Editarea vizuală a modelului de obiecte active

Tudor Groza
Universitatea Tehnică Cluj-Napoca
Bariţiu 26-28, 400027, Cluj-Napoca, Cluj
Tudor.Groza@cs.utcluj.ro

Dorian Gorgan
Universitatea Tehnică Cluj-Napoca
Bariţiu 26-28, 400027, Cluj-Napoca, Cluj
Dorian.Gorgan@cs.utcluj.ro

REZUMAT
Lucrarea de faţă îşi propune să ofere o soluţie de editare vizuală a modelului de obiecte active. Editorul are ca scop asistarea utilizatorul în procesul de modelare a entităţilor active şi pasive existente în aplicaţiile din lumea reală. Eforturile principale în dezvoltarea acestuia s-au axat în special pe găsirea unei modalităţi cât mai naturale de reprezentare a acestor entităţi conform modelului obiectelor active. Prezenţa a două stiluri de modelare, atât vizuală, cât şi textuală, oferă utilizatorului flexibilitate. Pe parcursul întregii modelări, utilizatorul va fi îndrumat, prin intermediul tehnicilor de interacţiune folosite, înspre păstrarea consistenţei modelului, dar şi a fidelităţii acestuia faţă de modelul teoretic. Rezultatul final va fi o specificaţie în AOML (limbajul modelului obiectelor active) care va putea fi ulterior executată cu ajutorul unui proces executor.

Categorii şi descriptori ai subiectelor
D.1.7 [**Visual Programming**]

Termeni generali
Termenii generali care pot fi folosiţi sunt aleşi dintre următorii: *Documentation*, *Design*, *Experimentation*, *Standardization*, *Languages*, *Theory*.

Cuvinte-cheie
Modelare, entităţi active, entităţi pasive, tehnici vizuale.

1. INTRODUCERE
Interesul pentru proiectarea şi dezvoltarea de aplicaţii vizuale interactive a crescut considerabil în ultimii ani. Motivul care stă la baza acestui interes este posibilitatea de a permite accesul unui număr maxim de persoane la aplicaţii software, având scopuri foarte variate în contexte la fel de variate.

Modalitatea vizuală reprezintă un canal de comunicare cheie în tehnicile de interacţiune utilizator, în ciuda răspândirii puternice a tehnologiei multimedia, care adaugă, de exemplu, voce sau gesturi. Interfeţele interactive vizuale reprezintă de asemenea şi un domeniu de aplicaţie provocator pentru metodele formale. Structura lor internă devine tot mai complexă. Motivul acestei complexităţi este creşterea continuă a bazei de suport pentru un număr cât mai variat de tehnici de interacţiune, utilizatori, sarcini sau medii cu tot mai multe dialoguri active concomitent.

De fapt, baza tuturor acestor abordări stă în faptul că sistemul care trebuie considerat este cel mai complex posibil, şi anume, utilizatorul uman, al cărui comportament nu poate fi anticipat în detaliu.

Vizualizarea reprezintă tehnica de a crea imagini, diagrame sau animaţii pentru a comunica anumite mesaje. Tehnicile de interacţiune vizuală implică interactivitate prin manipulare directă a unor entităţi care au o prezentare grafică în interfaţa utilizator. De aici se ajunge la limbajele de programare vizuală care permit unui utilizator specificarea unui program în două sau mai multe dimensiuni. Diferenţa faţă de limbajele non-vizuale este dată de faptul că interpretoarele celor din urmă procesează programele ca un stream unidimensional de caractere.

Practic, noţiunea de programare vizuală implică dezvoltarea de programe prin intermediul unor expresii construite vizual. În general, utilizatorul va dezvolta, folosind abordarea vizuală, semantica, structura şi funcţionalitatea unui program. Orice limbaj a cărui entităţi de programare au o prezentare grafică poate fi numit limbaj de programare vizuală.[1]

Restul lucrării este organizat în felul următor: Secţiunea 2 introduce conceptul de model al obiectelor active care stă la baza acestui editor. În secţiunea 3 este descrisă modalitatea de reprezentare a entităţilor active şi pasive în editor. Secţiunea 4 detaliază modul de realizare al interacţiunii model-utilizator. În secţiunea 5 sunt analizate câteva rezultate experimentale. La final, secţiunea 6 prezintă concluziile şi dezvoltările ulterioare.

2. MODELUL OBIECTELOR ACTIVE
Evoluţia modelelor computaţionale a trecut prin diferite etape. Pornind de la abordarea procedurală, care utilizează proceduri şi funcţii înlănţuite în algoritmi pentru rezolvarea unui anumit tip de problemă, trecând prin abordarea obiectuală care utilizează încapsularea datelor şi conceptul de obiect pentru a modela cât mai natural procese din lumea reală, ajungem la modelul obiectelor active care încearcă să ofere o soluţie de compromis pentru aplicaţii de tipul simulărilor dinamice sau realitate virtuală, domenii în care paradigmele anterior menţionate nu fac faţă.

„Modelul obiectelor active constă dintr-o serie de entităţi active şi pasive. Este un model dinamic cu o evoluţie

paralelă, concurentă şi adaptivă în spaţiul virtual nativ. Fiecare componentă activă are un comportament privat care este definit pe baza noţiunii de traiectorie în contextul spaţiului şi timpului. Evoluţia modelului este influenţată de comportamentul fiecărei componente, reprezentând un rezultat al interacţiunilor între entităţile active. Facilitatea manipulării directe a entităţilor active necesită o utilizare corespunzătoare a tehnicilor de programare vizuală şi astfel AOM (modelul obiectelor active) devine suportul de bază pentru aplicaţii precum cele menţionate mai sus."[2]

Persistenţa modelului este menţinută prin intermediul AOML, limbajul modelului obiectelor active, o specificare de nivel înalt care menţine şi proprietăţile dinamice ale modelului. În acest fel, problema portabilităţii este redusă la implementarea modulelor de import şi export, cu analizarea (în momenul încărcării) şi generarea fişierelor sursă (în momentul salvării). Mai multe detalii legate de principiile de modelare, se pot afla consultând [2].

3. REPREZENTAREA ENTITĂŢILOR ACTIVE ŞI PASIVE

Metafora principală aleasă pentru a reprezenta modelul teoretic AOM în implementare a fost *obiectul*. Alegerea este justificată de faptul că noţiunea este una familiară, intuitivă şi larg răspândită în mediul IT. De asemenea, paradigma obiectuală se mapează corespunzător pe necesitatea reprezentărilor complexe ale proceselor şi noţiunilor prezente în modelul obictelor active.

În general, obiectele din lumea reală au o stare (aspectul static) şi un comportament (aspectul dinamic) proprii. Obiectele software încearcă să modeleze obiectele din lumea reală prin prisma structurii lor, dar şi a informaţiilor privind utilitatea şi funcţionalitatea lor. În acelaşi timp, obiectul poate fi văzut ca un concept de comunicare, necesar specificării unei interacţiuni optime între utilizator şi aplicaţie. Acestea au fost motivele care au stat la baza utilizării *obiectului* ca abstractizare principală în proiectarea interfeţei utilizator. *Obiectul* va fi utilizat atât în modelarea entităţilor AOM, cât şi a tuturor informaţiilor adiacente necesare procesului de editare.

În cazul nostru, un *obiect* este descris de următoarele concepte:

- *Proprietăţi* – modelează starea statică a obiectului. Ele conţin atât prezentarea vizuală, cât şi însuşirile non-vizuale
- *Operaţii* – definesc funcţionalitatea obiectului. O operaţie reprezintă un set bine definit de acţiuni care trebuiesc expuse utilizatorului, astfel încât acesta să poate opera asupra lor. Expunerea operaţiilor specifice fiecărui obiect se face prin intermediul tehnicilor de interacţiune.
- *Relaţii* – modelează apartenenţa obiectelor la sistem. Evoluţia finală a modelului este determinată în mare măsură de traiectoria globală a componentelor şi a interacţiunilor dintre acestea. De aceea, înţelegerea şi reprezentarea corectă a relaţiilor dintre componente este esenţială. În principiu există trei tipuri de relaţii determinate de contextele în care evoluează obiectele din AOM: *(i)* relaţii de constrângere; *(ii)* relaţii de agregare; *(iii)* relaţii de apartenenţă la mulţimi.

Fiecare din entităţile pasive şi active ale unui model au o reprezentare structurată sub formă de obiect. Întreaga structură a modelului, care reprezintă atât suportul tranzitoriu, cât şi cel persistent, a fost proiectată astfel încât să nu sufere schimbări semnificative în timp. Facilităţile adiţionale care ar putea fi introduse, se vor putea mapa corespunzător pe acest fundament stabil.

În cele ce urmează vom prezenta câteva exemple de modelare a entităţilor prin prisma conceptului de obiect.

```
OBJECT Agent
  ATTRIBUTE Name
  ATTRIBUTE State
  ATTRIBUTE Parent: Agent
  ATTRIBUTE Position: VirtualPosition
  ATTRIBUTE Attributes: Set of Attribute
  ATTRIBUTE Behaviour: Behaviour
  ATTRIBUTE Default_visibility: {TRUE|FALSE}
  ATTRIBUTE Presentation: Presentation
  ATTRIBUTE Visibility_BoundTo: Set of Entity
  ATTRIBUTE Visibility_BoundFrom: Entity
END
```

Aşa după cum se poate observa, un agent are, printre alte proprietăţi, un nume, o stare, un părinte, etc. Toate aceste proprietăţi sunt conforme cu modelul teoretic al AOM, impunând chiar fidelitatea faţă de acesta. De exemplu, un agent poate avea numai un comportament sau numai o prezentare, la un anumit moment dat. De aceea, proprietatea *Behaviour* are tipul *Behaviour* şi nu *Set of Behaviour*.

```
OPERATOR OnEntityProperties(entity: Entity)
PRE
  Model.EntitySelected != NULL
POST
  VerifyInputEntity
  Entity.UpdateProperties
  Model.UpdateState
END
```

Exemplul de mai sus, reprezintă o operaţie care poate fi executată asupra oricărei entităţi prezente în model. Precondiţia execuţiei este ca în model să fie selectată cel puţin o entitate. Dacă precondiţia nu este îndeplinită, execuţia operaţiei este anulată. Postcondiţia operaţiei specifică acţiunile executate după terminarea acesteia. În acest caz este vorba de verificarea consistenţei entităţii,

apoi actualizarea stării entității şi în ultimul rând actualizarea stării întregului model.

4. DIALOGUL MODEL-UTILIZATOR

4.1 Metodologia de proiectare

Având în vedere că avem de a face cu un sistem interactiv, editorul prezintă două componente principale: *(i)* interfața grafică; *(ii)* partea funcțională. Cele două componente şi relația dintre ele pot fi observate în Figura 1.

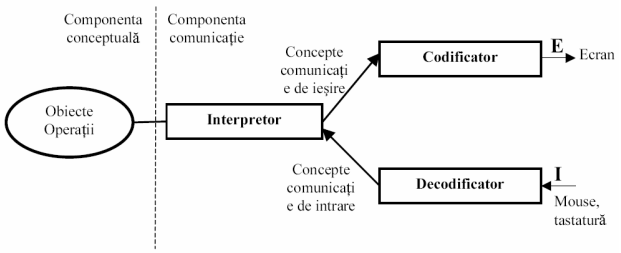

Figura 1. Metodologia de proiectare

Componenta comunicație reprezintă limbajul utilizat în comunicarea conceptelor definite în modelul conceptual. Acest limbaj se caracterizează printr-o semantică, o sintaxă şi un lexic. Semantica defineşte informații despre obiectele şi operațiile implicate în comunicația dintre utilizator şi aplicație. Sintaxa şi lexicul definesc codificarea conceptelor de comunicație transmise. Pentru a putea construi un astfel de limbaj de comunicare, a fost nevoie de realizarea unei analize amănunțite a acțiunilor pe care editorul trebuie să le efectueze pentru a crea şi modifica un model.

S-a ales această metodologie deoarece prezintă câteva avantaje, cum ar fi: *(i)* reduce numărul ciclurilor de proiectare; *(ii)* permite dezvoltarea detaliată a interfeței; *(iii)* nu implică implementare intermediară.

În final, trebuie menționat faptul că proiectarea intregii interfețe grafice a fost gândită spre menținerea fidelității față de modelul teoretic, prin ghidarea utlizatorului în crearea modelului, fără a introduce inconsistențe.

4.2 Implementarea dialogului model-utilizator

Tehnicile vizuale utilizate pentru implementarea dialogului model-utilizator se împart în urmatoarele categorii:

- Ferestre de editare cu acțiuni de tip OK, CANCEL, APPLY, HELP pentru editarea proprietăților diferitelor entități componente ale modelului
- Meniuri mobile de tip POPUP pentru alegerea uneia dintre acțiunile posibile asupra entității actual selectate
- Meniu bară pentru acțiuni generale asupra întregului model

Din punct de vedere al controalelor folosite în scopul manipulării anumitor valori, fie ele numerice sau simbolice, s-au utilizat numai controale uzuale. Motivul care stă la baza acestei alegeri a fost proiectarea unei interfețe grafice cât mai apropiată de cunoştințele unui utilizator obişnuit.

Interfața grafică se împarte în două părți. Partea stângă a acesteia este reprezentată de arborele de ierarhie al modelului. Entitățile prezente în el pot fi manipulate direct, având acțiuni corespunzătoare asociate. Partea dreptă a interfeței este reprezentată de fereastra de editare.

Pentru ca utilizatorul să nu fie legat de un mod de editare anume, aplicație oferă două posibilități de construire şi menținere a unui model. Şi anume, editarea vizuală, care este destinată tuturor tipurilor de utilizatori, fără a fi nevoie de cunoştințe suplimentare de AOML; cealaltă posibilitate se adresează utilizatorilor cunoscători ai AOML, fiind vorba de un mod textual de editare. În funcție de modul curent de editare, fereastra de editare îşi schimbă proprietățile. În cele ce urmează, vom prezenta cele două moduri de editare.

4.2.1 Modul textual de editare

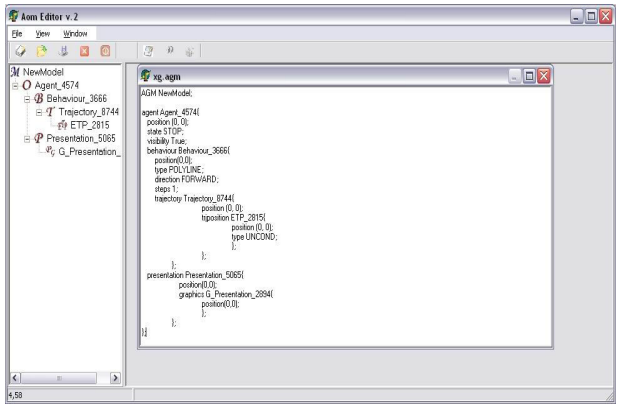

Figura 2. Modul textual de editare

Acest mod de editare este destinat utilizatorilor care sunt buni cunoscători ai AOML. Crearea şi manipularea modelului se face cu ajutorul arborelui de ierarhie şi cu ajutorul ferestrei de editare (Figura 2). Actualizarea informației între cele două părți nu se face automat. Utilizatorul este nevoit să reactualizeze manual arborele de ierarhie prin executarea acțiunii de compilare asupra ferestrei de editare (din meniul mobil al acesteia sau din meniul bară al aplicației). În cazul invers, la actualizarea fişierului, după modificarea arborelui de ierarhie, este necesară execuția unei acțiuni de actualizare asupra ferestrei de editare (din meniul mobil al acesteia).

Cu toate că interacțiunea dintre cele două părți componente ale ecranului aplicației nu este realizată automat, se

consideră că acest mod de editare se apropie de ideea compilator sau translator al fişierului suport AOML.

La trecerea dintr-un mod de editare în celălalt, interpretorul intern va avea grijă ca modificările survenite pe parcursul modelării să fie analizate pentru a păstra consistenţa generală.

4.2.2 Modul grafic de editare

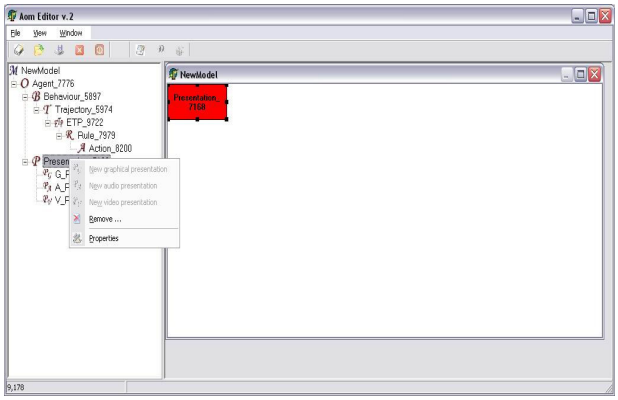

Figura 3. Modul grafic de editare

Modul grafic (Figura 3) este destinat utilizatorilor care preferă ca modelarea să se facă prin manipulare directă. Crearea entităţilor din model se face treptat, conform dorinţelor utilizatorului, dar implicit respectând modelul teoretic.

Astfel, de exemplu, după crearea unui agent, pasul următor ar putea fi ataşarea unui comportament sau a unei prezentări acestuia. Dacă se alege una din cele două opţiuni, cea selectată va fi invalidată pentru o nouă selectare, deoarece modelul teoretic afirmă ca un *agent poate avea ataşat un singur comportament şi o singură prezentare la un moment dat*. Respectiva opţiune va redeveni validă automat în momentul în care se face detaşarea entităţii selectate de agent. Un astfel de exemplu poate fi observat şi in figura anterioară, unde în centrul atenţiei se află entităţile ataşabile unui prezentator. Toate acestea vin să demostreze modul în care aplicaţia ghidează utilizatorul în construcţia unui model consistent şi fidel.

Editarea proprietăţilor entităţilor se face cu ajutorul unor ferestre de proprietăţi cu acţiuni de tip OK, CANCEL, APPLY. Organizarea acestor ferestre este realizată sub formă de pagini de proprietăţi, care grupează serii de atribute ale entităţii în funcţie de anumite caracteristici.

Interacţiunea dintre aceste editoare şi arborele de ierahie este realizată implicit. Astfel, execuţia unei acţiuni de tip OK sau APPLY se va reflecta automat în configuraţia modelului, şi va fi vizibilă instantaneu şi în interfaţa grafică.

Pentru ca acţiunile utilizatorului asupra obiectelor grafice să se reflecte şi în modelul propriu-zis a fost nevoie de introducerea unei interacţiuni între cele două concepte amintite. De exemplu, actualizarea poziţiei unei entităţi este realizată implicit după selectarea acesteia (selecţie observabilă atât pe obiectul grafic, cât şi în arborele de ierarhie) şi mişcarea ei prin spaţiul ferestrei obiectelor grafice.

Printre alte facilităţi prezente în aplicaţie, se numără: un editor de expresii complex, care ajută utilizatorul în creare de expresii ataşate diferitelor atribute ale entităţilor sau posibilitatea de vizualizare permanentă a poziţiei prezentatorilor grafici ataşaţi entităţilor.

5. REZULTATE

Testarea editorului s-a realizat în diferite etape. În prima fază s-a început editarea de la un model nou, s-au creat o serie de entităţi şi s-a încercat executarea tuturor operaţiilor posibile asupra acestora, pentru a se observa conservarea consistenţei şi a fidelităţii faţă de modelul teoretic.

Următoarea etapă a încercat sa scoată în evidenţă comportamentul editorului faţă de modelele persistente stocate pe suport hardware. Astfel, s-au încărcat în editor fişiere salvate anterior şi s-a testat corectitudinea interpretării, respectiv reprezentarea vizuală a acestora.

Toate testele efectuate au scos în evidenţă un comportament corespunzător per ansamblu, al editorului. Interpretarea modelelor, reprezentarea vizuală, tehnicile de interacţiune, toate au reacţionat în modul aşteptat. Cu toate acestea, s-au observat şi câteva lipsuri atât la capitolul tehnici de interacţiune, cât şi la capitolul acoperirii integrale a fundamentului teoretic. Aceste lipsuri sunt subiectul dezvoltărilor ulterioare ale editorului.

6. CONCLUZII

Modelul obiectelor active este o abordare ideală pentru realizarea de simulări sau prototipizări rapide. Având în vedere complexitatea relativă a limbajului AOML, utilitatea unui editor vizual care să ascundă detaliile sintactice ale acestuia, este semnificativă. În aceeaşi măsură, posibilitatea de editare textuală a modelului, oferă flexibilitatea necesară utilizatorului familiar cu AOML.

Efortul principal depus în dezvoltarea editorului s-a axat pe construirea unei interfeţe grafice prietenoase, cu prezentatori sugestivi şi un control atent în aplicarea operaţiilor de editare. Rezultatul este un editor robust care urmăreşte păstrarea în permanenţă a consistenţei modelului şi a fidelităţii acestuia faţă de modelul teoretic.

Dezvoltările ulterioare ale editorului actual se vor axa în principal pe îmbunătăţirea dialogului model-utilizator, prin introducerea de facilităţi, precum: auto-completarea codului (code completion) şi scoaterea în evidenţă a sintaxei (syntax highlightning) în modul de editare textual sau extinderea editării la mai multe modele concomitent, cu posibilitatea de interacţiune între ele.

7. REFERINȚE

[1] Gorgan, D. *Tehnici de programare vizuală.* http://users.utcluj.ro/~gorgan/res/vp/vp.html

[2] Gorgan, D. *Active Objects Model*. http://users.utcluj.ro/~gorgan/res/aom/aom.htm