Metodologii de dezvoltare a sistemelor multi-agent

Manaţe Bogdan Ioan

March 30, 2011

Abstract

Următoarea metodologie este propusă pentru dezvoltarea sistemelor multi-agent folosind platforma JADE. Metodologia propusă se axează pe problemele cheie în analiza şi proiectarea sistemelor multi-agent. Partea de analiză este generică, iar partea de design se bazează pe exemple concrete bazate pe platforma JADE. Metodologia este ilustrată prin aplicarea ei asupra unor scenarii ipotetice, astfel încât programatorii familiarizaţi cu platforma JADE pot aplica, analog, această metodologie pentru a reduce semnificativ timpul de dezvoltare al unui nou sistem.

1 Introducere

În dezvoltarea software există patru faze fundamentale: planificare, analiza, design şi implementare. În cadrul etapei de implementare este util să se folosească o listă de paşi. Această listă de paşi reprezintă metodologia de dezvoltare. Folosirea unei metodologii reduce din timpul şi efortul necesar, prin cristalizarea paşilor importanţi pe care dezvoltatorul trebuie să-i urmărească.

Ingineria produselor software bazate pe agenţi, este un câmp relativ nou care poate fi gândit ca o evoluţie a programării orientate-obiect. Cu toate că tehnologia orientată pe agenţi oferă metode eficiente de rezolvare a anumitor probleme, în momentul de faţă nu există prea multe metodologii de dezvoltare. Metodologia propusă nu încearcă să extindă tehnicile orientate obiect, în schimb este orientată pe agenţi şi pe abstractizarea oferită de paradigma orientată pe agenţi. Formalizarea fazelor de planificare şi implementare din ciclul de viaţă software nu vor fi prezentate în cadrul acestei metodologii.

Faza de design este orientată pe platforma JADE şi asupra conceptelor oferite de aceasta. JADE este un cadru de programare scris în limbajul de programare JAVA, care simplifică implementarea sistemelor multi-agent şi care oferă un set de unelte pentru fazele de depanare şi lansare. Prin concentrarea specifică asupra platformei JADE în faza de design, dezvoltatorul poate trece peste partea de analiză direct la faza de implementare. Acest lucru va reduce semnificativ timpul necesar dezvoltării aplicaţiei.

2 Prezentare generală a metodologiei

În general o metodologie de dezvoltare software poate fi compusă din:

- Un proces, adică o secvenţă de paşi sau faze care ghidează dezvoltatorul în construirea sistemului.
- Un set de reguli euristice, care ajută dezvoltatorul să facă alegeri relevante.
- Diagrame, scheme sau documente reprezentând unul sau mai multe modele ale sistemului.
- O notare potrivită.
- Un set de şabloane care pot fi aplicate pentru rezolvarea situaţiilor comune.
- Unu sau mai multe unelte care automatizează cât mai mult posibil fazele şi paşii specificaţi în proces.

Observând figura 1 se observă că nu există o graniţă strictă între faza de analiză şi faza de design. În plus metodologia este de natură iterativă, permiţându-i dezvoltatorului să se mute între fazele de analiză şi design. La sfârşitul fazei de design dezvoltatorul poate să treacă direct la faza de implementare, partea în care intervine scrierea de cod.

În cadrul acestei metodologii se vor face următoarele presupuneri:

- Platforma JADE este aleasă pentru implementare.
- Se vor folosi un număr relativ redus de agenţi.
- Structura organizaţională este statică, nu se asteaptă un comportament non-emergent în timpul rulării.
- Securitatea nu este importantă.

3 Cum se defineşte un agent ?

Un agent poate fi văzut ca o aplicaţie software care rezidă pe o platformă, care îi oferă agentului posibilitatea de a comunica în funcţie de complexitatea mediului în care se află.

Agenţii pot avea următoarele seturi de proprietăţi:

- Autonomie - agenţii pot opera fără intervenţia utilizatorului, având control asupra acţiunilor lor şi stării interne.
- Abilităţi sociale - agenţii pot interacţiona cu alţi agenţi sau cu utilizatorii folosind un limbaj de comunicare.
- Reactivitate - agenţii percep mediul în care se află şi răspund adecvat schimbărilor survenite.
- Proactivitate - agenţii pot lua iniţiativă, prezentând comportamente orientate pe îndeplinirea unui anumit scop.

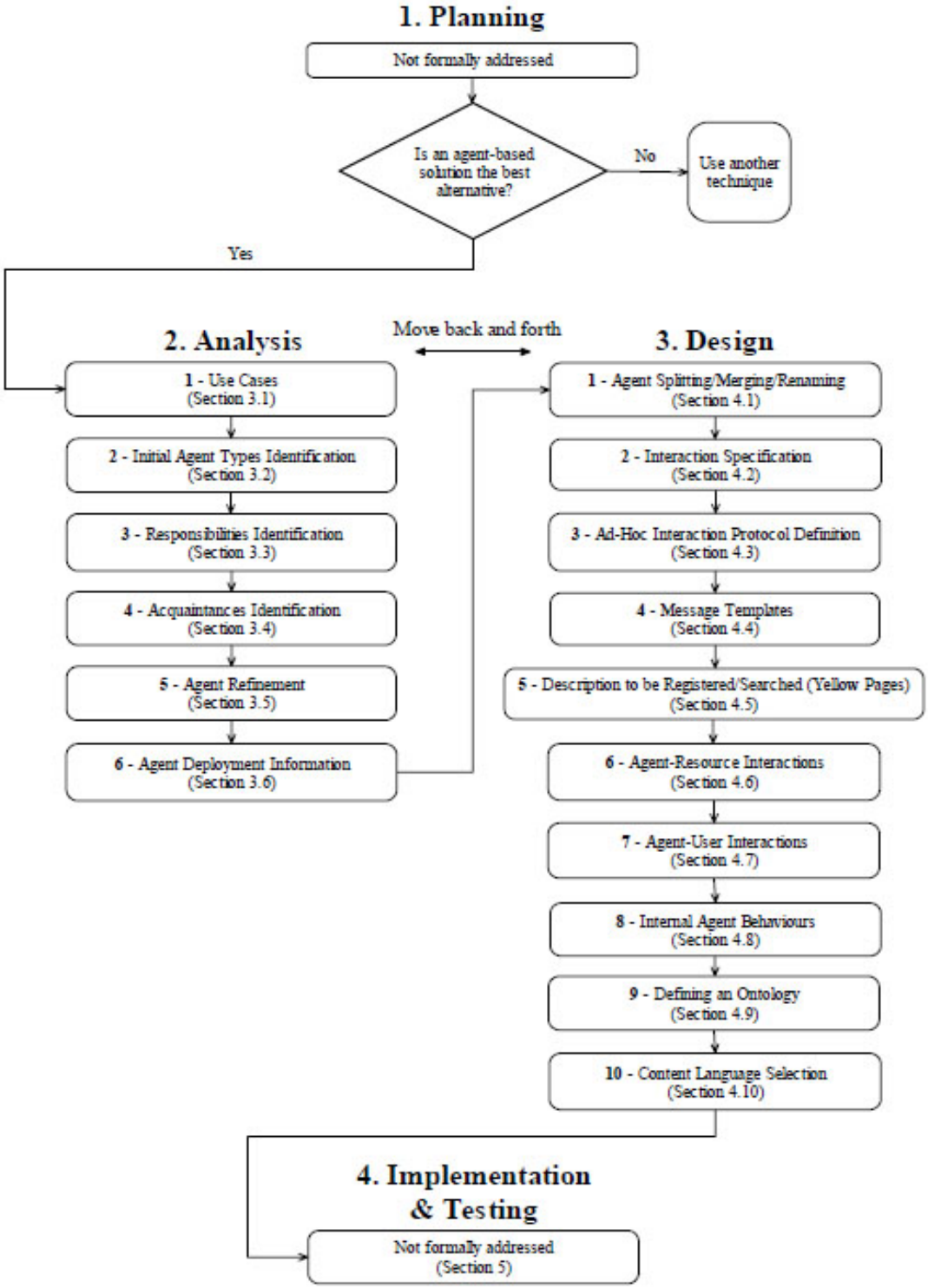

Figure 1: Prezenare generală a metodologiei

4 Scenariu

Scenariul asupra căruia se va aplica această metodologie presupune crearea unuei aplicaţii care să permită organizarea ieşirilor la cinema. Această metodologie va fi de asemenea aplicată analog şi în dezvoltarea middleware-ului. Pentru scenariul mentionat se presupune că un operator de telefonie mobilă, doreşte să le ofere abonaţilor săi posiblitatea să organizeze evenimente la cinema. Serviciul ar trebui să le permită abonaţilor să-şi invite prietenii să vadă un film, să colecteze preferinţele fiecăruia şi să ofere cea mai bună opţiune pe baza preferinţelor. Sistemul va oferii informaţii doar despre cinematografele locale în funcţie de poziţia utilizatorului.

5 Analiza

În partea de analiză se clarifică toate aspectele legate de soluţie. În cadrul acestei metodologii, faza de analiză este împărţită în şase paşi, definiţi in secţiunile următoare.

5.1 Cazuri de utilizare

Cazurile de utilizare reprezintă o metodă efectivă de capturare a potenţialelor cerinţe funcţionale ale unui nou sistem. Fiecare caz de utilizare prezintă unul sau mai multe scenarii care demonstrează cum ar trebui sistemul să reacţioneze când interacţionează cu utilizatorul sau alt sistem.

Pe baza descrierii aplicaţiei şi dupa chestionarea potenţialilor utilizatori este posibil să se construiască o listă preliminară cu scenarile posibile.

5.2 Identificarea tipurilor de agenţi

Acest pas presupune identificarea tipurilor iniţiale de agenţi şi construirea unei diagrame de agenţi. În această fază se pot aplica următoarele reguli:

- Adăugarea unui tip de agent pentru fiecare utilizator sau dispozitiv.
- Adăugarea unui tip de agent pentru fiecare resursă.

Diagrama agenţor conţine patru tipuri de elemente:

- Tipurile de agenţi - reprezentate prin cerc.
- Utilizatorii - oamenii care interacţionează cu sistemul.
- Resursele - sisteme externe care interacţionează cu sistemul.
- Cunoştiinţele - reprezentate prin săgeţi care unesc instanţele elementelor de mai sus, specificând ca elementele legate vor trebui să interacţioneze într-un anumit fel.

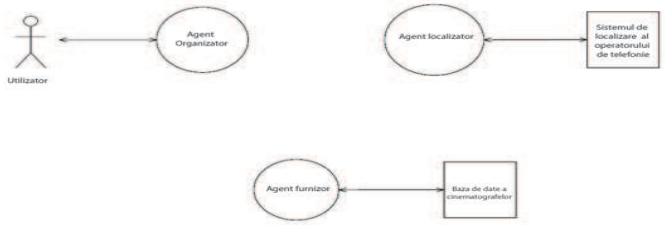

Figure 2: Diagrama agenţilor

În diagramele agenţilor, spre deosebire de diagramele de utilizare UML, se face distincţie între utilizatorii umani şi sistemele externe. Înteracţiunea cu utilizatorii printr-o interfaţă grafică ridică probleme adiţionale faţă de interacţiunea cu alte sisteme. Modul în care sistemele externe/sistemele vechi şi utilizatorii interacţionează prezintă o importantă consideraţie. Pentru a realiza interacţiunea dintre sisteme, respectiv utilizatori şi agenţi se poate folosi una dintre tehnicile de mai jos [10]:

- Folosirea unui agent traducător. Agentul traducător serveşte ca interfaţă între sisteme şi alţi agenţi din sistem. Agentul traducător primeşte mesaje de la agenţii din sistem pe care le traduce în limbaj nativ al sistemului pe care il accesează. Similar se întamplă pentru cerinţele sistemelor, agentul traducător traduce cerinţele în mesaje şi le trimite celorlalţi agenţi din sistem.

- Introducerea unui adaptor. Această metodă presupune injectarea de cod nou în sistem. Codul inserat va permite sistemului să comunice folosind ACL (agent communication language).

- Rescrierea codului. Această metodă este cea mai extremă şi presupune scrierea de cod care să mimeze sau chiar să extindă operaţiunile şi capabilităţiile sistemului existent, adăugânduse şi cod care să permită comunicarea folosind ACL. Această metodă se alege doar când celălalte metode nu sunt viabile.

5.3 Identificarea responsabilităţilor

În cadrul acestei etape, pentru fiecare agent identificat se crează o listă care conţine principalele responsabilităţi scrise într-un mod formal şi intuitiv. În această etapă se pot aplica următoarele reguli:

- Găsirea seturilor principale de responsabilităţi se realizează utilizând cazurile de utilizare.

- Se vor considera doar agenţii pentru care responsabilităţile sunt clare, identificare responsabilităţilor pentru ceilalţi agenţi se va face în fazele următoare.

Unele metodologii existente, cum ar fi GAIA[11] sau MESSAGE[4] propun o abordare diferită, care presupune găsirea rolurilor atomice, care apoi vor fi transformate în tipuri de agenţi. Aceste abordări sunt considerate mai puţin intuitive, deoarece în anumite cazuri este dificil să se determine cum se vor transforma rolurile atomice în tipuri de agenţi. Definirea tipurilor de agenţi şi apoi a responsabilităţilor înlătură această ambiguitate.

5.4 Identificarea cunoştiinţelor

În această fază se vor identifica entităţile care interacţionează, iar diagrama agenţilor va fi actualizată adăugându-se noi relaţii pentru agenţii care trebuie sa interacţioneze. O relaţie evidentă în acest caz este între agenţii organizatori şi agenţii învitaţi. Deoarece un agent organizator trebuie să prezinte programul cinematografelor utilizatorului, iar programul cinematografelor este stocat în baza de date a cinematografelor, care este administrată de un alt agent, rezultă că între cei doi agenţi ar trebui să existe o legătură.

5.5 Rafinarea tipurilor de agenţi

În faza de rafinare a agenţilor se au în vedere următoarele:

- Suportul: de ce informaţii au nevoie agenţii pentru a-şi îndeplini responsabilităţile, şi cum, unde şi când această informaţie este stocată.
- Descoperirea: cum agenţii legaţi direct de aceeaşi cunoştiinţă se pot descoperi.
- Management şi monitorizare: sistemul trebuie să fie capabil să monitorizeze agenţii existenţi şi să-i poată oprii sau pornii la comandă.

6 Design

O dată ce problema a fost clarificată pentru un nivel suficient de detalii, se trece de la faza de analiză la faza de design a cărei ţintă este specificarea soluţiei. Nu există o definire exactă între aceste două faze, iar pe durata dezvoltării se poate face de mai multe ori tranziţie între cele două faze. Similar cu faza de analiză, faza de design este alcătuită din mai mulţi paşi care se pot intercala.

6.1 Gestionarea tipurilor de agenţi

Acest pas presupune observarea artefactelor produse în faza de design şi luarea deciziilor în privinţa tipurilor de agenţi din diagrama de agenţi. Se poate decide împărţirea tipurilor existente în subtipuri sau unirea unora dintre tipurile existente într-un singur tip. În această fază trebuie să se ţină cont de următoarele:

- Evitarea duplicării datelor.

- Evitarea duplicării codului care accesează resurse.
- Împărţirea agenţilor trebuie făcută, doar dacă este necesar.
- Fiecare agent trebuie să se afle pe o singură maşină-
- Evitarea agenţilor care au prea mult cod şi sunt complecşi.

În cadrul proiectelor care au un număr mic de tipuri de agenţi, operaţiile asupra tipurilor de agenţi nu sunt considerate problematice.

6.2 Specificarea interacţiunilor

În cadrul acestui pas se specifică interacţiunile pentru fiecare tip de agent. Interacţiunile pot fi prezentate sub formă de tabel în care se specifică interacţiunea, responsabilitatea, protocolul folosit, rolul (iniţiator sau participant), agentul cu care se interacţionează şi momentul în care interacţiunea trebuie să aibă loc.

6.3 Şabloane de mesaje

La acest pas se specifică formatul obiectelor de tipul MessageTemplate care vor fi folosite pentru a se realiza interacţiunile. Obiectele de tipul MessageTemplate vor fi adăugate la comportamentele care se ocupă de comunicarea cu ceilalţi agenţi. Pentru a realiza cât mai eficient şabloanele de mesaje, se poate ţine cont de următoarele reguli:

- ID-urile asignate agenţilor trebuie să fie unice.
- Comportamentele care se ocupă de comunicare şi care folosesc acelaşi şablon, ar trebui să formeze un singur comportament.
- Analizarea conflictelor şi modificarea clasei MessageTemplate folosită de comportamentul care se ocupă de comunicare.

De asemenea în cazul în care şabloanele create nu acoperă toate cazurile posibile de comunicare, se pot folosi şi şabloane dinamice, care pot fi modificate în timpul execuţiei.

6.4 Identificator unic pentru serviciile de căutare/înregistrare

Pentru ca agenţii să se poată înregistra la serviciul Yellow Pages oferit de platforma JADE, aceştia trebuie să aibă un identificator unic. În general convenţiile de nume depind de domeniul pe care rulează platforma JADE, fiind indicat să se folosească şi o notare folosind limbajul natural.

6.5 Accesare resurselor de către agenţi

În unele cazuri agenţii trebuie să interacţioneze cu resurse externe cum ar fi baze de date, fişiere sau alte sisteme software. Resursele pot fi împărţite în două categorii:

- Resurse pasive: resurse care îşi schimbă starea doar când sunt accesate de agentul care le controlează. Resursele pasive pot fi accesate folosind pachetul java.io.* în cazul fişierelor, în cazul librăriilor C se poate folosi JNI, iar în cazul bazelor de date se poate folosi JDBC.
- Resurse active: resurse care îşi pot schimba starea independent de agentul care le controlează. Resursele active pot reprezenta o bază de date în care se introduc date de un operator uman sau de o altă aplicaţie software.

6.6 Comportamentele interne ale agenţilor

Sarcina pe care un agent trebuie să o execute este executată în cadrul unui "comportament". Platforma JADE oferă un pachet de clase specializate, care oferă suport pentru majoritatea comportamentelor. În cazul în care comportamentele deja existente nu satisfac cerinţele, acestea se pot extinde.

- OneShotBehaviour - implementează o sarcină atomică, care rulează o singură dată şi se termină imediat.
- CyclicBehaviour - implementează o sarcină care este tot timpul activă şi care realizează aceeaşi sarcină de fiecare dată când este programată.
- TickerBehaviour - implementează o sarcină care execută periodic aceeaşi operaţie.
- WakerBehaviour - implementează o sarcină care rulează o singură dată după un anumit interval de timp.

În cazul în care se operează cu responsabilităţi complexe, este indicat ca acestea să fie împărţite în mai multe sarcini simple care pot fi combinate mai apoi folosind una dintre clasele JADE care oferă suport pentru comportamente compuse:

- SequentialBehaviour - implementează o sarcină compusă, care îşi programează sub-sarcinile secvenţial.
- FSMBehaviour - implementează o sarcină compusă, care îşi programează sub-sarcinile în funcţie de rezultatele pe care acestea le returnează la sfărşitul execuţiei.

6.7 Definirea de ontologii

Când agenţii din sistem interacţionează, schimbă informaţii care se referă la entităţi abstracte sau concrete şi care există în mediul în care se află agentul. Aceste entităţi pot fi reprezentate prin tipuri primitive de date, cum ar fi String sau Int, sau pot avea o structură complexă. Entităţile complexe pot fi referite prin termenul de concepte. Entităţile sunt legate prin intermediul unor relaţii, care pot fi adevărate sau false. La fel ca entităţile complexe, relaţile pot avea de asemenea structuri complexe. Relaţile complexe pot fi referite ca predicate.

Un caz particular de entităţi complexe este reprezentat de acţiunile pe care agenţii le pot realiza. Acţiunile agenţilor pot fi descrise folosind clasa AgentActions disponibilă în librăria JADE.

O ontologie reprezintă un set de concepte, predicate şi acţiuni ale agenţilor care au o legătură cu un anume domeniu.

References

[1] A. H. Bond and L. Gasser (Eds.), Readings in Distributed Artificial Intelligence, Morgan Kaufmann Publishers: San Mateo, CA, 1988.

[2] S. Bussmann, N. R. Jennings, and M. Wooldridge, Multiagent Systems for Manufacturing Control, Springer-Verlag, 2004.

[3] G. Caire and D. Cabanillas, "JADE tutorial: creating and using application specific ontologies", 2004, see: http://jade.tilab.com/doc/CLOntoSupport.pdf.

[4] G. Caire, W. Coulier, F. Garijo, J. Gomez, J. Pavon, F. Leal, P. Chainho, P. Kearney, J. Stark, R. Evans, and P. Massonet, "Agent Oriented Analysis Using Message/UML," Lecture Notes in Computer Science, M. Wooldridge, G. Weiss, and P. Ciancarini (Eds.), Springer-Verlag, vol. 2222, 2002, pp. 119-135.

[5] C. Campo, Directory Facilitator and Service Discovery Agent, FIPA Document Repository, 2002, see: http://www.fipa.org/docs/input/f-in-00070/f-in-00070.pdf.

[6] A. Collinot, A. Drogoul, and P. Benhamou, "Agent oriented design of a soccer robot team," in Proceedings of the 2nd International Conference on Multi-Agent Systems (ICMAS-96), Kyoto, Japan, 1996, pp. 41–47.

[7] R. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng, "Modeling agent conversations with colored petri nets," in Workshop on Specifying and Implementing Conversation Policies, 1999, pp. 59-66.

[8] A. Dennis and B. H. Wixom, Systems Analysis and Design: An Applied Approach, John Wiley and Sons, 2000.

[9] Foundation for Intelligent Physical Agents (FIPA), see: http://www.fipa.org/.

[10] M. R. Genesereth and S. P. and Ketchpel, "Software Agents," Communication of the ACM, vol. 37(7), 1994.

[11] M. Wooldridge, N. R. Jennings, and D. Kinny, "The gaia methodology for agentoriented analysis and design," Autonomous Agents and Multi-Agent Systems, vol. 3(3), pp. 285-312, 2000.