosind capabilități).

Un program Agent-0 este format din normele unui angajament, normele de executare a capabilităților și un set de convingeri inițiale. În Agent-0, angajamentele pot fi făcute numai pentru a executa acțiuni primitive, orice activitate care ar necesita o planificare trebuie să fie angajată anticipat după planul său.

Agenții pot efectua două clase de acțiuni: private și de comunicare. Acțiunile Private sunt gestionate de către o entitate separată (alta decât interpretatorul AOP) atunci când sunt întâlnite. Nici un mecanism nu descrie în Agent-0 în ce mod vor fi executate acțiunile private ci doar faptul executării.

Acțiuni comunicative utilizează comenzi de tip act-vorbire pentru a conversa cu alți agenți. Cele patru clase de mesaje Agent-0 sunt INFORM, REQUEST, UNREQUEST, și REFRAIN. O acțiune de tip INFORM trimite un fapt unui agent receptor, o acțiune de tip REQUEST notifică agentului receptor, că alt agent solicită o acțiune, care urmează să fie realizată. O UNREQUEST este inversa unei REQUEST. Un REFRAIN solicită ca o acțiune să nu fie angajata de către agentul de primire.

Agentul-0 a fost primul efort în direcția unei AOPL și are multe deficiențe. Nu definește nici un mecanism în care acțiunea privată va fi efectuată, de asemenea, actele de vorbire susținute în acțiunea de comunicare sunt foarte puține. Nu este abordată problema transmiterii mesajelor de la un agent la altul, ci pur și simplu presupune că este responsabilitatea platformei de bază pentru a transmite mesajul de la un agent la altul.

Mai mult, în Agent-0 nici un agent nu s-ar angaja vreodată la nimic dacă nu i s-ar fi cerut de către un alt agent. Cu toate că Agent-0 oferă un suport pentru așa caracteristici cum ar fi atributele mentale, dar are lacune în furnizarea altor caracteristici. Așa cum ar fi că din cauza deficiențelor menționate mai sus și lacune în furnizarea unor caracteristici cheie Agent-0 nu poate fi utilizat pentru dezvoltarea agenților în mediul eterogen curent și nu poate fi numit un adevărat AOPL în sensul modern al cuvântului.

3.2. Extensia PLACA Agent-0

PLACA (PLAning Communication Agents) a fost propus de Rebecca Thomas ca o extensie a Agent-0 [31]. În Agent-0, una dintre cele mai importante limitări poate fi numita Plan-directedness, adică fără niciun angajament la nimic, pot fi realizate doar acțiunile primitive, deoarece agentul nu are nici un mecanism pentru dezvoltarea de scopuri. PLACA încearcă să facă agenți care pot face planuri. Cu toate că PLACA este o formă îmbunătățită de Agent-0, dar fiind un descendent al lui Agent-0, moștenește toate deficiențele lui. De aceea, din cauza aceluiași motiv, PLACA nu se încadrează în criteriile de AOPL.

3.3. Extinderea Agent-0 Agent-K

Agent-K este un efort de a introduce unele standarde în transmiterea de mesaje în funcționalitatea Agent-0 [6]. Pentru a asigura că mesajul scris în alt limbaj poate fi manipulat, agent-K integrează sintaxa lui Agent-0 cu formatul KQML. Două schimbări majore în Agent-0 sunt introduse de către agentul-K: mesajele trimise INFORM, REQUEST și UNREQUEST sunt înlocuite cu o singură comandă. Mesajele primite, cu toate acestea, sunt transformate în bază tipurilor Agent-0 - cele trei tipuri de mesaje sunt suficient de expresive pentru a încapsula celelalte tipuri de mesaje KQML aproape complet.

Agent-K permite mai multe angajamente legate de un singur mesaj. În Agent-0, acest lucru nu a fost definit, iar interpretul pur și simplu selecta prima regulă care se potrivește acestui mesaj.

Agent-K sporește capacitățile de comunicare ale agenților dar încă moștenește toate celelalte deficiențe ale Agnet-0. Deci, din nou, nu poate fi descris ca un AOPL ideal.

3.4. Mozart

Limbajul Mozart, dezvoltat de consorțiul Mozart, este o platformă de dezvoltare cu un scop general [23]. Acesta a fost proiectat special pentru a sprijini concurența, distribuție și calculul simbolic. El implementează Oz, un limbaj concurent, funcțional orientat pe obiecte, care are la baza un model simplu bazat pe constrângeri simultane extinse cu stări de ordine mai înaltă. Oz combină programarea concurentă și distribuită cu logică constrângerii bazată pe deducție. Această caracteristică îl face o alegere buna pentru dezvoltarea sistemelor multi-agent.

Mozart acceptă unele dintre funcțiile cum ar fi concurența, distribuția și o anumită formă de sprijin pentru reprezentarea cunoștințelor și a luării deciziilor. Dar aceasta nu oferă nici un sprijin pentru alte caracteristici cheie, cum ar fi autonomia și mobilitatea. De asemenea, se bazează pe o altă limbă, care este, în principiu

orientata pe obiect. Aplicații bazate pe agenți pot fi dezvoltate folosind acest limbaj, dar acest lucru poate fi făcut în multe alte limbaje (C / C ++, Java, Prolog, etc.), și acest lucru nu asigură un Agent Oriented Language.

3.5. APRIL

Un alt limbaj numit APRIL (Agent Process Interaction Language) a fost dezvoltat în Imperial College din Londra [2,20]. Multe dintre caracteristicile în APRIL sunt luate din alte limbaje, în special Parlog, Erlang, PCN, CSP, Dijkstra's guarded commands, Lisp, Prolog și APL. Se pretinde a fi un limbaj simbolic distribuit. Ideea cheie din spatele APRIL este calcul simbolic. Blocuri de construcție în programele din APRIL sunt procese. Toate procesele au identificatoare unice la nivel global. Acestea oferă anumita structurare a datelor și posibilități de manipulare a expresiilor în ordine pentru a permite comunicarea între ele. Caracteristicile de ordin superior ale APRIL sunt utilizate pentru structurarea programului și de asemenea, pentru a oferi o facilitate pentru migrarea codului de la un proces la altul.

Deși APRIL oferă suport pentru unele caracteristici cum ar fi comunicarea, concurența și într -o oarecare măsură, mobilitatea, nu se poate spune ca este un adevărat limbaj orientat pe agent și nu oferă suport pentru alte caracteristici. El ar putea fi, probabil, cel mai bine considerat un limbaj orientat spre procese sau un limbaj concurent orientat spre obiecte în cazul în care obiectele active sunt procese.

4. Concluzii

AOP este o posibilă nouă paradigmă, este diferită și extinde existența metodologii de dezvoltare software. Nici un limbaj existent nu poate fi numit un AOPL în sens complet și am susține că există un domeniu de aplicare și o nevoie pentru un astfel de limbaj.

În lucrarea a fost detaliată lista de atribute, pe care considerăm că ar trebui să le posede un AOPL ideal. Totuși, în același timp, se consideră că, din cauzele discutate mai sus, există foarte puține șanse de acceptare globală a oricărui nou AOPL. Munca făcută de diferite organizații precum FIPA, sperăm, va fi capabilă pentru introducerea unor noi standarde de agenți și AOP. Odată ce există un standard convenit atunci introducerea unui nou AOPL ar putea fi acceptată și pe plan global.

Una dintre dezvoltările posibile este că vom avea în final mai multe limbaj de programare, fiecare pentru un anumit tip de agent, de exemplu un limbaj pentru agent mobil, altul pentru agent de interfață și așa mai departe. Acest lucru nu ar trebui să pară ciudat, dacă de uitat la istoria dezvoltării limbajelor de programare "convenționale", se observă că au fost dezvoltate limbaje de programare diferite pentru diferite tipuri de domenii (cum ar fi COBOL pentru afaceri și Fortran în scopuri științifice). Doar ulterior acestea au fost combinate în limbaje mai generale. În prezent este greu de găsit o lucrare în această direcție în termeni de limbaje de programare. Dar această tendință există deja și sunt create unele dintre instrumentele agent builder, cum ar fi IBM Aglet [1], care este un instrument pentru dezvoltarea agenților mobili, în timp JADE suportă dezvoltarea agenților în conformitate cu arhitectura FIPA.

O altă abordare dezirabilă ar fi dezvoltarea bibliotecilor independente de limbajele de programare sau cel puțin o descriere a interfeței pentru o serie de biblioteci standard. În prezent, noi avem diferite instrumente disponibile pentru dezvoltarea agenților, dar aproape toate aceste instrumente sunt pe baza unor limbaje specifice. O abordare mai sofisticată poate fi bibliotecile de clasă, independente de limbaj pentru dezvoltarea pe sisteme bazate pe agenți, similar cu standardul bibliotecii POSIX pentru sistemele de operare. Acest lucru ar putea introduce un fel de standardizare în metodologiile de dezvoltare a agenților, care poate fi prelucrat și extins într-o bază adecvată și largă AOPL mai târziu.

În ultima vreme, Oracle revizuiește o specificație, JSR00087 [16], pentru agentul Java Services. Proiectul Service Java Agent [14] este o inițiativă de definire a specificației de standard a unei industrii și API pentru dezvoltarea agenților și a arhitecturii serviciilor de rețea. Se știe că Java este deja favorit pentru o clasă mare de programatori ce lucrează în domeniul tehnologiei agenților. Introducerea noilor servicii Java Agent va fi mai mult decât bine primit de o clasă. Se crede că acest lucru ar putea duce la nașterea unui nou AOPL.

Bibliografie

1. Aglets Software development kit, 2001, http://www.trl.ibm.com/aglets/.
2. April Agent Platform, 2001, http://www.nar.fla.com/aap/index.html.
3. Bradshaw, J. *Software Agents* AAAI press/The MIT press, 1995.
4. Caglayan, A., Harrison, C. *Agent Source Book*, John Willey & Sons, 1997.
5. Coulouris, G., Dollimore, J., Kindberg, T. *Distributed Systems*, Addison Wesley, 1994.
6. Davies, W. Edwards, P. *Agent-K: An Integration of AOP and KQML*, King's College Technical Report AUCS/TR9406,1994.
7. Ferber, J. *Multi-Agent System. An Introduction to Distributed Artificial Intelligence*. Addison Wesley, 1999.

8. FIPA, Agent Communication Language, 2001, http://www.fipa.org/repository/aclspecs.html.
9. Gensereth, M. Fikes, R. *Knowledge Interchange Format*, Version 3.0 Reference Manual,1992.
10. Gerhard, W. *Multiagent Systems A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.
11. IBM Intelligent Agent Resource Manager, Open Blueprint G325-6592-0,1996.
12. IBM alphaWorks, 2001, http://alphaworks.ibm.com.
13. JADE, 2001, http://sharon.cselt.it/projects/jade/.
14. Java Agent Services Project, 2001, http://www.java-agent.org/index.html.
15. Jennings, N., Wooldridge, M. *Agent Technology*, Springer, 1997.
16. JSR 00087 Sun Micro System, 2001, http://java.sun.com/aboutJava/communityprocess/jsr/jsr_087_jas.html.
17. Labrou, Y., Finin, T. *Semantics for an agent communication language*. The fourth International Workshop on Agent Theories, Architectures, and Languages. Rhode Island, USA,1997.
18. Lander, S. *Issues in Multiagent design systems*, 1997, IEEE Expert12 (2): 18-26.
19. Maes, P. *Designing Autonomous Agents*. Cambridge, MA:MIT Press, 1991.
20. McCabe, F., Clark, L. *April -Agent PRrocess Interaction Language, Intelligent Agents*. ECAI-94 Workshop on Agent Theories, Architectures, and Languages Proceedings, 1995 pp. 324-40.
21. Meyer, J. *Agent languages and Their Relationship to Other Paradigms*, 5th International Workshop ATAL-98 Proceedings, 1999, pp. 309-16
22. Ming, T., Weihmayer, R. *Integrating Agent Oriented Programming and Planning For Cooperative Problem Solving*, Notes From AAAI-92 Workshop on Cooperation Among Heterogeneous Intelligent Systems, 1992.
23. Mozart Programming System, 2001, www.mozart-oz.org.
24. Muller, P. *The Design of Intelligent Agents - A Layered Approach*, volume 1177 of LNAI, 1996.
25. Open Agent Architecture, 2001 http://www.ai.sri.com/~oaa/whitepaper.html.
26. Reilly, D. *Simple Agent Communication Protocol*, 2000, http://www.davidreilly.com/sacp.
27. Sadek, D., Bretier, P., Panaget, F. *ARTIMIS technology*, 1997, http://www.cselt.it/fipa/torino/cfp1/propos97_015.htm.
28. Shoham, Y. *AGENT0: a simple agent language and its interpreter*. In Proceedings of AAAI Anaheium, 1991.
29. Shoham, Y. *Agent-oriented programming*, Artificial Intelligence, 60(1),1993.
30. Shoham, Y. *Agent Oriented Programming: an overview of the framework and a summary of recent research, Knowledge Representation and Reasoning Under Uncertainty*, 1994, pp. 123-9.
31. Thomas, R. The PLACA Agent Programming Language, Intelligent Agents. ECAI-94 Workshop on Agent Theories, Architectures, and Languages Proceedings, 1995, pp. 355-70.