Tehnici de optimizare accesibile din produsele de creaţie multimedia

Conf.dr. Ion SMEUREANU,
Catedra de Informatică Economică, A.S.E., Bucureşti

Lucrarea tratează două dintre cele mai importante tehnici de optimizare implementabile prin software-ul de creaţie multimedia: partajarea script-urilor şi crearea obiectelor autoconţinute. Exemplificarea se face pe Multimedia ToolBook.
***Cuvinte cheie**: script, ierarhia obiectelor, notificare, forwarding.*

Nevoia de a crea aplicaţii multimedia independente de platforme a orientat creatorii de soft spre sisteme de tip ***author***, bazate pe limbaje ***script***. Chiar dacă aplicaţiile sunt create în mare parte în manieră vizuală, componentelor le corespund proprietăţi memorate undeva şi descrieri ale comportamentului lor, toate făcute în termenii unui limbaj de programare. Autorul poate interveni uşor în aceste programe sau poate crea altele, complet noi.

Limbajul de tip *script* descrie foarte eficient o monitorizare a comportamentului obiectelor, ca răspuns la mesajele generate de evenimente. Descrierea este foarte generală şi poate fi făcută independent de maşină, urmând ca transpunerea în practică să fie preluată ulterior de un *player* specific platformei de redare.

Dificultatea programării derivă din necesitatea identificării şi tratării tuturor posibilităţilor de interacţiune cu utilizatorul pe de o parte şi din multitudinea tipurilor de sincronizări ce trebuie realizate între părţile componente ale aplicaţiei, pe de altă parte.

Tipologia evenimentelor din sistemele multimedia este destul de vastă. O clasificare a lor după mobilul declanşării este foarte instructivă. În această idee deosebim:

- evenimente **bazate pe timp**;
- evenimente **bazate pe contextul derulării aplicaţiei** (spre exemplu, intrarea într-o pagină nouă - *EnterPage*, începerea unei acţiuni - *notifyBefore*, terminarea unei acţiuni - *notifyAfter*, intrarea într-un moment de inactivitate - *idle*.
- evenimente **externe**, declanşate de utilizator, prin acţionări de taste sau mouse;
- evenimente bazate pe **localizări spaţiale** (coliziune sau suprapuneri între obiecte, trecerea cu mouse peste un obiect (*MouseEnter, MouseLeave*) etc. şi care pot induce modificări simultane în comportamentul mai multor obiecte.

Limbajele de tip *script* sunt de regulă limbaje ale programării orientate obiect. Ele permit reutilizarea codului deja existent, prin implementarea conceptului de ***moştenire*** şi totodată o adaptare permanentă a acestuia prin mecanismul ***polimorfismului,*** realizat prin virtualizarea şi/sau supraîncărcarea funcţiilor şi operatorilor.

Independenţa faţă de maşină se realizează prin ascunderea nivelului cel mai de jos, codul generat fiind interpretat de un ***player*** abia la momentul execuţiei, când acesta cunoaşte maşina pe care se face prezentarea. Aşadar sistemele de tip ***authoring*** traduc *script*-urile într-un cod intermediar, inteligibil pentru un interpretor, care-l traduce în instrucţiuni maşină abia la momentul execuţiei. Acesta îl poate readuce într-o formă textuală, specifică de data aceasta platformei de prezentare, în vederea vizualizării adaptărilor făcute sau a operării unor eventuale intervenţii autor.

Acest mecanism în trepte, ne lămureşte şi asupra consumului mai mare de memorie şi timp calculator, depinzând în

afară de numărul, tipul şi dimensiunea obiectelor şi funcţiilor folosite, şi de tipurile de interactivitate suportate de aplicaţie sau de grup; pe spaţii virtuale; constructivă; conversaţională. Datorită consumului mare de resurse, software-ul pentru multimedia este în permanenţă supus optimizărilor. Gestiunea unitară a unor resurse, crearea de obiecte cât mai generale, partajabile între aplicaţii şi transferabile fără a necesita adaptări sunt doar câteva direcţii vizate de procesul optimizării. Vom încerca să le exemplificăm făcând apel la Multimedia ToolBook.

Partajarea *script*-urilor între obiecte

În mod obişnuit *script*-urile sunt introduse ca proprietăţi ale obiectelor, ele fiind manipulate odată cu acestea (copiere, mutare etc.). Dacă un eveniment trebuie tratat de mai multe obiecte, atunci toate obiectele care-l tratează trebuie să deţină *handler*-e adecvate modului particular în care trebuie tratat acest eveniment. Mesajul produs de eveniment este recepţionat de un obiect ţintă, este tratat de acesta şi eventual avansat celorlalte obiecte, prin clauza ***forward***, în ordinea impusă de ierarhia obiectelor.

Ierarhia obiectelor este definită în ToolBook prin următoarele reguli:

- mai multe obiecte pot forma un grup, care se comportă la rândul lui ca obiect părinte;
- mai multe grupuri sau grupuri şi obiecte, pot fi organizate într-un grup părinte;
- pagina este plasată în ierarhie deasupra tuturor obiectelor individuale şi grupurilor conţinute de ea;
- *background*-ul este părinte pentru toate paginile care-l folosesc;
- deasupra *background*-urilor se află cartea, în ansamblul ei;
- mai multe cărţi care abordează acelaşi domeniu formează un sistem de cărţi;
- sistemul ToolBook se află pe punctul cel mai înalt în ierarhie, putând gestiona mai multe cărţi sistem.

Mesajul este automat avansat în ierarhie şi dacă obiectul care a primit direct mesajul nu dispune de un *handler* pentru tratarea lui. Speculând acest lucru, pentru tratarea unui mesaj mai general, se recomandă punerea *script*-ului în pagină, sau mai sus în ierarhie, deoarece se poate ajunge aici, de la oricare obiect conţinut în pagină şi care poate fi ţinta mesajului respectiv. Nu este astfel nevoie să scriem de mai multe ori acelaşi *handler*, pentru mai multe obiecte, când tratarea mesajului este aceeaşi, însă dorim ca mesajul să fie recunoscut de mai multe obiecte.

Acest mod de lucru prezintă totuşi un dezavantaj mare. Cum deosebim obiectele care trebuie să recunoască un mesaj, de cele care nu trebuie să-l recunoască? Pentru rezolvarea acestei probleme este nevoie ca *script*-ul, deşi apare la nivelul paginii, să particularizeze acţiunea în funcţie de obiectul ţintă care a primit iniţial mesajul şi de la care a parvenit până la nivelul paginii. Acest lucru este posibil prin identificarea obiectului ţintă şi execuţia condiţionată a unor acţiuni:

```
to handle eveniment
 conditions when object of target = "obiect1"
                              actiune1;
  when object of target = "obiect2"
                              actiune2; .....
  else  forward
  end conditions
end
```

Din construcţia condiţiilor se observă că în raport cu un eveniment dat, obiectele se împart practic în două: unele care-l recunosc şi reacţionează fiecare după un comportament specific şi unele care nu-l recunosc şi îl lasă să treacă mai departe în ierarhie, pentru a-şi găsi o eventuală destinaţie. În ultimă instanţă, dacă nu şi-a găsit un *handler* adecvat, evenimentul

va ajunge până la ToolBook System, care-l recunoaşte, dar nu execută nimic.
Dacă mai multe obiecte au acelaşi comportament la un mesaj primit, se observă că o acţiune va apare descrisă de mai multe ori, în blocul condiţiilor.
O soluţie mai elegantă este partajarea *script*-urilor între obiecte. Conceptul a fost introdus tot prin mecanismul gestiunii unitare a unor resurse comune. Astfel, în meniul **Object/Resources** a fost introdusă în tipologia resurselor, alături de iconuri, meniuri etc., şi resursa de tip *SharedScript*.
Crearea unei noi resurse de tip *script* partajabil se face similar oricărei resurse, cu **managerul de resurse**, care disponibilizează în acest caz, o fereastră pentru editarea *script*-ului dorit.
Crearea resursei se poate face şi prin comenzi OpenScript:

```
new SharedScript;
name of It = "scriptul_meu";
script of SharedScript "scriptul_meu"
        = " to handle ...; end";
```

sau printr-un import de text, dintr-un fişier extern sau dintr-un câmp al aplicaţiei.
Obiectele care se abonează la o resursă *script*, menţionează ca proprietate *SharedScript*, numele unuia dintre *script*-urile partajate, prezente în carte.
Ataşarea la resursă se poate face şi prin comandă OpenScript, după modelul:

```
sharedScript of button "B"=SharedScript
"scriptul_meu";
```

iar detaşarea prin:

```
sharedScript of button "B" = NULL;
```

Se observă că împărţirea obiectelor după comportamentul lor faţă de un eveniment se face mai uşor, obiectele menţionând sau nu, printre proprietăţi, resursa de tip *SharedScript* asociată evenimentului în cauză.
Un *script* partajabil este aşadar o resursă stocată unitar, la nivelul unei cărţi şi care nu aparţine nici unui obiect individual; ea poate fi însă partajată de oricâte obiecte, care se pot conecta la această resursă.
Deşi în ierarhia obiectelor *script*-urile partajabile sunt plasate la nivelul cărţii, contextul cărora se supun este cel al obiectului care le apelează. Astfel, calificatorii *parent*, *this*, *my*, *self* vor referi contextul obiectului pentru care se rulează *script*-ul, ca şi cum ar fi vorba de un *script* al obiectului respectiv.
Dacă obiectul dispune şi de un *script* propriu, el se execută cu prioritate; dacă *script*-ul propriu conţine şi *forward*, clauza va avea în acest caz ca efect transferarea mesajului şi către *script*-ul partajat. Când clauza *forward* e conţinută într-un *script* partajat ea provoacă ciclare infinită, deoarece *script*-ul partajat e tratat ca *script* propriu, *forward* reavansând mesajul tot *script*-ului partajat.
Pentru a beneficia totuşi de un efect de tip forward, *script*-urile partajate trebuie să precizeze clauza sub forma ***forward to parent***.
În acest mod, mesajul după tratare este transmis direct obiectului părinte, sărind peste *script*-ul partajat. Se poate folosi *forward to parent* şi în *script*-ul propriu obiectului atunci când în anumite condiţii date se doreşte tratarea mesajului nu la nivelul obiectului, ci la un nivel mult mai înalt. În acest caz, mesajul trebuie să sară şi peste *script*-ul partajat, parcurgând ierarhia în sus, până la întâlnirea *handler*-ului dorit.
Un efect particular îl are şi comanda *send* plasată în *script*-uri partajabile. Când nu are o destinaţie explicită, *send* este inter-pretată ca ***send ... to self***, care în virtutea considerării *script*-ului partajat ca *script* propriu, trimite mesajul obiectului care a apelat *script*-ul respectiv. În acest fel, acelaşi *send mesaj* dintr-un *script* partajabil declanşează la nivelul fiecărui obiect un comportament specific, apelând *handler*-ul asociat mesajului, din *script*-ul propriu fiecărui obiect.

Crearea de obiecte autoconținute

Funcțiile cu care sunt înzestrate obiectele pentru tratarea mesajelor descriu în general acțiuni independente de context. Există situații în care se dorește ca reacția unui obiect la un mesaj să fie particularizată la context. În acest caz, trebuie scrise programe generale, care se adaptează la context, reducând substanțial munca de programare. Frecvent se folosesc două modalități pentru a realiza acest lucru.

Prima modalitate constă în a interoga elementele de context, în raport cu care se dorește adaptarea funcției de tratare a mesajului. Putem corela acțiunea cu numele unei pagini, cu nivelul *Reader* sau *Author*, cu starea unui buton.

O altă modalitate se bazează pe notificarea evenimentelor care modifică contextul și de care un obiect este interesat să fie informat. În acest scop, obiectul cere înregistrarea sa într-o listă a obiectelor ce sunt interesate de producerea unui eveniment și care vor fi anunțate înainte de producerea iminentă a evenimentului (**notifyBefore**), sau după producerea acestuia (**notifyAfter**). Spre exemplu, un *handler* de tipul:

```
notifyBefore reader    hide self    end
```

plasat în *script*-ul unui obiect face ca obiectul să fie anunțat înainte de producerea evenimentului de trecere în regim "vizualizare aplicație", adică *reader*, moment în care el va deveni invizibil. Pentru a redeveni vizibil la nivelul "author", ca să permită programatorului să-l modifice sau să-l folosească, obiectul tehnic trebuie să conțină în *script*-ul său și un *handler* ce se execută la revenirea în regim de lucru:

```
notifyAfter author    show self    end
```

Obiectele care se adaptează urmărind singure schimbarea contextului se numesc **autoconținute (self contained)**, deoarece ele pot fi copiate și folosite ca atare în altă parte, fără a fi nevoie de modificări în *script*-ul lor sau al altor obiecte. ToolBook introduce totuși câteva restricții în implementarea acestui mecanism, specific programării orientate obiect. Astfel, obiectele *page*, *background*, *viewer* și *book* nu pot avea *handler*-e de tip *notify*.

Gestiunea obiectelor înscrise la notificarea asupra unui eveniment se ține la nivel de pagină. Doar evenimentele care ajung la pagină pot fi notificate obiectelor ce au solicitat acest lucru. Dacă un eveniment este tratat de un obiect și nu mai ajunge până la pagină în ierarhia de obiecte (clauza *forward*) el nu va fi notificat. De asemenea, un obiect poate ascunde evenimente ce-i parvin direct sau din avalul ierarhiei, declarând un *handler* ce tratează fictiv acest eveniment (nu precizează nici o acțiune, ci doar numele mesajului recunoscut) și neindicând clauza *forward* pentru a-l lăsa să treacă mai sus, în ierarhie.

Un alt aspect specific Multimedia ToolBook este și faptul că obiectele din *background*, fiind plasate în ierarhie deasupra paginii, când primesc mesaje, indiferent dacă le tratează sau nu, mesajele nu ajung și la pagină și deci nu vor fi anunțate obiectelor deși ele au cerut notificarea acestui tip de eveniment.

Pentru a evita ciclarea la infinit, a unui mesaj prin ierarhia obiectelor, un *handler* de tip *notify* nu trebuie să conțină clauza *forward*. Nu există restricții cu privire la numărul obiectelor ce se înscriu pentru a primi notificarea asupra producerii unui eveniment.

Un *handler* de tip *notify* este declanșat în două situații: automat când s-a primit notificarea producerii evenimentului vizat, sau la o comandă **sendNotifyAfter** sau **sendNotifyBefore**, care forțează execuția *script*-ului.

Bibliografie

*** *Multimedia ToolBook, User Manual & OpenScript Reference*, Asymetrix Corporation, 1996.

Smeureanu I. - *Multimedia ToolBook, Concepte și practică*, *PC Report*, nr. 51, 52 / 1997.