SUPORT GRAFIC ORIENTAT PE OBIECTE PENTRU REALIZAREA DE INTERFEŢE OM - CALCULATOR

ec. Costin Pribeanu,

Institutul de Cercetări în Informatică

Rezumat: Creşterea interesului pentru interfeţele om-calculator cu caracter grafic a favorizat dezvoltarea de noi modele de interacţiune om-calculator. În acest articol se propune o abordare orientată pe obiecte în realizarea de interfeţe utilizator, care realizează o separare mai clară între partea grafică şi restul aplicaţiei. Editorul grafic propus este conceput ca un nivel intermediar între un nucleu de grafică pe calculator, realizat în maniera tradiţională, şi o clasă de aplicaţii orientate pe obiecte care se pot dezvolta deasupra acestuia.

Cuvinte cheie: interfeţe om-calculator, grafică pe calculator, modele de interacţiune om-calculator.

1. Introducere

Tehnologiile orientate pe obiecte au căpătat o importanţa critică în ultimii ani, având un impact deosebit asupra ciclului de viaţă în dezvoltarea software. Influenţa acestora s-a extins asupra unei arii largi, de la analiza şi proiectarea sistemelor complexe, până la sistemele de gestiune a bazelor de date şi realizarea interfeţei cu utilizatorul. În [1] se preconizează ca, pâna la sfârşitul acestei decade, întregul spectru software va fi orientat pe obiecte.

Complexitatea în sistemele software este inerentă şi se datorează mai multor elemente : complexitatea domeniului problemei, dificultatea de a conduce procesul de dezvoltare, potenţialul de flexibilitate al software-ului, dificultatea de a carateriza comportamentul sistemelor discrete [2]. Complexitatea software-ului reclamă noi tehnologii care să permită încadrarea dezvoltării unui produs în termenele şi bugetul stabilit. În acest sens, orientarea pe obiecte atrage prin posibilitatea de reutilizare a obiectelor şi facilităţile de abstractizare a datelor.

Un alt factor care a impulsionat programarea orientată pe obiecte l-a constituit creşterea interesului beneficiarilor în dezvoltarea interfeţelor utilizator cu caracter grafic. Software- ul grafic, care permite manipularea de ferestre şi iconiţe (icons), este mai complex şi mai dificil de dezvoltat decât cel bazat pe text. Creşterea importanţei interfeţei utilizator în structura sistemelor software a condus la dezvoltarea de noi modele de interacţiune om-calculator. În acest sens, următoarele două secţiuni sunt dedicate prezentării unor abordări actuale în realizarea de interfeţe om-calculator şi în dezvoltarea de modele de interacţiune orientate pe obiecte.

Conceptele de clase multinivel, crearea de instanţe şi definirea de metode se armonizează bine cu cerinţele de construcţie, întreţinere şi procesare a unor ierarhii de obiecte necesare unui sistem grafic. Literatura de specialitate atestă o serie de abordări orientate pe obiecte în domeniul graficii pe calculator, atât în ceea ce priveşte rezolvarea unor modele standard, cât şi în definirea unor modele originale.

În cadrul acestui articol se prezintă un editor grafic orientat pe obiecte, conceput ca o extensie a funcţionalităţii furnizate de modelul GKS, şi destinat să suporte dezvoltarea de aplicaţii grafice, orientate pe obiecte. Sistemul de grafică pe calculator propus se constituie într-un nivel intermediar, situat între un nucleu de grafică pe calculator existent şi o clasă de aplicaţii orientate pe obiecte care se pot dezvolta deasupra acestuia. Abordarea intenţionează să conducă la o separare mai clară a părţii grafice de restul aplicaţiei şi să furnizeze suport pentru realizarea de interfeţe utilizator. În două secţiuni separate se prezintă, atât cadrul de lucru teoretic aferent soluţiei adoptate, cât şi modelul obiect al editorului grafic.

2. Interfeţe utilizator cu caracter grafic

Interfaţa utilizator constituie cea mai importantă arie a unui sistem interactiv. O interfaţă bine proiectată are ca efect creşterea productivităţii, satisfacerea utilizatorului şi furnizarea de facilităţi de exploatare completă a posibilităţilor oferite de către hardware. O interfaţă utilizator bine structurată poate fi configurată cu uşurinţă pentru a raspunde diferitelor cerinţe individuale.

În sistemele de grafică pe calculator, interacţiunea om- calculator constituie o bază teoretică pentru realizarea unor aplicaţii cu grad ridicat de interactivitate. Sarcina componentelor de interacţiune o constituie preluarea inputului de la utilizator. În modelul de structură propus de Foley şi van Dam [3] se disting patru task-uri de bază în specificarea inputului : poziţia, textul, selecţia şi cuantificarea. Pentru fiecare task se pot folosi diferite tehnici de interacţiune. Prin combinaţii ale celor patru task-uri elementare se pot obţine instrumente de interacţiune compozite.

Modelul face o analogie cu limbajele de comunicare, propunând o structură în care componentele interfeţei utilizator reclamă proiectarea distinctă a patru niveluri : conceptual, funcţional, secvenţial şi lexical. Primele două niveluri se referă la înţeles, iar celelalte două la formă.

Designul conceptual stabileşte definiţia conceptelor de bază ale aplicaţiei care trebuie să fie stăpânite de utilizator. Din acest motiv, este cunoscut şi sub denumirea de model utilizator al aplicaţiei. În general, în cadrul acestei etape se definesc obiecte, relaţii între acestea şi operaţii asupra obiectelor. Designul funcţional defineşte funcţionarea detaliată a interfeţei, specificând ce informaţie este necesară pentru fiecare operaţie asupra obiectului, ce erori pot

să apară, ce rezultate (efect) are fiecare operaţie. Se mai numeşte şi design semantic.

Designul secvenţial defineşte ordonarea inputului şi outputului. Este legat de implementarea concretă a unor tehnici de interacţiune într-un mediu de dezvoltare. Se mai numeşte şi design sintactic. Designul lexical are sarcina de a face legatura cu hardware-ul. Aparţine de partea formală a interfeţei şi are sarcina de a determina în ce fel secvenţele de definire a outputului şi a inputului grafic sunt realizate prin primitive furnizate de hardware.

In [4] sunt menţionate trei stiluri de prezentare şi de interacţiune diferite, care caracterizează interfeţele om-calculator : WYSIWYG, manipularea directă şi interfeţele iconice. Există şi alte stiluri bazate pe selecţii prin meniu, limbaje de comandă, dialog în limbaj natural, dialog de tip întrebare- răspuns, dar care nu sunt specifice graficii pe calculator.

WYSIWYG (What You See Is What You Get) este caracterizat de faptul că reprezentarea cu care utilizatorul interacţionează pe ecran este în esenţă ultima imagine creată de aplicaţie. Pentru majoritatea aplicaţiilor este dificil de implementat acest concept în formă pură.

Interfeţele iconice utilizează iconiţe (simboluri grafice) ca reprezentări vizuale sugestive ale obiectelor, proprietăţilor, acţiunilor sau ale altor concepte.

Interfeţele bazate pe manipulare directă se caracterizează prin faptul că obiecte, atribute, relaţii cu care se poate opera sunt reprezentate vizual. Operaţiile sunt invocate prin acţiuni executate asupra reprezentării vizuale (în mod tipic, cu ajutorul unui mouse). În cadrul acestui tip de interfaţă, comenzile nu mai sunt invocate în mod tradiţional, prin meniu sau tastatură, ci în mod implicit prin acţiuni asupra reprezentării vizuale.

În general, interfeţele bazate pe manipulare directă incorporează şi alte stiluri de interacţiune (meniuri, tastatură), utilizarea exclusivă a manipularii directe fiind neconvenabilă ca viteză de lucru pentru operatorii experimentaţi. În [4] se afirmă că un singur stil de interacţiune nu este suficient pentru a satisface cerinţele unei interfeţe utilizator. Interfeţele modale interpretează inputul utilizator în mod diferit, depinzând de starea în care se află sistemul la un moment dat. O posibilă definire a unui mod este de stare sau de colecţie de stări, în care numai un subset din interacţiunile posibile este permis. Modurile furnizează un context de operare, atât pentru sistem, cât şi pentru utilizator.

În cadrul unui mod, prompting-ul şi help-ul pot fi mult mai specifice, iar meniurile mai scurte şi mai uşor de traversat, fapt care măreşte productivitatea utilizatorului. Trebuie să se furnizeze feedback care să indice modul curent şi comenzile disponibile. De asemenea, trebuie să se includă modalitaţi simple şi rapide pentru ieşirea din mod. Sistemele de administrare a ferestrelor furnizează suport pentru realizarea de moduri cu vizibilitate ridicată.

Sintaxa limbajului de comandă are o influenţă importantă asupra structurii modului. În unele aplicaţii, utilizatorul tinde să execute mai multe operaţii asupra aceluiaşi obiect sau să execute aceeaşi operaţie asupra mai multor obiecte. Aceasta sugerează utilizarea unei sintaxe de mod-comandă.

O discuţie asupra nivelului sintactic în conexiune cu realizarea de interfeţe modale este făcută în [4]. Sunt prezentate aspecte ale utilizării sintaxelor postfixate (se selectează obiectul şi apoi se dă comanda) şi prefixate (se specifică o comandă care se aplică apoi mai multor obiecte). Este prezentat conceptul CSO (Currently Selected Object) care este util în cazul factorizării unui obiect care a fost selectat.

Structura unui editor grafic utilizat pentru descrierea schemelor de proces, care se încadrează în categoria interfeţelor bazate pe structuri de interacţiune modale, este prezentată în [5]. Este implementat conceptul de mod-comandă pentru funcţiile de vizualizare, transformare geometrică, multiplicare (copiere) de entităţi şi sunt definite moduri de operare (niveluri de editare) specifice nivelurilor ierarhice ale entităţilor grafice.

În prezent, după unii autori [6], interfeţele modale sunt considerate inferioare, intrucât creează unele dificultăţi utilizatorului care trebuie să ştie în ce mod se află, cum se poate ajunge într-un anumit mod, ce operaţii sunt permise în fiecare mod, cum se poate ieşi dintr-un anumit mod. Deşi unele din deficienţe se pot elimina (prin feedback, help), rămâne totuşi problema restricţionării operaţiilor permise în fiecare mod. O bună reutilizare a componentelor interfeţei utilizator necesită o tratare ortogonală, în sensul unei inerţii în definirea setului de operaţii şi rezolvarea specifică (efectul) în cadrul fiecarui mod de operare. De asemenea, o problemă dificilă o constituie manipularea datelor şi transmiterea rezultatelor între două moduri diferite.

În [7] este făcută o trecere în revistă a principalelor abordări în domeniul realizării de interfeţe utilizator şi instrumentelor pentru construirea acestora. Clasificarea distinge între pachete de instrumente (toolkits), editoare de interfeţe utilizator, sisteme de administrare a interfeţei utilizator (UIMSs) şi generatoare de aplicaţii.

În [8] sunt menţionate sistemele X-Toolkit şi Peridot. X-Toolkit este conceput astfel încât să ascundă implementarea unor tehnici individuale de acţiune. Din acest motiv, multe dintre componetele sistemului nu oferă un cadru pentru dezvoltarea de noi tehnici de interacţiune. De asemenea, este necesar studiul şi furnizarea multor detalii pentru a implementa cu succes fiecare tehnică de interacţiune. Sistemul Peridot se bazează pe o abstractizare a conceptelor şi prin definirea unor protocoale simple de comunicatie între

diferite niveluri, care fac astfel posibilă dezvoltarea de noi tehnici de interacţiune.

O altă clasificare [9] grupează componentele pentru interfeţe în biblioteci de tehnici de interacţiune. Aceste biblioteci reprezintă colecţii de dispozitive logice input/output, editabile, care conduc la creşterea calităţii interfeţei utilizator şi a consistenţei metodelor de interacţiune, atât în interiorul acesteia, cât şi de-a lungul mai multor interfeţe.

În prezent, bibliotecile de tehnici de interacţiune constituie o tehnologie matură, care se îndreaptă spre o standardizare de facto. Acestea furnizează programatorului o bibliotecă de componente de bază pentru dialog (widgets) cum sunt meniuri, butoane. Editoarele pentru construirea de interfeţe utilizator permit crearea şi compunerea unor asemenea componente în mod interactiv şi stocarea lor în fişiere de resurse utilizabile de către program. O deficienţă care caracterizează majoritatea instrumentelor din această categorie o constituie faptul că nu captează dinamica interacţiunii (creare dinamică, răspuns la reacţia utilizatorului).

Sistemele UIMS (User Interface Management Systems) adresează problematica construirii complete a dialogului, incluzând relaţiile între componente şi având un rol comparabil compilatoarelor în producerea de cod. Aceste sisteme sunt destinate pentru realizarea de compilatoare de interfeţe şi colectarea mecanismelor de interacţiune într-un singur sistem.

Exemple de abordări în acest sens sunt prezentate în [10,11]. Ca model de structură, sistemele UIM cuprind trei module care realizeaza trei dintre nivelurile cuprinse în modelul de structură definită de Foley şi van Dam :

- modulul de prezentare (nivelul lexical al interacţiunii);
- modulul de control al dialogului (nivelul sintactic);
- modulul de interfaţă cu aplicaţia (nivelul semantic).

În [9] se menţionează ca deficienţe ale UIMSs, faptul că acestea conţin prea multe referinţe către structura aplicaţiei, sunt complexe şi ridică probleme dificile în specificare.

Generatoarele de aplicaţii oferă facilităţi pentru controlul outputului, de editare şi de comportament interactiv, fiind dedicate pentru a genera aplicaţii într-un anumit domeniu. Un domeniu este caracterizat de un model de date, care descrie informaţia ce va fi procesată de aplicaţie. Din punct de vedere semantic, aplicaţia este reprezentată printr-un scenariu.

3. Modele de interacţiune orientate pe obiecte

Paradigma MVC (Model-View-Controller) din Smalltalk a influenţat dezvoltarea interfeţelor utilizator datorită unei diviziuni care împarte responsabilităţile pentru interfaţa utilizator în trei părţi :

- model : reprezintă structura de date a aplicaţiei; conţine şi/sau are acces la informaţia care se afişeaza în diferite vederi;
- vedere : administrează partea grafică; solicită informaţie de la model şi o afişează; o vedere poate fi continută într-o supravedere şi poate să conţină subvederi în cadrul unei ierarhii care furnizează un comportament de tip windowing (clipping, transformări);
- controller: furnizează interfaţa între inputul utilizator şi modelul / vederea, asociate şi gestionează interacţiunea cu alte controllere.

Ciclul de interacţiune standard, prezentat în figura 1, se desfăşoară astfel : utilizatorul efectuează o acţiune input, controlerul activ răspunde prin invocarea unei acţiuni corespunzatoare asupra modelului [6]. Modelul execută acţiunea respectivă (care îi poate determina o schimbare de stare) şi transmite un mesaj către toate vederile asociate. Fiecare vedere poate interoga modelul asupra noii stări şi poate actualiza imaginea (dacă este necesar).

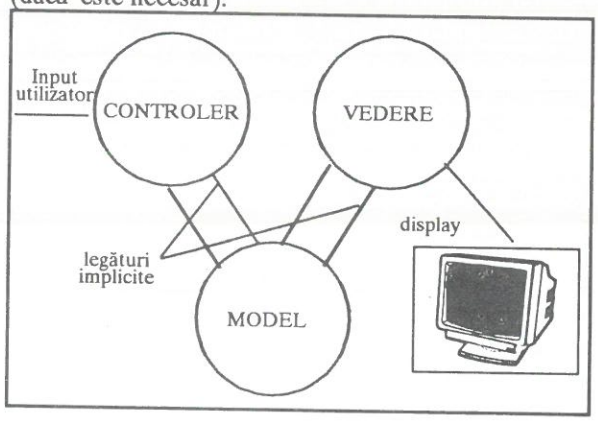

Figura 1. Modelul de interacţiune MVC

Avantajul acestui concept constă într-o diviziune elegantă în trei părţi, la nivelul abstract, fapt care constituie o bază în ghidarea proiectării de componente independente pentru interfeţe utilizator.

Din punctul de vedere al implementatorului, nu este importantă separarea vizualizării şi controlerului în două module distincte. Din acest motiv, multe din implementările acestui concept asociază modelului perechi de controler-vedere, fapt care limitează utilizarea controlerului la vederea şi modelul cu care este asociat, conducând la o slabă reutilizare a componentelor [6].

Interfeţele pentru manipulare directă sunt sisteme modale care nu au deficienţele menţionate în secţiunea anterioară. Practic, acestę interfeţe împart ecranul în mici arii-mod care interacţionează cu utilizatorul în mod diferit. Modurile sunt permanent vizibile, intrarea

şi ieşirea din mod se fac simplu, cu ajutorul mouse-ului. Cel puţin unul dintre atribute este diferit de cele ale modurilor din regiunile învecinate.

In [6] este prezentată o abordare orientată pe obiecte, pentru construirea de interfeţe utilizator cu facilităţi de manipulare directă. Câteva trăsături caracteristice ale sistemului MODE, propus în această lucrare, sunt prezentate în continuare.

Entitatea de bază o constituie modul care este definit prin trei atribute : aparenţă, interacţiune şi semantică. Mediul de lucru oferă posibilitatea de definire a unui mod şi furnizează reguli de compoziţie care fac posibilă compunerea unei ierarhii de moduri.

Într-o proiectare orientată pe obiecte, un mod este un obiect, cele trei componente fiind obiecte deţinute de obiectul mod conform structurii prezentate în figura 2. Obiectele comunică printr-un protocol intern. Pentru fiecare stare a aparenţei este definit un obiect, schimbarea stării conducând la înlocuirea obiectului afişat.

Ciclul standard de interacţiune este acelaşi ca în paradigma MVC. Obiectul interacţiune detectează inputul utilizator şi îl procesează local, prin apelarea obiectului aparenţă. Când inputul indică o acţiune semantică, obiectul semantic este activat pentru procesarea comenzii. Rezultatul poate fi trecerea controlului către aplicaţie, către un alt obiect semantic cu care este conectat sau doar schimbarea aparenţei (actualizarea stării).

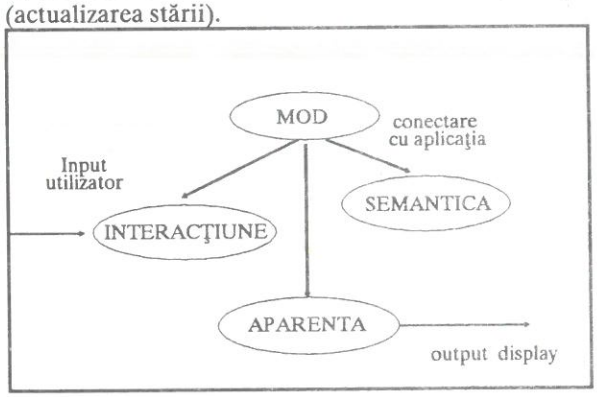

Figura 2. Structura sistemuluiMODE

Este de menţionat că, în timp ce în abordarea MVC este interogat modelul şi actualizată vederea, în cadrul de lucru MODE se furnizează doar structura în care cooperează cele trei componente [6]. Cadrul de lucru, propus de sistemul MODE, se constituie într-o bază pentru proiectarea ortogonală (independentă) a componentelor unei interfeţe utilizator.

Sistemul Papillon, prezentat în [9], a fost proiectat ca un subsistem grafic, configurabil pentru CIM. Obiectivul a fost de a crea instrumente, atât pentru dezvoltarea aplicaţiei, cât şi a interfeţei utilizator. Sistemul permite generarea automată a interfeţei utilizator, fară a fi necesară o codificare directă.

Modelul Papillon, creat pentru a furniza o interfaţă separabilă, orientată pe obiecte, cuprinde 5 module legate pentru a forma o interfaţă completă pentru fiecare obiect sau ADT (Abstract Data Type). Comunicaţia între interfaţa utilizator şi aplicaţie se face exclusiv prin aceste componente, neexistând apeluri directe pentru comunicaţie. Separarea este benefică, atât prin modularitatea codului de interacţiune, cât şi prin posibilităţile oferite pentru o dezvoltare independentă a aplicaţiei şi a interfeţei utilizator.

Structura modelului evidenţiază modulele REQ, SEM, REP (request, semantics, representation) care implementează etapele de achiziţie date, execuţie şi feedback, modulul CON (control), care controlează interacţiunea şi modulul MODEL, care realizează interfaţa cu aplicaţia.

Ciclul de interacţiune standard începe prin selectarea unei opţiuni din meniu. Modulul CON apelează câte o funcţie din fiecare dintre cele trei module REQ, SEM, REP şi apoi solicită aplicaţiei, prin intermediul componentei MODEL, să execute operaţia. Modulul REQ asigură implementarea inputului grafic, având rolul de a colecta parametrii şi de a verifica sintactic şi lexical datele introduse. Modulul SEM implementează semantica interfeţei, incluzând toate comenzile care modifică structura de date a aplicaţiei. Funcţiile de reprezentare a informaţiei grafice sunt realizate de modulul REP.

O soluţie în rezolvarea problemei de dezvoltare de noi tehnici de interacţiune, pe baza unui set existent, este dată de modelul stratificat prezentat în [8]. Acest model propune o integrare, într-o ordine pornind de la hardware la operator, a unor niveluri de interacţiune constând în evenimente fizice, evenimente abstracte, instrumente de interacţiune de bază (butoane, meniuri) şi instrumente compozite, definite ca paternuri de interacţiune.

O structură a acestui model evidenţiază la nivelul superior paternurile de interacţiune, care sunt exprimate într-un format liber, ca expresii de nivel înalt, destinate să creeze noi abstractizări. Acest nivel are sarcina de a ordona interacţiunea în secvenţe, alternative, repetiţii. O aplicaţie realizată în aceasta filozofie este o ierarhie de paternuri şi instrumente de interacţiune. Fiecare patern este implementat ca un obiect.

4. Editor grafic orientat pe obiecte

În secţiunile precedente au fost evidenţiate o serie de cerinţe în grafica pe calculator, care pot fi mai bine satisfăcute de paradigma orientării pe obiecte:

- separarea graficii de partea specifică aplicaţiei;
- cadru de lucru pentru descrierea de interfeţe utilizator cu facilităţi în descrierea acţiunilor

complexe;
- suportarea de modele ierarhizate şi facilităţi de manipulare directă a entităţilor.

În [12] se prezintă o serie de consideraţii privind oportunitatea definirii unei noi generaţii de standarde de grafică pe calculator, orientate pe obiecte. Standardizarea nu se poate baza pe nucleele grafice orientate pe obiecte, întrucât acestea sunt definite în mod dependent de limbaj şi nu furnizează întreaga funcţionalitate care s-a dovedit a fi necesară.

Nucleele grafice existente, orientate pe obiecte, nu furnizează un set suficient de complet de funcţii de bază pentru o programare grafică independentă de dispozitiv. Nu este furnizat un model pentru generare, menţinere şi arhivare pe termen lung a ierarhiilor de obiecte grafice. În general, aceste nuclee se încadrează în categoria sistemelor de instrumente (toolbox).

La momentul definirii standardelor de grafică, limbajele şi metodologia orientate pe obiecte nu au fost luate în consideraţie. Ca urmare, grafica pe calculator nu a fost influenţată de paradigma orientarii pe obiecte. În general, standardele existente s-au concentrat asupra funcţionalităţii, neglijând sarcina de a furniza un limbaj adecvat pentru descrierea entităţilor grafice şi trăsăturilor acestora. De asemenea, diversitatea cerinţelor unor categorii diferite de utilizatori a condus la definirea în paralel a unor standarde de grafică.

Cerinţele de portabilitate la diferite niveluri (aplicaţie, limbaj, dispozitiv grafic), conduc la ideea de a căuta o convergenţă (în sens sinergetic, de efect conjugat) între cele două concepte, care să îmbine avantajele utilizării unui model standardizat de grafică pe calculator cu cele furnizate de paradigma orientării pe obiecte. În acest sens, potenţialul de cuplare dintre grafica pe calculator şi abordarea orientată pe obiecte poate fi studiat din mai multe puncte de vedere :

a) utilizarea conceptelor specifice orientării pe obiecte pentru rezolvarea definiţiilor unui model existent de grafică pe calculator: o asemenea abordare presupune reproiectarea, eventual extinderea structurii de date a nucleului de grafică şi restructurarea funcţiilor interne care implementează filozofia acestuia, sarcina dificilă, dată fiind complexitatea modelelor existente;

b) realizarea unui nivel intermediar între nucleul de grafică pe calculator şi aplicaţie, constituit dintr-un set de funcţii de nivel foarte înalt (metafuncţii), orientate pe obiecte, care să permită cuplarea dintre structura de date specifică aplicaţiei şi metodele de procesare adecvate; metodele sunt construite deasupra modelului de grafică pe calculator, constituind secvenţe de funcţii ale acestuia care operează asupra structurii de date a aplicaţiei;

c) definirea şi implementarea unui nou model de grafică pe calculator, care să răspunda, atât cerinţelor satisfăcute de standardele GKS, PHIGS, CGM, CGI, cât şi cerinţelor apărute ulterior.

În cadrul acestui articol se prezintă un editor orientat pe obiecte, ca nivel intermediar între un nucleu de grafică pe calculator existent, realizat în maniera tradiţională şi o aplicaţie orientată pe obiecte. Editorul este destinat să suporte interacţiunea grafică şi să permită definirea şi modelarea entităţilor grafice, structurate pe mai multe niveluri.

Ca nucleu de grafică pe calculator se propune o implementare a standardului GKS [13], având în vedere urmatoarele considerente:
- obiectivul propus îl constituie orientarea pe obiecte a programării grafice, văzută într-un cadru mai larg al orientării pe obiecte a dezvoltării aplicaţiilor CAD / CAE; GKS are o răspândire largă, fiind cunoscute şi staţii grafice performante, care implementează hardware standardul GKS;
- este important ca modelul ales să fie riguros definit; din acest punct de vedere un standard pare a fi alegerea cea mai bună, conceptele având un grad de acceptare ridicat în mediile de cercetare / dezvoltare din domeniu;
- problematica referitoare la cercetarea şi dezvoltarea modelelor de grafică pe calculator este în sine complexă; este preferabilă alegerea unui model cunoscut şi experimentat de către autor pentru a putea analiza comparativ rezultatele şi pentru a evalua efectele abordării;
- standardul GKS defineşte niveluri de implementare, care permit extragerea de subseturi de funcţii, fapt care facilitează o etapizare mai bună a cercetării.

În construirea modelului experimental, se propune implementarea funcţionalităţii unui editor utilizabil în CADCS şi în aplicaţii de proiectare tehnologică asistată de calculator. O structură funcţională pentru un asemenea editor, construit pentru reprezentarea diagramatică a proceselor, a fost prezentată în [5] şi cuprinde :
- funcţii de modelare geometrică 2D;
- funcţii de vizualizare (zooming, panning, windowing);
- funcţii de arhivare / regăsire a entităţilor grafice;
- funcţii de simulare a proceselor.

Atributele utilizate de editor sunt din categoria atributelor statice, individuale, rămânând neschimbate pe toată perioada de existenţă a primitivei. Atributele iniţiale sau cele specificate de utilizator, pe parcursul editării, reprezintă atributele curente cu care se desenează primitivele grafice. Este important de reţinut faptul că restaurarea unei entităţi grafice, care face apel la interfaţa GKSM, poate determina modificarea setului curent de atribute.

Structura de date specifică aplicaţiei, prezentată în figura 3, evidenţiază trei niveluri de structurare : schemă, bloc şi simbol. Aşa cum s-a arătat, singura posibilitate de structurare oferită de GKS o constituie segmentarea. Aceasta poate fi completată cu utilizarea

metafişierului GKS pentru a permite o modificare a conţinutului unui segment existent. Extensia funcţională a nucleului GKS, propusă în această lucrare, are în vedere, atât o agregare de segmente în entităţi de nivel superior, cât şi o descompunere a unui segment în primitivele componente. În acest fel se realizează o mapare a a structurii arborescente a aplicaţiei pe structura liniară a nucleului grafic.

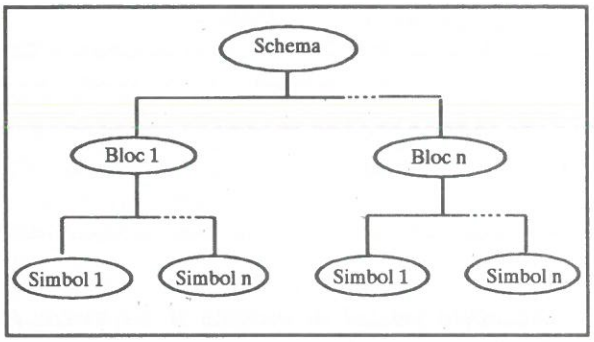

Figura 3. Structura ierarhizată a entităţilor grafice

Soluţia adoptată constă în definirea unei ierarhii de entităţi grafice, construite deasupra structurii de segmente furnizate de nucleul de grafică. Pentru fiecare tip de entitate se defineşte un context de editare separat, care realizează o abstractizare la nivelul funcţionalităţii editorului (abstractizare de acţiuni). Implementarea fiecărui context de vizualizare este realizată prin intermediul unei ferestre de editare, căreia îi corespunde o staţie de lucru GKS.

În sistemele care utilizează multiple ferestre de vizualizare se cunosc două tipuri de ferestre : suprapuse (overlapped) şi alăturate (tiled). Primul tip permite definirea de suprafeţe de lucru mai mari, dar are dezavantajul că nu toate sunt vizibile simultan. Al doilea tip permite vizibilitatea permanentă a fiecarei ferestre, cu preţul micşorării suprafeţei de lucru. Asigurarea vizibilităţii prezintă un avantaj suplimentar, eliminând operaţiile de ştergere a ferestrei deschise la revenirea în contextul apelant.

În cadrul acestei abordări, s-a adoptat o soluţie combinată: o fereastră permanent vizibilă, alocată meniurilor utilizate de interfaţa utilizator şi un set de trei ferestre suprapuse, corespunzător celor trei niveluri de editare implementate. Fereastra corespunzătoare nivelului curent de editare este complet vizibilă, iar cele aferente contextelor apelante sunt parţial vizibile.

Una dintre cerinţele importante pentru un sistem de grafică pe calculator o constituie posibilitatea de a modifica, atât tipul unei entităţi grafice, cât şi instanţele acesteia. În acest sens, abordarea propusă permite ambele operaţii :

- modificarea tipului, prin intermediul operaţiilor de salvare / restaurare, tipul entităţii fiind reţinut în arhiva GKSM şi putând fi inserat de multiple ori în cadrul unui context oarecare;
- modificarea instanţei prin intermediul funcţiilor OPEN şi CLOSE, funcţii care permit editarea entităţii într-un context separat şi revenirea în contextul apelant.

Abordarea constituie o evoluţie în raport cu o interfaţă modală (în sensul prezentat în secţiunile anterioare), întrucat nivelurile de editare utilizate sunt permanent (chiar dacă doar parţial) vizibile. Aceasta ajută utilizatorul să ştie, în orice moment, pe ce nivel se află şi cum a ajuns acolo. Trecerea de la un nivel de editare la altul se face în mod natural, fără a solicita utilizatorului operaţii intermediare.

În acelaşi timp, se furnizează utilizatorilor experimentaţi facilităţi de iterare a comenzilor în cadrul unui mod de operare. Funcţiile de ştergere, de transformare şi multiplicare de entităţi, care necesită actualizarea imaginii prin regenerare şi/sau specificarea mai multor parametri, sunt astfel implementate încât să se poată aplica unor seturi de entităţi şi/sau parametri fară a fi necesară repetarea comenzii.

5. Modelul obiect al editorului grafic

Programele operează asupra instanţelor unei clase utilizând funcţii generice. Conceptual, o funcţie generică efectuează o operaţie de nivel înalt. Pentru un set de obiecte diferite, această operaţie poate necesita o prelucrare diferită, fiecare implementare a unei anumite clase necesitând o tratare adecvată. Spre deosebire de funcţiile obişnuite, funcţiile generice specifică numai interfaţa, implementarea fiind distribuită de-a lungul unei ierarhii de clase care o utilizează [14].

În literatura de specialitate este întâlnit termenul de polimorfism pentru caracterizarea unei acţiuni al cărui nume este utilizat în comun de-a lungul unei ierarhii de obiecte, fiecare obiect în cadrul ierarhiei implementând acţiunea într-un mod adecvat (specific). Problema de a scrie proceduri care execută aceeaşi acţiune, în mod diferit pentru tipuri diferite de date, se rezolvă prin mecanismul de moştenire.

Metodele statice se utilizează pentru optimizarea vitezei şi eficienţa memorarii; metodele virtuale conferă avantajul flexibilităţii. În unele limbaje de programare conceptul de funcţie generică nu este definit, comportamentul fiind implementat exclusiv pe baza metodelor. Conceptual, se pot însă defini funcţii generice, modalitatea fiind de a apela în cadrul unei metode statice, moştenită de descendenţi, metode virtuale, implementate la fiecare nivel al ierarhiei.

În această secţiune se prezintă o descriere a funcţionalităţii editorului, evidenţiind structura pe clase de obiecte şi comportamentul aferent fiecărei

clase. Având în vedere că abordarea urmăreşte abstractizarea funcţionalităţii, structura fiecărei clase este mai puţin relevantă.

Modelul obiect al editorului grafic are la bază definirea şi implementarea a trei clase de obiecte, corespunzator celor trei niveluri de structurare a datelor. Metodele asociate fiecărei clase vor cuprinde, într-o rezolvare adecvată fiecărei clase, cvasitotalitatea funcţionalităţii editorului. Această funcţionalitate este definită la nivel de staţie grafică de lucru, permiţând o vizualizare în ferestre distincte a fiecărei clase de entităţi.

Clasificarea a avut în vedere atât definirea celor trei niveluri de editare (simbol,bloc şi schemă), căreia îi corespunde o ierarhie de clase, cât şi identificarea unei funcţionalităţi generale a unui editor, independentă de un context de editare particular. În acest scop, funcţiile de vizualizare (zoom, pan, window) şi cele de definire a primitivelor şi de specificare a atributelor au fost generalizate în cadrul unei superclase (EdWn).

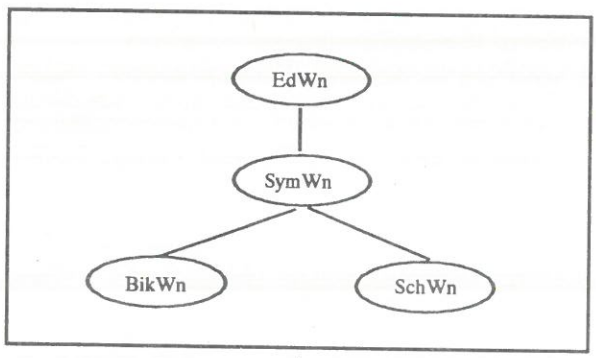

Figura 4. Structura ierarhizată a claselor de obiecte

Structura ierahică a claselor de obiecte este prezentată în figura 4 şi evidenţiază două clase descendente clasei SymWn.

Această relaţie permite transferul dintr-un context în altul al entităţii selectate pentru editare.

Pentru ca prezentarea funcţiilor editorului să fie cât mai clară, orientată pe cerinţele utilizatorului, în definiţia fiecărei clase au fost incluse numai metodele care corespund funcţiilor de editare afişate în meniu (seviciilor efectuate), evitând încărcarea descrierii cu elemente de detaliu.

Clasa EdWn este o clasă generică: nu se instanţiază, dar comportamentul acesteia este moştenit de clasele descendente. În clasa EdWn sunt incluse şi două funcţii utilitare, necesare claselor descendente, care permit multiplicarea şi transformarea unui segment GKS. Funcţiile sunt utilizate în implementarea funcţiilor generice de multiplicare şi transformare de pe fiecare nivel de editare.

Structura acestei clase cuprinde definiţia ferestrei la staţia de lucru (în spaţiul NDC) şi a viewportului asociat (în spaţiul DC), identificatorul staţiei de lucru şi definiţia zonelor de ecou, necesare iniţializării inputului grafic.

Clasa SymWn este o clasă specializată în editarea simbolurilor grafice, permiţând accesul la nivel de primitivă într-un context de editare distinct. Funcţiile de salvare, restaurare, transformare, multiplicare şi ştergere de entităţi sunt implementate prin intermediul a două tipuri de metode :

a) metode statice, la nivelul acestei clase (SymWn), având rolul de funcţii generice, apelabile din orice context descendent; cuprind partea comună, de specificare a parametrilor şi de control a interacţiunii şi apelul către funcţia virtuală corespunzatoare;

b) metode virtuale, la nivelul fiecarei clase (SymWn, BlkWn, SchWn), având rolul de a implementa partea specifică structurii de date (care implică maparea entităţilor grafice în segmente GKS).

Clasa BlkWn este o clasă specializată în editarea blocurilor alcătuite din mai multe simboluri grafice. Blocul realizează o mapare a simbolurilor grafice în segmente GKS. Moşteneşte cea mai mare parte a funcţionalităţii de la clasa ascendentă, implementarea necesitând redefinirea la nivel de metodă virtuală a operaţiilor de salvare/restaurare a unei entităţi grafice.

Clasa SchWn este o clasă specializată în editarea schemelor (instalaţiilor) alcătuite din blocuri şi/sau simboluri grafice. Este dependentă de structura de date a aplicaţiei şi de aceea redefinirea clasei necesită rescrierea metodelor care implementează funcţionalitatea.

Procedurile care asigură transferul datelor între două contexte de editare realizează funcţiile de deschidere şi închidere a unei ferestre de editare. Există două proceduri de deschidere : OPEN destinată editării unei entităţi existente şi CREATE destinată creării unei noi entităţi. În primul caz se deschide o fereastră de editare corespunzatoare nivelului ierarhic al entităţii selectate; în cel de-al doilea se cere utilizatorului specificarea nivelului de editare.

Din punct de vedere al implementării, se disting două module realizate ca unităţi Pascal, care diferenţiază două niveluri de vizibilitate a abstractizării.

Primul modul conţine implementarea claselor EdWn şi SymWn, care sunt furnizate programatorului de aplicaţie. Pe baza acestor clase se poate construi o ierarhie de obiecte care să corespundă necesităţilor unei aplicaţii particulare. Acesta este modulul de legatură între nucleul GKS şi interfaţa orientată pe obiecte.

Al doilea modul contine implementarea claselor BlkWn şi SchWn, ultima fiind dependentă de structura de date a aplicaţiei, precum şi funcţiile care asigură transferul datelor între două contexte de editare. Acest modul poate fi rescris şi completat, în funcţie de cerinţele aplicaţiei.

6. Concluzii

Orientarea pe obiecte reprezintă un mijloc de a trata complexitatea sistemelor software, nu un scop în sine. Această paradigmă realizează o translatare a centrului de greutate (interes) de la procesele utilizate pentru a construi obiecte către obiectele construite. Orientarea pe obiecte conduce la abandonarea concepţiei centrate pe proces în favoarea unei concepţii centrate pe produs, determinată de relaţia producator - consumator.

În [15] se arată că unul din elementele cheie în revoluţia industrială a software-ului îl constituie crearea unei pieţe de părţi standardizate, care să permită asamblarea de componente de nivel coborât în soluţii de nivel înalt. În acest sens, orientarea pe obiecte oferă mari avantaje, atât prin posibilitatile de realizare a componentelor, cât şi prin deschiderea către un design participativ, conferită de facilităţile de prototipizare rapidă.

Având în vedere costurile ridicate ale dezvoltării şi întreţinerii de sisteme grafice şi impactul tehnologic asupra acestui domeniu, orientarea pe obiecte în grafica pe calculator este de natură să satisfacă mai bine cerinţele realizării de aplicaţii grafice portabile, care să integreze cu uşurinţă progresul tehnologic.

Evoluţia şi tendinţele actuale în activitatea de cercetare şi dezvoltare relaţionată graficii pe calculator conduc la evidenţierea următoarelor idei :

1) cerinţe noi în abordarea interfeţei utilizator, ca urmare a dezvoltării sistemelor inteligente şi a facilităţilor oferite de medii de dezvoltare de tip Windows, reclamă extinderea funcţionalităţii modelelor la nivelul interfeţei cu aplicaţia şi cu operatorul;

2) creşterea performanţelor depinde de mărirea flexibilităţii pentru integrarea progresului tehnologic fără modificări radicale ale software-ului; o modalitate de a construi sisteme configurabile de grafică pe calculator poate fi indusă de orientarea pe obiecte, care permite implementarea unei ierahii de clase cu facilităţi de moştenire a metodelor;

3) abordarea propusă în această lucrare conferă un grad ridicat de flexibiliate, asigurând editarea intr-un context separat, prin intermediul unor clase specializate, a entităţilor grafice structurate ierarhic; definirea unei clase generale de obiecte, la nivelul vizualizării, permite moştenirea şi aplicarea cu efort minim de programare, a funcţiilor de vizualizare. De asemenea, această abordare oferă un mai mare potenţial de editare integrată a datelor alfanumerice cu cele grafice.

Bibliografie

1. **VARHOL, P.**: Object Oriented Programming. The Software Development Revolution, Computer Technology Research Corporation, Charleston, 1992.
2. **BOOCH, G.**: Object Oriented Design, The Benjamin/Cummings Publishig Co., Redwood City, 1991.
3. **FOLEY, J.D., van DAM, A.**: Fundamentals of Interactive Computer Graphics, Addison-Wesley, Reading, Massachusetts, 1984.
4. **FOLEY, J.D., van DAM, A., FEINER, S.K., HUGHES, J.F.**: Computer Graphics - Principles and Practice, Addison-Wesley, Reading, Massachusetts, 1992.
5. **PRIBEANU, C., VASILIU, C., POPESCU, D.**: Subsistem grafic off-line, Faza la tema 3.5.9 - Sisteme grafice in timp real, ICI, 1992.
6. **SHAN,Y. P.**: An Object Oriented Framework for Direct Manipulation User Interfaces, In: E.H.Blake and P.Wisskirchen (eds.), Advances in Object Oriented Graphics I, Springer-Verlag, 1991, pp.3-20.
7. **PLATEAU, D., BORRAS, P., LEVEQUE, D., C.MAMOU, J.-C., TALLOT, D.**: Building User Interfaces with the LOOKS Hyper-Object System, In: E.H.Blake and P.Wisskirchen (eds.), Advances in Object Oriented Graphics I, Springer-Verlag, Berlin, 1991, pp.35-46.
8. **LAFFRA, C., van den BOS, J.**: A Layered Object Oriented Model for Interaction, In: E.H.Blake and P.Wisskirchen (eds.), Advances in Object Oriented Graphics I, Springer-Verlag, Berlin, 1991, pp.95-116.
9. **NEELAMKAVIL, F., MULLARNEY, O.**: Separating Graphics from Application in the Design of User Interfaces, The Computer Journal, vol.33, no.5, 1990, pp.437-443.
10. **VASILIU, C., PRIBEANU, C.**: Sistem grafic în timp real - paradigmă pentru o structură de interacţiune om-maşină, în: Revista Română de Informatică şi Automatică, vol.3, no.1, 1993, pp.17-30.
11. **ENCARNAÇAO, J.L.**: Incorporating Knowledge Engineering and Computer Graphics for the Efficient and User Friendly Interactive Graphics Applications In: C.E.Vandoni (ed.) Eurographics'85, Elsevier Science Publisher B.V., Amsterdam, 1985.
12. **WISSKIRCHEN, P., KANSY, K.**: The new Graphics Standards - Object Oriented," In: E.H.Blake and P.Wisskirchen (eds.), Advances in Object Oriented Graphics I, Springer-Verlag, Berlin, 1991, pp.199-216.
13. **ISO** 7942:1985 Information processing Systems - Computer graphics - Graphical Kernel System (GKS) functional description.
14. **KEENE, S.**: Object Oriented Programming in Common Lisp, Addison-Wesley, Reading, Massachusetts, 1989.
15. **COX, B.J.**: "There is a Silver Bullet," Byte, vol.15, no.10, 1990, pp.209-218.