Universitatea Politehnica Timişoara

TEZA DE DOCTORAT

CONCEPȚIA ȘI REALIZAREA INTERFEȚELOR CU UTILIZATORUL ÎN DEZVOLTAREA APLICAȚIILOR CAD/CAM

ing. GEORGE CRISTIAN SAVII

Conducator ştiinţific
Prof. dr. ing. Octavian Proştean

Timişoara
2006

Cuprins

1. Introducere 1
1.1. Oportunitatea şi obiectivele tezei 5
1.2. Prezentarea conținutului tezei 7
1.2.1. Indexul principalelor abrevieri 11
2. Abordări moderne în concepția şi realizarea profesională a interfețelor cu utilizatorul 12
2.1. Nevoia de studiu al IOC 12
2.1.1. Psihologia cognitivă 13
2.2. Dezvoltarea de sisteme utilizabile 16
2.2.1. Analiza 18
2.2.1.1 Analiza sarcinii 19
2.2.1.2 Metode orientate pe funcții sau pe obiecte 21
2.2.2. Proiectarea 22
2.2.3. Evaluarea 27
2.3. Interfețe cu utilizatorul 30
2.4. Implicațiile ergonomiei software în proiectarea interfețelor cu utilizatorul 39
2.4.1. Direcții de bază în ergonomia software 39
2.4.2. Dileme ale ergonomiei software 42
2.4.3. Condiții de studiu 43
2.4.4. Concluzii 44
2.5. Concluzii 45
3. Analiza şi modelarea interfețelor cu utilizatorul 47
3.1. Sisteme cu Evenimente Discrete 49
3.2. Modelarea interfețelor cu utilizatorul 56
3.2.1. Modelele principale de interfață cu utilizatorul 59
3.2.1.1 Modelarea sarcinii 60
3.2.1.2 Modelarea Dialogului 62
3.2.1.3 Dispozitive interactor 66
3.2.2. Modelarea elementelor de interacțiune concretă 72
3.3. Modelarea ca SED a unui proces standard de selecție de obiecte 75
3.4. Concluzii şi contribuții 82
4. Tehnologii software utilizate în realizarea de interfețe cu utilizatorul 83
4.1. Introducere 83

4.2. Programarea orientata spre obiecte. 85
4.2.1. Etapele de dezvoltare ale produsului 87
4.2.1.1 Analiza domeniului sistemului 87
4.2.1.2 Analiza cerinţelor sistemului 89
4.2.1.3 Proiectarea sistemului 92
4.2.1.4 Analiza cerinţelor software 92
4.2.1.5 Proiectarea software 93
4.2.1.6 Implementarea 99
4.2.1.6.1 Implementarea template-urilor 102
4.2.1.6.2 Prototipizarea 109
4.2.1.6.3 Implementarea unei componente reutilizabile 122
4.3. Proiectarea elementelor de interfaţă grafică cu utilizatorul 128
4.3.1. Utilizabilitate 128
4.3.2. Criterii pentru realizarea interfeţelor grafice cu utilizatorul 134
4.3.3. Reguli de proiectare a elementelor de interfaţă personalizate 146
4.4. Programarea vizualizării utilizând biblioteca grafică OpenGL 152
4.4.1. Utilizarea OpenGL sub Microsoft Windows 152
4.4.2. Selecţia şi feedback 159
4.4.3. Desenarea *wireframe* şi cu linii ascunse 165
4.4.4. Descompunerea prin triunghiularizare 168
4.5. Criterii de evaluare a sistemelor CAD 170
4.5.1. Studiu de caz: dezvoltarea unui sistem CAD de construcţii în lemn 174
4.6. Concluzii şi contribuţii 177
5. Interfaţa pentru un sistem CAD integrat cu aplicaţie în proiectarea construcţiilor din lemn 179
5.1. Introducere 179
5.2. Managerul de proiecte 182
5.2.1. Gestiunea unui proiect 184
5.2.2. Gestiunea unei poziţii 185
5.3. Elemente personalizate de interfaţă 186
5.3.1. Proiectarea cutiilor de dialog cu elemente speciale personalizate 186
5.3.2. Principii de funcţionare ale cutiilor de dialog 187
5.3.3. Introducerea de date 197
5.3.4. Parametri globali ai interfeţei cu utilizatorul 200

5.3.5. Bare de instrumente (Toolbars) 202
5.3.6. Operații cu fișiere 203
5.3.7. Biblioteci de modele de proiectare 204
5.3.8. Punct-simboluri (PSB) 206
5.3.8.1 Cerințe 206
5.3.8.2 Soluție de implementare a PSB 208
5.3.9. Liste de componente şi de dimensiuni generate în sistemul Dietrich's 213
5.3.9.1 Sortarea elementelor 213
5.3.9.2 Liste de materiale 215
5.4. Selecție standard de obiecte 221
5.4.1. Definirea parametrilor funcției şi a procesului de selecție 221
5.4.2. Implementarea clasei 223
5.4.2.1 Clasa template CDhpInputDeviceSelectObjects 226
5.5. Gestionare specială a cutiilor de dialog 229
5.6. Vizualizarea 3D a modelelor utilizând OpenGL 233
5.6.1. Structuri şi clase 234
5.6.2. Selecția de obiecte 238
5.6.3. Evidențierea obiectelor 242
5.6.4. Vederea cu linii ascunse 245
5.6.5. Triangularizarea 249
5.7. Concluzii şi contribuții 251
6. Concluzii finale şi contribuții personale. Perspective 253
Bibliografie 261

1. Introducere

1.1. Oportunitatea şi obiectivele tezei

Creşterea complexităţii produselor software în general, şi a aplicaţiilor din domeniul CAD/CAM în particular, dezvoltarea şi fundamentarea teoriei moderne a interacţiunii om-calculator precum şi impactul creat de progresele extraordinare ale tehnologiilor software şi mai ales hardware utilizate la proiectarea interfeţelor cu utilizatorul, creează baza teoretică şi practică a concepţiei şi realizării de interfeţe performante cu utilizatorul în dezvoltarea aplicaţiilor CAD.

Datorită faptului că majoritatea interfeţelor actuale cu utilizatorul se bazează pe interacţiunea discretă, bazată pe evenimente, ca prim obiectiv al acestei lucrări se constituie modelarea abstractă a elementelor de interfaţă cu utilizatorul ca sistem cu evenimente discrete, pentru utilizarea avantajelor metodelor din cadrul analizei sistemelor de aprofundare a comportamentului interfeţei, de optimizare a acesteia şi de evaluare a performanţelor.

Odată cu apariţia limbajelor de programare structurate, firesc s-a trecut la realizarea metodelor de analiză şi proiectare care au condus la optimizarea procesului de dezvoltare a produselor software. Programarea structurată însă, este orientată spre funcţii, tratând entităţile de date separat de funcţii. In urma creşterii complexităţii produselor software a fost necesară o restructurare a conceptelor ce stau la baza dezvoltării acestora astfel explicându-se apariţia modelului de programare orientat spre obiecte. Unul din obiectivele acestei lucrări constă în analiza avantajelor programării orientate spre obiecte şi sintetizarea etapelor de dezvoltare orientată spre obiecte a unui produs software cu detalierea în cadrul etapei de implementare a unor aspecte concrete privind template-urile, prototipizarea şi reutilizarea.

In domeniile ingineriei şi mai ales cele din domeniul construcţiilor, proiectarea implică o cantitate mare de lucru cu informaţii grafice. Un sistem interactiv de proiectare trebuie să fie orientat către utilizator, presupunând o interfaţă cu utilizatorul prietenoasă şi uşor de înţeles. Prin urmare, un alt obiectiv al tezei, de o importanţă primordială pentru proiectare, constă în cunoaşterea în profunzime a conceptelor şi a caracteristicilor implicate în proiectarea interfeţelor cu utilizatorul prin analiza problemelor referitoare la utilizabilitate

şi stabilirea unor criterii de realizare a interfeţelor cu utilizatorul, concretizându-se sub forma unei colecţii de reguli şi repere, care se pot constitui într-un ghid de proiectare a interfeţelor performante cu utilizatorul.

Grafica pe calculator a evoluat de la simpla reprezentare sub formă de linii, în prezent existând aplicaţii cu grafică realistă interactivă care rulează pe hardware disponibil în calculatoare uşor accesibile. Datorită faptului că dezvoltarea tehnologiilor hardware de accelerare a graficii pe calculator este mult mai rapidă decât creşterea nivelului de expertiză software specifică, este necesară o înţelegere aprofundată a mecanismelor graficii accelerate precum şi a modurilor sale de exploatare. In acest context, un alt obiectiv avut în vedere în cadrul tezei constă într-un studiu critic aprofundat al unor probleme complexe de afişare şi manipulare a obiectelor tridimensionale, propunându-se un set de consideraţii teoretice şi practice utilizând biblioteca grafică OpenGL, în scopul de a furniza dezvoltatorilor un instrument de lucru şi o metodologie de implementare adecvată.

Un alt obiectiv propus şi realizat este acela de prezentare a soluţiilor de aplicare şi implementare a recomandărilor şi metodelor de dezvoltare software propuse, în cadrul interfeţei sistemului CAD Dietrich's de construcţii din lemn realizată integral de către autor, cu scopul de a eficientiza lucrul cu programul prin reducerea timpului de învăţare al sistemului, accelerarea activităţilor de rutină efectuate în cadrul programului precum şi accesibilizarea aplicaţiei prin dezvoltarea unei interfeţe adecvate majorităţii utilizatorilor acestui program care, în general, nu posedă cunoştinţe avansate de operare a calculatorului.

In final, ultimul obiectiv constă în prezentarea contribuţiilor autorului în problematica dezvoltării interfeţelor cu utilizatorul performante în general, şi a metodelor, tehnicilor şi instrumentelor software specifice dezvoltării interfeţelor cu utilizatorul în particular, ca rezultat al unei activităţi de cercetare teoretică şi aplicativă desfăşurată pe parcursul a mai multor ani privind realizarea şi optimizarea (îmbunătăţirea utilizabilităţii) interfeţei sistemului CAD Dietrich's, contribuind în mare măsură la creşterea gradului de satisfacţie a utilizatorului.

Parte dintre soluţiile din cadrul tezei, utilizate în cadrul dezvoltării interfeţei cu utilizatorul a sistemului CAD Dietrich's de proiectare a construcţiilor din lemn, constituie subiectul a 8 lucrări ştiinţifice (din care 6 ca unic autor) prezentate în cadrul unor conferinţe

internaţionale sau publicate în reviste de specialitate, parte sunt prezentate în premieră în cadrul tezei.

Lucrarea se bazează pe 150 de referinţe bibliografice din care 2 în calitate de co-autor şi 9 în calitate de unic sau prim autor.

Contribuţiile de ordin teoretic şi produsul program conceput şi dezvoltat, conferă lucrării un real caracter de originalitate şi aplicabilitate practică.

1.2. Prezentarea conţinutului tezei

Obiectivele propuse au condus la structurarea lucrării pe 6 capitole, al căror conţinut este prezentat în continuare.

Capitolul 2 prezintă consideraţii teoretice cu privire la interacţiunea om-calculator (IOC) în urma unei ample sinteze critice a tematicii unui important număr de lucrări de specialitate reprezentative. Se definesc principalele tendinţe care se fac simţite în domeniul realizării interfeţelor cu utilizatorul, fiind analizată şi reliefată pluridisciplinaritatea activităţilor implicate în domeniul IOC.

Este definit şi analizat conceptul de utilizabilitate. Se prezintă o sintetiză a schimbărilor de curente, sunt evidenţiate pe larg rolurile analizei, proiectării şi evaluării din cadrul dezvoltării sistemelor informatice. Prezentarea tipurilor de interfeţe cu utilizatorul oferă o imagine asupra modurilor de interacţiune om-calculator. De asemenea, o atenţie specială este acordată rezultatelor cercetărilor din cadrul ergonomiei software.

Ca suport pentru modelarea abstractă a interfeţelor cu utilizatorul, capitolul 3 sintetizează principalele caracteristici ale sistemelor cu evenimente discrete, se tratează trei tipuri principale de modele de interfaţă cu utilizatorul iar în finalul capitolului se modelează ca sistem cu evenimente discrete un proces de selecţie standard de obiecte din cadrul interfeţei cu utilizatorul pentru un sistem CAD, prezentat în cap. 5.

Capitolul 4 satisface nevoia de cunoaştere în profunzime a domeniului tehnologiilor software actuale în cadrul procesului de proiectare a unei interfeţe performante cu utilizatorul, în scopul de a identifica şi dezvolta acele elemente specifice care sunt cele mai adecvate aplicaţiilor propuse în cadrul tezei.

S-au selectat trei tipuri de tehnologii, acestea constituindu-se în sursele cele mai cuprinzătoare privind soluțiile tehnologice destinate realizării de interfețe cu utilizatorul de înaltă performanță, corespunzător obiectivului prioritar declarat al acestei lucrări:

- programarea orientată spre obiecte;
- programarea elementelor de interfață cu utilizatorul;
- programarea vizualizării utilizând biblioteca grafică OpenGL.

Paragraful 4.2 abordează, unde s-a considerat necesar până la nivel de detaliu, prin prisma necesităților aplicației dezvoltate în capitolul 5, avantajele programării orientate spre obiecte, prin realizarea unei analize paralele între proiectarea OO şi modul de organizare a realității. In scopul rezolvării problemelor ce implică sistemele complexe tratate în cadrul tezei, se propune adoptarea unei strategii care implică trei direcții de dezvoltare, *cunoaşterea sintaxei specifice* limbajului de programare utilizat, necesitatea cunoaşterii *conceptelor specifice* de baza ale unui limbaj OO şi necesitatea cunoaşterii *unor aspecte specifice implicite proiectării OO.*

Se justifică alegerea unui limbaj de programare orientat spre obiecte, sintetizând etapele esențiale de dezvoltare ale unui produs software: *analiza domeniului* prin identificarea, definirea şi clasificarea claselor ce simbolizează obiecte ale lumii reale, *analiza cerințelor sistemului* prin utilizarea conceptului de scenariu, *proiectarea sistemului* care realizează în fapt alocarea cerințelor sistemului către componentele hardware şi software, *analiza cerințelor software* a fiecărei componente software, *proiectarea software* prin generarea structurii software a aplicației şi crearea de componente arhitecturale software şi a interfețelor acestora, *implementarea* în care arhitectura software este creată utilizând facilitățile limbajului de programare ales. In cadrul acestei ultime etape se tratează în detaliu, cu exemple, aspecte concrete privind utilizarea template-urilor, utilizarea prototipizării şi implementarea unei componente reutilizabile.

Paragraful 4.3 tratează în detaliu, în cadrul interfețelor cu utilizatorul, probleme legate de utilizabilitate şi anume: *cardinalitatea* ca măsură a numărul entităților ce participă în cadrul unei relații, *reacția (feedback-ul)* ca mesaj informațional, *modelele explicite de utilizator* corespunzătoare modului de înțelegere a utilizatorului raportat la sistem, *suportul pentru planul de lucru* pentru captarea în timp a celor mai bune practici de utilizare, *suportul tranzacțiilor, răspunsul la erori, internaționalizarea şi localizarea, revocarea şi anularea*

acțiunilor, *compensarea tranzacțiilor*, *timpii de așteptare* și *redresarea după căderi ale sistemului*.

De asemenea se enunță *criterii privind realizarea interfețelor grafice cu utilizatorul* și 7 principii majore necesare unei afișări vizuale satisfăcătoare a informației. S-a analizat impactul asupra interfețelor cu utilizatorul a folosirii idiomurilor, a graficii si pictogramelor, a echilibrului dintre text și simboluri, a generării diagramelor de stare pentru interfețe, a amplasării și utilizării cutiilor de dialog, a navigării interfeței, a procesului de selecție, a manipulării obiectelor 3D, a satisfacerii tuturor categoriilor de utilizatori, a generării de rapoarte și tipărituri, a tratării excepțiilor precum și a mesajelor de notificare.

Dat fiind faptul că elementele de interfață sunt diferențiate în funcție de tipul de informație pe care îl transferă între utilizator și sistem, în cadrul procesului de proiectare s-au selectat și, după caz, s-au adaptat, dezvoltat și proiectat *elemente personalizate de interfață* corespunzătoare cerințelor produselor software dezvoltate în capitolul 5.

Paragraful 4.4 este dedicat descrierii modului de implementare a vizualizării OpenGL cu detalierea unor *algoritmi* ce realizează *vizualizarea* în ferestre multiple, *selecția* unui obiect pe ecran *prin utilizarea modului de selecție*, *evidențierea* pe ecran a unei entități grafice *prin utilizarea modului de reacție (feedback)*, *desenarea în mod wireframe și cu linii ascunse* pentru o mai bună percepție a reprezentării tridimensionale a modelului 3D proiectat, *descompunerea și triunghiularizarea poligoanelor complexe* pentru ca acestea să fie reprezentate corect pe ecran.

Capitolul 5, în totalitate original, prezintă *soluțiile de aplicare și implementare a metodelor și recomandărilor* de dezvoltare software propuse în capitolul anterior, în cadrul interfeței sistemului CAD Dietrich's de construcții din lemn realizată integral de către autor. In concret, sunt prezentate detaliat soluțiile de implemetare legate de:

- *dezvoltarea unei interfețe cu utilizatorul pentru Manager de Proiecte CAD* pentru eficientizarea operațiilor de gestiune a proiectelor și a pozițiilor de proiect;
- *dezvoltarea unor elemente personalizate de interfață* pentru obținerea unei flexibilități și eficiențe sporite a interfeței cu utilizatorul, activitate ce implică:
    - stabilirea de principii de funcționare a cutiilor de dialog,

- determinarea, proiectarea şi implementarea de cutii de dialog şi funcţii reutilizabile pentru introducerea datelor;
- determinarea, în cadrul interfeţei, a unui set de parametri configurabili de către utilizator;
- proiectarea şi implementarea de bare configurabile de instrumente;
- proiectarea unui model de bază pentru cutiile de dialog de operare cu fişiere;
- identificarea, proiectarea şi implementarea de elemente de interfaţă pentru funcţiile unui manager de biblioteci de modele;
- proiectarea şi implementarea de elemente de interfaţă cu utilizatorul privind o modalitate de grupare a prelucrărilor sub forma de punct-simboluri;
- proiectarea şi implementarea unei modalităţi de generare a listelor de materiale;

- *dezvoltarea unui proces standard de selecţie de obiecte* pentru uniformizarea procesului de selecţie a obiectelor atât din punctul de vedere al utilizatorului cât şi din cel al programatorului, beneficiindu-se de facilităţile intrinseci ale template-urilor;
- *dezvoltarea unei gestiuni speciale a unui grup de cutii de dialog* pentru reutilizarea codului existent;
- *dezvoltarea unui modul de vizualizare şi manipulare 3D a modelelor prin utilizarea bibliotecii grafice OpenGL,* prin integrarea într-o asemenea manieră încât codul existent suferă modificări minime pentru a funcţiona corect, cuprinzând următoarele activităţi:
  - proiectarea şi implementarea clasei care realizează toate operaţiile necesare vizualizării OpenGL, precum şi a structurilor de date pentru păstrarea informaţiei de model;
  - proiectarea şi implementarea într-un mod eficient şi transparent a procesului de selecţie standard;
  - sintetizarea a două proceduri, prin desenarea directă şi prin modul de randare cu reacţie al OpenGL, pentru evidenţierea obiectului selectat;
  - implementarea paşilor descrişi în cadrul algoritmilor de desenare cu linii ascunse şi de triunghiularizare propuşi în capitolul anterior.

Capitolul 6 sistematizează concluziile rezultate în urma realizării actualei teze, evidenţiindu-se contribuţiile originale aduse în dezvoltarea interfeţelor cu utilizatorul, care au

fost partajate în 2 categorii teoretice si aplicative, care la randul lor au fost subdivizate în contribuții aplicative metodologice si respectiv produs program.

1.2.1. Indexul principalelor abrevieri

AS – Analiza Sarcinii

CAD – Proiectare Asistată de Calculator (*Computer Aided Design*)

CAM – Manufacturare Asistată de Calculator (*Computer Aided Manufacturing*)

GKS – *Graphical Kernel System*

GOMS – scopuri, operatori, metode, reguli de selecție (*Goals. Operators, Methods, Selection Rules*)

GUI – Interfață Grafică cu Utilizatorul (*Graphical User Interface*)

IOC– Interacțiunea Om-Calculator

ISO – Organizația Internațională pentru Standardizare (*International Organization for Standardization*)

IT – Tehnologia Informatiei (*Information Technology*)

OO – Orientare spre Obiecte

PHIGS - *Programmer's Hierarchical Interactive Graphics Standard*

POO – Programarea Orientată spre Obiecte

SED – Sisteme cu Evenimente Discrete

SI – Sisteme Informatice

UI – Interfață cu Utilizatorul (*User Interface*)

UML – Limbaj Unificat de Modelare (*Unified Modeling Language*

Autorul îşi exprimă considerația profundă față de conducătorul ştiințific, D-l Prof. Dr. Ing. Octavian PROSTEAN căruia îi este extrem de recunoscător şi îi adresează vii mulțumiri pentru îndemnuri şi sfaturi, exigență, răbdare, solicitudine şi disponibilitate, toate oferite cu generozitate pe întreaga perioadă de pregătire a tezei. Autorul mulțumeşte în mod deosebit D-lui Prof. Dr. Ing. Toma Leonida DARGOMIR pentru bunăvoința de a fi acceptat să parcurgă lucrarea şi pentru aprecierile şi recomandările făcute. Autorul îşi exprimă profunda gratitudine regretatului Prof. Dr. Ing. Ioan MURESAN pentru impulsul şi sprijinul acordat în demararea acestui doctorat. De asemenea autorul aduce calde mulțumiri colegilor de la SC Computing Approach SRL şi în special D-nei Dr. Ing. Voichița MURESAN pentru ajutorul nemijlocit pe care l-au oferit ori de câte ori a fost necesar. Multe mulțumiri sunt adresate familiei şi în special părinților pentru sprijinul moral şi ajutorul oferit cu dragoste, fără de care finalizarea acestei teze la această dată nu ar fi fost posibilă.

2. Abordări moderne în concepția şi realizarea profesională a interfețelor cu utilizatorul

In prezent, sistemele de proiectare asistată de calculator (CAD) sunt utilizate pentru proiectare în numeroase domenii de activitate. Un obiectiv de o importanță majoră în cadrul proiectării a însuşi sistemului CAD o reprezintă interfața cu utilizatorul. Dezvoltarea unei interfețe cu utilizatorul pentru un sistem CAD implică găsirea de răspunsuri la o mulțime de întrebări în ceea ce priveşte dispozitivele şi tehnicile de introducere şi de afişare precum şi modurile de dialog ce le leagă. Toate aceste întrebări derivă din problema centrală de determinare a naturii fundamentale a interacțiunii om-calculator în cadrul procesului de proiectare suportat de către un sistem CAD.

Pentru a răspunde la aceste întrebări este necesară o sinteză a conceptelor şi problematicilor din cadrul domeniilor implicate în dezvoltarea interfețelor cu utilizatorul în general, cu o focalizare asupra elementelor specifice sistemelor CAD. Capitolul actual încearcă o astfel de sinteză prin studiul interfețelor om-calculator (IOC), a tipurilor de interfețe, a implicațiilor ergonomiei software, care va sta la baza elaborării metodologiilor de proiectare a interfețelor cu utilizatorul din cadrul capitolului următor.

2.1. Nevoia de studiu al IOC

Introducerea pe scară largă a calculatoarelor în societate a condus la schimbări fundamentale atât în modul de lucru cât şi asupra mediului de lucru [Lif1998]. Astfel interfața către proces s-a mutat de la maşini la ecranul calculatorului.

În momentul luării unei decizii este necesar accesul rapid la orice informație. Astăzi aproape toată informația este păstrată în imense sisteme informatice. Informațiile de care este nevoie trebuie să fie disponibile într-o formă adecvată pentru a ajuta utilizatorul în luarea rapidă de decizii optime. Proiectarea sistemelor pentru satisfacerea eficientă a utilizatorului în timpul luării deciziilor reprezintă un obiectiv major în cadrul IOC.

La introducerea calculatoarelor opțiunile legate de proiectare erau destul de limitate. Utilizatorul şi programatorul erau de obicei una şi aceeaşi persoană. Nu existau motive pentru a face sistemul util şi altor persoane [Lif1998] [Opp1989]. În prezent, majoritatea sistemelor de calcul sunt concepute pentru a fi folosite de utilizatori cu puține sau chiar fără abilități în operarea calculatoarelor. Interfețele grafice cu utilizatorul şi diferite tehnici de interacțiune au

mărit posibilitatea creării de sisteme informatice uşor de învățat şi eficiente în utilizarea curentă. Numărul opțiunilor de proiectare a crescut, prin urmare este necesar ca interfața să îndeplinească cerințele utilizatorilor care lucrează într-un anumit context.

Domeniul IOC este foarte extins. *Este necesară înțelegerea modului de gândire şi percepere a informației de către utilizator, modul de cooperare a oamenilor în timp ce lucrează, modul de realizare a echipamentelor şi programelor potențial satisfăcătoare pentru utilizator, modul în care se pot crea metode de dezvoltare a aplicațiilor, modurile de evaluare a aplicațiilor etc.*

IOC reprezintă o activitate multidisciplinară bazată pe cunoştințe obținute din mai multe domenii cum ar fi psihologia cognitivă, dezvoltarea de aplicații, ergonomie, analiza sistemelor, controlul proceselor şi design industrial.

2.1.1. Psihologia cognitivă

Proiectarea de interfețe cu utilizatorul este strâns dependentă de înțelegerea modului de gândire sau percepere a mediului. Creierul uman este mult mai complex decât calculatorul. Viziuni legate de calculatoare ce gândesc au existat în literatură de mult timp. Până în prezent, nu a fost posibilă realizarea acestei viziuni din cauza complexității problemei. Ceea ce se poate face este să se analizeze comportamentul uman în diferite medii. Unele dintre aceste rezultate au avut un impact major asupra progresului dezvoltării IOC.

Unul din cele mai recunoscute modele ale memoriei umane este descris de *teoria stadiu* [Gle1991]. Conform acestei teorii, toate ființele umane păstrează informația într-o memorie pe termen scurt (MTS) sau într-un sistem de memorie pe termen lung (MTL).
Sistemul uman MTS poate face față doar unei cantități mici de informație. Pe baza a numeroase studii, evidențele sugerează faptul că limita de întindere a MTS se reduce la şapte unități sau *calupuri* de memorie plus sau minus două. Calupurile se referă la unități de memorie ce rezultă prin înregistrarea de unități sau integrarea împreună de unități mai simple. Dimensiunea acestor calupuri diferă prin modul de structurare a informației (de exemplu, un număr de telefon poate fi memorat ca 4 9 1 8 0 7 (şase elemente) sau 491 807 (două elemente)). Durata de memorare în MTS a informației este de asemenea limitată. Uitarea în cadrul MTS poate fi datorată în mare parte îmbătrânirii sau substituirii [Wau1965]. Materialul păstrat în MTS îmbătrâneşte după aproximativ 15 secunde dacă el nu este procesat în continuare (ex. prin repetare sau altă strategie de memorare).

În contrast, capacitatea MTL este enormă. Informația stocată în cadrul MTL poate fi accesată cu ajutorul unor elemente activatoare cum ar fi cuvinte, miros, sunete. Un activator în sine este deseori suficient pentru aducerea aminte a unei cantități semnificative de informație; de exemplu un acord de pian poate activa cuvintele unei melodii.

Funcționalitatea sistemului de memorare are implicații semnificative în cadrul operarii cu calculatorul. Din cauza capacității limitate a MTS, toate informațiile necesare luării de decizii trebuie să fie afişate simultan pe ecran [Lin1991a]. Altfel, utilizatorul trebuie să păstreze informația în MTS sau să ia notițe pe hârtie. Acestea vor încetini munca utilizatorului şi vor conduce la o încărcare cognitivă inutilă.

Un model simplificat al procesării informației de către oameni implică faptul că procesele cognitive pot opera la nivele diferite de percepție [Ras1983] [Rea1987]. La un nivel cognitiv conştient este posibilă efectuarea unui singur proces în acelaşi timp. Acest nivel se aplică atât citirii şi înțelegerii informației semantice cât şi pentru rezolvarea problemelor complexe. La un nivel mai scăzut de înțelegere este posibilă efectuarea de operații în paralel şi în mare parte automat fără a se depune efort cognitiv. Efectuarea succesivă a unor operații frecvente poate duce la automatizarea lor pe nivele cognitive mai scăzute [Sch1977] [Shi1981]. Un exemplu în acest sens îl reprezintă conducerea unui automobil. La început poate fi dificilă schimbarea vitezelor, aprecierea circulației din apropiere şi menținerea maşinii pe şosea, simultan. Astfel, pentru un şofer începător aceste procese sunt tratate pe un nivel cognitiv mai ridicat. După o perioadă de antrenament, toate aceste procese sunt automatizate (adică pe un nivel cognitiv mai scăzut). Nivelul cognitiv ridicat poate fi utilizat pentru alte activități considerate mai importante. Acest model de procesare a informațiilor se poate constitui intr-un criteriu de proiectare a unui sistem de calcul. În consecință, interfața cu utilizatorul trebuie în aşa fel proiectată încât utilizatorul să o poată mânui automat (pe un nivel cognitiv mai scăzut) lăsând nivelele cognitive mai înalte pentru rezolvarea altor probleme legate de muncă [Nyg1996].

Psihologii cognitivi studiază, de asemenea, modul in care indivizii percep lumea înconjurătoare. Termenul de senzație se referă la detecția inițială de stimuli fizici, cum ar fi miros sau sunet. Percepția implică cunoaştere de nivel înalt în cadrul interpretării informației senzoriale (ex. caracterizarea mirosului). Un domeniu de mare interes este reprezentat de abilitatea de a detecta stimuli. Conform *teoriei detecției semnalului* [Tan1954], principalii factori care influențează această abilitate sunt magnitudinea semnalului stimulator, natura

acțiunii, aşteptările observatorului şi consecințele ce urmează recompensei şi pedepsei [Sol1991].

Recunoaşterea de modele este o altă temă care a atras interesul şi atenția psihologilor cognitivi. În acest cadru, un model se referă la „o compunere complexă de stimuli senzoriali pe care observatorul uman o poate recunoaşte ca fiind un membru al unei clase de obiecte" [Sol1991]. Au fost dezvoltate mai multe teorii asupra clasificării modelelor vizuale. Abordarea *Gestalt* accentuează faptul că noi percepem obiectele ca entități bine organizate şi nu ca părți separate [Mat1977]. O colecție de modele de puncte poate fi percepută în moduri diferite în funcție de criteriile de grupare ale observatorului.

Abordarea *Gestalt* prezintă un număr de reguli ale organizării percepției. Trei dintre acestea sunt *proximitatea, similaritatea* şi *închiderea* (Figura 2-1).

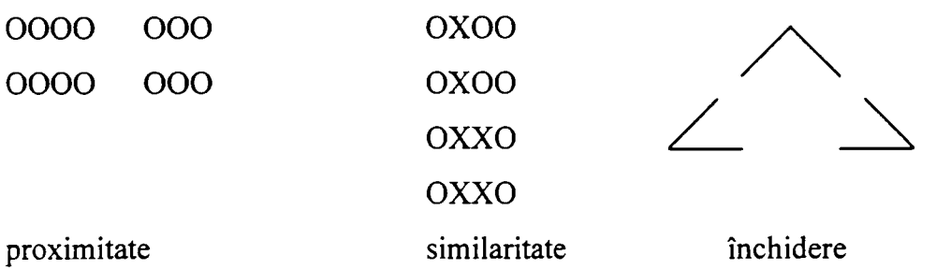


Figura 2-1. Exemple de legi Gestalt

Pentru a optimiza procesele de căutare şi citire, cunoaşterea modului de percepție este esențială în luarea de decizii asupra utilizării culorilor, tipurilor de litere, mărimii precum şi a modului de grupare a informației pe ecran. Utilizatorii experimentați decodifică rapid modelele semnificative [Nyg1996]. Dacă o colecție de obiecte are totdeauna aceeaşi localizare spațială pe ecran, se pot obține modele globale pentru ghidarea procesului de citire al utilizatorului. De asemenea este posibilă decodarea unui model dacă variabila are totdeauna aceeaşi culoare sau formă (conservarea anumitor atribute semnificative).

Nielsen [Nie1993] susține că termenul *user friendly* ("prietenos cu utilizatorul") a fost introdus în momentul în care dezvoltatorii de sisteme informatice au constatat că sistemele lor vor fi utilizate de persoane interesate mai mult de modul de interacțiune cu sistemul. Nielsen consideră că acest termen nu este corespunzător din mai multe motive. Unul din motive ar fi că „... *utilizatorii nu doresc maşini care să fie prietenoase cu ei, ei au nevoie de maşini care să nu îi încurce în desfăşurarea procesului de muncă*". Un alt motiv este subiectivitatea, în sensul că pentru unii maşinile pot fi „prietenoase", în timp ce pentru alții

ele pot fi „ostile". Ca alternativă, Nielsen propune termenul de *utilitate,* reflectând gradul în care sistemul poate fi util atingerii un anumit scop. Utilitatea poate fi divizată în două categorii: utilitate (propriu-zisă) și utilizabilitate [Gru1992].

Utilitatea exprimă gradul în care funcționalitatea necesară este inclusă în sistem (de ex. dacă toate uneltele necesare efectuării unei sarcini sunt prezente).

Utilizabilitatea reflectă cât de facil îi este utilizatorului folosirea acestei funcționalități.

Un alt criteriu important este măsura în care utilizatorul poate să interacționeze eficient cu sistemul fără eforturi mentale inutile cauzate de *probleme cognitive din cadrul mediului de lucru.* [Lin1991b]. Impedimentele, care sunt deseori cauzate de realizarea necorespunzătoare a interfațării om-calculator, pot sta la baza problemelor de sănătate somatică și mintală, procedurilor ineficiente de muncă, performanțelor slabe și neacceptării de către utilizator.

2.2. Dezvoltarea de sisteme utilizabile

In timp, s-au determinat metode de dezvoltare a sistemelor atât în cadrul Sistemelor Informatice (SI) cât și în domeniul IOC. Ehn și Löwgren [Ehn1997] au realizat o sintetizare a schimbărilor evolutive de curente în cadrul dezvoltării sistemelor, afirmând că atât în cadrul IOC cât și SI s-a pus accentul mai întâi pe metodele obiective, apoi pe cele sociale, pentru ca în prezent să se pună accentul pe metodele subiective.

Considerația *obiectivă* poate fi găsită în cadrul ingineriei utilizabilității (în IOC) și în ingineria software (la SI).

Ingineria utilizabilității, ca proces, constă de obicei, din trei pași principali: (1) analiza utilizatorului și a sarcinilor; (2) specificațiile de utilizabilitate, unde sunt identificate un număr de obiective măsurabile; și (3) procesul iterativ de proiectare, testare a utilizabilității și reproiectare până la atingerea obiectivelor [Goo1986].

Ingineria utilizabilității se ocupă în principal de performanțele utilizatorului în termeni de erori și timpi (rezultate măsurabile obiectiv).

Abordarea *ingineriei software* este similară metodelor sistematice din cadrul dezvoltărilor matematice și logice. Necesitățile informaționale într-o organizație sunt analizate ca fapte obiective. „O ipoteză des întâlnită în cadrul majorității abordărilor obiective SI în domeniul proiectării IT este aceea că utilizatorii trebuie să fie capabili de a descrie complet și explicit cerințele lor." [Ehn1997].

Ca o reacție la modul de abordare obiectivă, s-a dezvoltat o altă perspectivă asupra procesului de dezvoltare în SI care se concentrează asupra utilizatorilor și a influenței lor asupra sistemelor - *abordarea socială*. Conform acestei abordări, atât utilizatorii cât și sarcinile lor sunt reprezentați de probleme prea complexe pentru a putea fi înțelese și rezolvate numai de către dezvoltătorul de sisteme. Ca urmare, rezultă necesitatea ca și utilizatorii să participe la procesul de dezvoltare. Muller, Haslwanter și Dayton [Mul1997] au identificat următoarele trei motive pentru *proiectarea participativă*: primul este *democrația* (dreptul utilizatorului de a influența deciziile la locul lor de muncă), al doilea implică *eficiența*, *experiența* și *calitatea* (eficiența și calitatea software-ului sunt îmbunătățite prin implicarea utilizatorilor reali, ei fiind experți în domeniul lor) iar al treilea este reprezentat de *angajament* și *implicare* (este mult mai probabil ca utilizatorii finali să accepte sistemul dacă au luat parte la dezvoltarea lui).

Greenbaum și Kyng [Gre1991] definesc principiile procesului de *proiectare participativă*:

- Sistemele de calculatoare trebuie proiectate cu participarea deplină a utilizatorilor;
- Construirea unui sistem nou trebuie să îmbunătățească abilitățile de la locul de muncă și nu să le degradeze sau limiteze;
- Calculatoarele sunt unelte, în consecință ele trebuie proiectate pentru a fi controlate de oamenii care le folosesc;
- Calculatoarele trebuie să optimizeze productivitatea dar și calitatea muncii;
- Procesul de proiectare implică un caracter politic, generator de conflicte. Aceste conflicte trebuie luate în considerare pentru că ele sunt esențiale finalizării obiectivului;
- Procesul de proiectare trebuie să evidențieze problema utilizării calculatoarelor în contextul organizării muncii.

În cadrul IOC s-au dezvoltat câteva metode corespunzătoare proiectării participative. *Proiectarea contextuală* [Wix1990] este descrisă ca o metodă orientată pe client. Analiza muncii clientului este efectuată în contextul ei real prin *investigarea contextuală,* în acest cadru sunt observați și chestionați potențialii utilizatori finali [Hol1993]. Procesul de investigare contextuală este ghidat de trei mari principii: context, parteneriat și concentrare. El prezintă rezultatul înțelegerii naturii muncii utilizatorului prin investigații în timpul lucrului acestuia. Principiul parteneriatului pornește de la ipoteza că proiectanții pot lua la

cunoştință experiența de lucru a acestora şi modul de utilizare a uneltelor numai prin dialog cu utilizatorii. Alegerea focalizării asigură procesului de investigare contextuală colectarea informației de la utilizator. Valoarea informațiilor rezultă din interpretarea observațiilor şi prin dialogul cu utilizatorii. Interfețele sunt dezvoltate în co-proiectare cu utilizatorii şi sunt evaluate în investigări contextuale ulterioare. Abordarea contextuală a fost criticată datorită „lipsei unei metodologii sistematizate, a unui cadru conceptual, a unei căi explicite de abstractizare pornind de la experiențe particulare" [Car1989].

Factori cum ar fi timpul, organizarea, costurile şi tipul proiectului ce trebuie inițiat determină măsura în care utilizatorii vor fi implicați în proiect. În scopul de a evidenția momentele când utilizatorii şi dezvoltătorii ar putea avea oportunități de cooperare, Grudin [Gru1991] a identificat trei tipuri diferite de dezvoltare a proiectelor. În *dezvoltarea pe bază de contract* utilizatorii sunt cunoscuți înainte, dar organizarea dezvoltării este identificată după atribuirea contractului. *Dezvoltarea de produs*, cu caracter comercial presupune ca aplicației să i se facă reclamă înainte de a se cunoaşte utilizatorii. *Dezvoltarea în-casă* reprezintă cazul în care atât dezvoltătorii cât şi utilizatorii se cunosc de la început.

Ehn şi Löwgren [Ehn1997] evidențiază caracterul evolutiv al abordării, în prezent, de la cea socială la cea *subiectivă*. Focalizarea s-a mutat de la utilizator către proiectant şi competențele sale. Proiectarea este privită ca o artă. Cunoştințele necesare realizării unui proiect util izvorăsc din experiență şi îndemânare. Mai mult, conceptul utilizabilității este conectat de noțiunile: noutate, estetică şi emoție. Un sistem informațional trebuie să fie plăcut, prezentabil, satisfăcător şi special pentru a deveni un produs popular. În acest context, piața este cea care joacă rolul decisiv.

În studiul metodelor de dezvoltare a sistemelor trebuie luat în considerare întreg lanțul analiză, proiectare, evaluare.

2.2.1. Analiza

În dezvoltarea sistemelor informatice este necesară înțelegerea modului în care utilizatorul execută munca sa şi interacționează cu mediul. În proiectele mari, unde sunt implicate mai multe persoane, este necesar ca cerințele utilizatorilor să fie descrise în aşa fel încât toți participanții să le poată înțelege. Documentația devine şi mai importantă atunci când oamenii implicați în faza de modelare nu sunt aceiaşi cu cei implicați în proiectarea

sistemului. Modelarea muncii utilizatorului constă în mod normal în două activități majore: de analiză şi de documentare [Lif1998].

2.2.1.1 Analiza sarcinii

Analiza sarcinii (AS) este legată, în general, de acțiunile întreprinse de utilizatori pentru a-şi atinge obiectivele [Pre1994]. Scopul AS este acela de a analiza munca utilizatorului şi descrierea ei în funcție de sarcini. *Analiza ierarhica a sarcinilor* (AIS) este o metodă de descompunere a sarcinilor în subsarcini şi operații [She1989]. Acestea sunt reprezentate grafic prin diagrame. Scopul este de a descrie sarcinile în termenii unei ierarhii de operații şi planuri. AIS este utilizată în identificarea paşilor necesari efectuării sarcinii. *Analiza cognitivă a sarcinii*, pe de altă parte, este mai preocupată de reprezentarea cunoştințelor pe care le au utilizatorii, sau este necesar să le aibă, pentru a îndeplini o sarcină. Aceasta se poate realiza cu ajutorul modelării *scop, operatori, metode, reguli de selecție (GOMS) (v. 2.4.1),* unde sunt specificate obiectivele, operatorii, metodele şi regulile de selecție [Car1983]. *Obiectivele* reprezintă acțiunile pe care utilizatorul trebuie să le îndeplinească; un *operator* este o acțiune derulată pentru îndeplinirea unui obiectiv; *metodele* sunt secvențele de operatori necesare îndeplinirii obiectivelor, iar *regulile de selecție* reprezintă cunoaşterea de către utilizator a metodei ce trebuie utilizată. *Gramatica sarcină-acțiune* (GSA) utilizează o notație de reguli pentru a specifica cunoştințele şi interacțiunea utilizatorului [Pay1989].

Există diferite tehnici de aplicare a AS [Jef1997]. O tehnică des utilizată în strângerea de date asupra sarcinii este *intervievarea* utilizatorilor ce lucrează într-un anumit domeniu. Interviurile cu utilizatorii se pot conduce individual sau în grup şi sunt importante atunci când este nevoie de o înțelegere mai profundă. Deşi aprofundarea înțelegerii utilizând această metodă este mai dificilă, chestionarele ar putea fi mai convenabile atunci când scopul este acela de a aduna informație de la o gamă largă de utilizatori. Combinarea ambelor metode poate conduce la rezultate optimizate. Numărul de persoane cărora le pot fi distribuite chestionare este mult mai mare comparativ cu numărul celor pasibili de a fi intervievați.

O altă metodă este de a *observa* în detaliu utilizatorii ce efectuează sarcina. Observațiilor pot conduce la informații pe care altfel utilizatorul le-ar putea omite [Sch1977].

Benyon [Ben1992] localizează principalele probleme care apar în cadrul metodelor de AS:

- **Dependența de dispozitiv**. AS nu este capabilă de a fi independentă de dispozitivul utilizat pentru îndeplinirea sarcinii. Dacă AS este utilizată în primele faze ale dezvoltării, există un mare risc de includere în noul sistem a practicilor curente.
- **Obiecte**. Majoritatea tehnicilor de AS se referă la obiecte. Aceste obiecte sunt dependente de dispozitivul fizic, definirea lor fiind relativă. Entitatea ar putea fi noțiunea adecvată.
- **Ierarhia**. AS utilizează reprezentarea ierarhică. O ierarhie este eficientă dacă sistemul poate fi descompus în mod obiectiv. Subiectivitatea utilizatorilor sistemului ar putea ridica probleme în reprezentarea ierarhică.
- **Gramatici**. Majoritatea tehnicilor AS utilizează gramaticile în locul reprezentărilor grafice, dar gramaticile nu sunt orientate pe utilizator pentru că utilizatorului îi este greu să le înțeleagă.

Benyon [Ben1992] subliniază că în cadrul analizei sistemelor există deja metode bine definite pentru analiza structurată. Prin urmare, cercetătorii IOC ar trebui să se concentreze asupra uneltelor de modelare pentru a completa metodele AS existente. În cadrul AS pot fi dezvoltate o serie de modele pentru a descrie un viitor sistem, fiecare model reprezentând o anumită perspectivă asupra sistemului:

- **Modelul relațional de date**. Acest model descrie inter-relaționarea informației din cadrul unui sistem. Modelul relațional constă din două concepte de bază: domenii şi relații. Domeniile sunt seturi de valori din cadrul cărora poate fi reprezentată valoarea unui element. Relațiile pot fi de tip tabelar. Fiecare relație poate avea un număr de atribute, de ex. relația *persoană* poate avea ca atribute codul numeric personal şi numele.
- **Modelul entitate-relație (E-R)** Modelul E-R este o abordare de sus în jos a modelării informației, demarează cu concepte generale, cum ar fi entități şi relații, şi apoi completează detaliile până când se atinge un nivel corespunzător. O entitate este definită ca fiind un grup de elemente informaționale, care pot fi identificate în sistemul analizat şi având anumite relații între ele. Modelul E-R este o reprezentare grafică a sistemului, pe baza căreia se pot purta discuții cu utilizatorii finali. Modelul relațional de informații este util în special în dezvoltările bazelor de date. Aceste două modele pot fi utilizate în conjuncție.
- **Diagrama vehiculării informațiilor** (DVI). Această diagramă descrie fluxurile informațiilor din cadrul unei organizații, de ex. de la persoană la persoană [DeM1978]. DVI este, în esență, un proiect logic al noului sistem, specificând

funcțiile necesare. Necesitățile de bază ale modelului procesului sunt intrările, ieşirile, procesele şi dispozitivele de păstrare a informației. Procesele, fiind activate de evenimente, preiau informația sub formă de date de intrare şi produc informație ca date de ieşire.

2.2.1.2 Metode orientate pe funcții sau pe obiecte

Metodele pentru dezvoltarea sistemelor pot fi clasificate în funcție de orientarea pe funcții sau pe obiecte.

Conform *abordării orientate pe funcții*, funcțiile şi datele nu sunt legate între ele. Datele sunt tratate ca un suport pasiv al informației, iar funcțiile sunt părțile active. Un sistem modelat conform acestei abordări este divizat în funcții, iar datele sunt transmise între funcții. Unul din dezavantaje este acela că toate funcțiile trebuie să cunoască felul cum sunt structurate datele [Jac1992]. Dacă o structură este schimbată, toate funcțiile ce operează cu informația respectivă trebuie rescrise.

În cazul unei *abordări orientate pe obiecte*, funcțiile şi datele sunt integrate. În cadrul sistemului modelat se pot distinge diferite obiecte, majoritatea având corespondent în lumea reală. Fiecare obiect posedă atribute ce păstrează informația şi operații ce descriu comportamente, *putând fi descris ca o cutie neagră cu intrări şi ieşiri (black-box), deci practic poate fi interpretat şi abordat sistemic*. În cazul interacțiunii a două obiecte nu este necesar să se cunoască structura internă a datelor astfel că schimbările în sistem tind să fie locale.

Modelarea orientată pe obiecte a devenit în ultimul timp tot mai utilizată. Cea mai utilizată tehnică de modelare este *Unified Modelling Language (UML* - limbaj de modelare unificat), [Boo1997]. UML este o versiune unificată a trei tehnici diferite de modelare orientată pe obiecte. Modelul este documentat în funcție de patru diagrame: *cazuri de utilizare, clasa, comportament şi implementare*. În cadrul UML, cerințele care stau la baza funcționării sistemului sunt descrise în funcție de *cazuri de utilizare* şi *actori* [Jac1992]. Există actori diferiți (umani şi non-umani) care schimbă informații cu sistemul şi reprezintă latura de interacțiune cu sistemul. Un actor non-uman este, de exemplu, un alt sistem de calcul, actorii nu sunt considerați parte a sistemului, astfel că nu sunt descrişi în detaliu. În UML, un actor este privit ca un fel de clasă unde fiecare instanță a unei asemenea clase reprezintă un utilizator,. care poate juca rolul mai multor actori (ex. manager şi

administrator). Un utilizator îşi efectuează activitatea printr-o succesiune de operații asupra sistemului, definind un caz de utilizare. Fiecare caz de utilizare reprezintă un anumit mod de interacțiune cu sistemul. Un caz de utilizare este, de asemenea, privit ca o clasă, astfel că fiecare scenariu executat de utilizator poate fi descris ca o instanță a unui caz de utilizare. Instanța există atât timp cât cazul de utilizare este activ.

Descrierea necesităților în funcție de actori şi cazuri de utilizare ajută definirea modului de comunicare între obiectele implicate din cadrul sistemului [Lif1998]. Utilizatorii devin implicați din primele etape şi pot descrie activitatea lor într-o terminologie ce poate fi adoptată atât de utilizatorii cât şi de dezvoltătorii sistemului.

În general, metodele de analiză a sistemelor sunt adecvate dezvoltării mai multor componente ale sistemului informațional, dar nu îndeplinesc condițiile necesare impuse de proiectantul interfeței cu utilizatorul. Totuşi, aceste metode sugerează proiectantului să creeze un proiect în care fiecare funcție sau caz de utilizare este reprezentat de către o fereastră pe ecran. De obicei utilizatorul, pentru a îndeplini o sarcină, trebuie să interacționeze cu mai multe ferestre, rezultând o interfață fragmentată, cu un număr prea mare de ferestre. Pentru proiectarea unei interfețe utilizabile este necesară efectuarea unor modelări adiționale.

2.2.2. Proiectarea

De obicei a proiecta înseamnă a crea (a descrie) o nouă entitate, cum ar fi un sistem sau o bază de date. În cadrul acestui capitol, procesul de proiectare semnifică realizarea interfeței cu utilizatorul.

Abordări ale proiectării

Wallace şi Anderson [Wal1993] au identificat patru moduri majore de abordare a proiectării interfețelor: măiestria, ingineria cognitivă, tehnologia şi ingineria software avansată. Prin *măiestrie* fiecare proiect este unic, făcând imposibilă utilizarea metodologiilor generale la proiectarea interfeței. În schimb, aceasta presupune anumite abilități ale factorilor umani. Această abordare porneşte de la ipoteza că un proiect poate fi produsul doar al unui „bun" proiectant. *Ingineria cognitivă* reprezintă o încercare de aplicare a teoriilor procesării informației şi a rezolvării problemelor la proiectarea interfețelor. Un exemplu este modelul Kestroke-Level [Car1983], care intenționează să cuantifice proprietățile acțiunilor utilizatorilor (cum ar fi timpul necesar deplasării cursorului sau introducerea unei litere la tastatură) iar parametri rezultați să fie luați în considerare la proiectare. Prin punerea la dispoziție de unelte ajutătoare procesului de proiectare, *tehnologii* urmăresc eliberarea

programatorilor de sarcina dificilă şi consumatoare de timp a proiectării interfeței. În sfârşit, abordarea *ingineriei software avansate*, pentru a ajuta procesul de proiectare, presupune ca metodele de analiză a sarcinilor să fie introduse ca o extensie a metodelor de inginerie software.

Există şi metode de analiză a sistemelor destinate dezvoltării anumitor părți ale sistemului de calcul. Totuşi, este nevoie de o metodologie complementară metodelor de analiză a sistemelor astfel încât ea să fie capabilă să conducă dezvoltarea către o proiectare a unei interfețe utilizabile. Într-o anumită măsură, proiectarea este un proces creativ astfel că, competența proiectantului va avea un impact critic asupra interfeței. Totuşi, pentru a fi posibilă luarea deciziilor corecte, proiectantul trebuie să primească un model cât mai complet care să descrie necesitățile utilizatorului pentru interfață.

Luarea deciziilor de proiectare

Proiectarea este în principal o problemă de optimizare a interfeței cu utilizatorul pe baza cerințelor sistemului informatic. Utilizatorii sunt experți în domeniu şi au unele aşteptări asupra modului de lucru cu noul sistem de calcul (Figura 2-2). Natura aşteptărilor este determinată în general în timpul analizei sistemului şi a sarcinii. Grupuri diferite de utilizatori au cerințe diferite ce trebuie îndeplinite în cadrul proiectului. Câteva din trăsăturile relevante pentru utilizator care trebuie luate în considerare sunt *cunoştințele, abilitățile, experiența, pregătirea profesională şi atributele fizice* [ISO1995]. În funcție de ghidurile de stil, *regulile şi recomandările* ce descriu modul de prezentare şi funcționare a interfeței (cum ar fi Windows) sunt esențiale. În plus, unele companii posedă un standard de corporație ce restricționează aranjamentul şi comportamentul interfeței. *Mediul tehnic* limitează şi el spațiul de proiectare. Dimensiunea ecranului şi rezoluția influențează în mod direct cantitatea de informație posibil de afişat simultan pe ecran. Unealta de construcție (de exemplu pentru generarea de prototipuri şi implementarea interfeței) poate impune restricții asupra utilizării facilităților grafice. Pe baza tuturor acestor interacțiuni, sarcina proiectantului este de a optimiza interfața pentru a găsi soluția optimă în sprijinul utilizatorului în timpul lucrului.

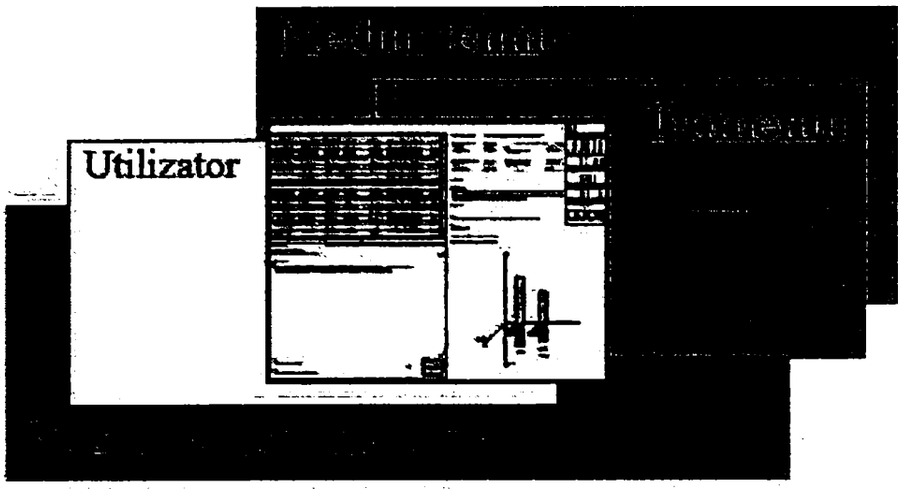


Figura 2-2. Exemplu de proiectare a interfeței prin optimizare în funcție de diferitele cerințe

Interfețele grafice cu utilizatorul, implicând o gamă largă de culori, caractere speciale, reprezentări 3D, s-au răspândit foarte mult în ultimii ani. De asemenea, gradele de liberate s-au înmulțit rapid. Prin utilizarea graficii în interfață a devenit posibilă crearea de interfețe bogate, distincte şi eficiente ce sunt uşor de învățat şi utilizat. Pe de altă parte însă, s-a mărit şi riscul de a lua decizii greşite de proiectare. Utilizarea incorectă a uneltelor grafice poate conduce la situația în care interfețele grafice să fie mai puțin eficiente decât vechile interfețe alfanumerice [Nie1993]. Astfel, extinderea spațiului de proiectare impune proiectantului condiții mai severe.

Susținerea procesului de proiectare

Optimizarea interfeței cu utilizatorul este o sarcină deosebit de complexă, iar o descriere exactă a modului de realizare a acesteia nu este posibilă, întrucât nu pot fi identificate soluții general valabile. Ceea ce este optim într-un anumit context poate fi dezastruos în altul. Există totuşi metode ce pot facilita acest proces cum ar fi *Design rationale* [Mac1991]. Scopul acesteia este de a susține procesul de decizie din cadrul proiectării şi a documentării acestor decizii. Problemele de proiectare, opțiunile şi criteriile sunt specificate pentru fiecare decizie. O problemă tipică de proiectare ar putea fi stabilirea modalităților de selecție a diferitelor operații în cadrul interfeței, unde pentru fiecare problemă sunt identificate diferite opțiuni de proiectare (ex. selectare din meniu sau cu ajutorul butoanelor). Fiecare opțiune este documentată împreună cu o listă de criterii astfel încât să poată fi selectată cea mai benefică opțiune de proiectare. *Design rationale* poate fi utilă în structurarea informațiilor relevante în luarea deciziilor în cursul proiectării, având însă dezavantajul de a fi mare consumatoare de timp.

Proiectarea poate fi îmbunătățită semnificativ prin realizarea continuă de prototipuri care să fie testate de utilizatorii finali într-o manieră iterativă. Abordarea *prototipurilor* permite proiectantului să testeze soluții de proiectare încă din primele faze. Dacă sunt detectate probleme potențiale legate de utilizabilitate încă din primele faze, ele pot fi evitate mai uşor în final. Pornind de la ideea că realizarea de prototipuri este în esență o abordare empirică a proiectării [Joh1992] rezultă logic că, pentru a asigura o eficiență satisfăcătoare, faza de realizare a prototipurilor trebuie să fie precedată de o analiză profundă.

Shneiderman [Shn1992] a sugerat un număr de elemente măsurabile, ale căror valori este neapărat necesar a fi luate în considerare în cadrul proiectării unui sistem: *timpul de învățare, viteza performanței, rata erorilor utilizatorilor, întârzierile de timp* şi *satisfacția subiectivă*. Shneiderman susține că este dificil pentru un proiectant să atingă valori optime ale tuturor acestor parametri astfel încât el este nevoit să facă compromisuri. Dacă rata erorilor se impune a fi menținută la un nivel scăzut, atunci în mod necesar este sacrificată performanța/viteza. Proiectantul trebuie să fie conştient de imposibilitatea evitării acestor compromisuri şi că trebuie să se concentreze asupra priorității anumitor obiective.

Un alt sprijin în procesul de proiectare este furnizat prin utilizarea *ghidurilor şi a euristicii*. Foley [Fol1995] este de părere ca în timpul procesului de proiectare să fie luate în considerare următoarele principii: *consecvența,reacția (feedback), minimizarea posibilităților de eroare, posibilitatea de revenire în urma unei erori, prezența mai multor nivele de experiență (abilități)* şi *minimizarea cantității de informație necesar de a fi memorată*. Aceste principii pot funcționa ca o listă de verificare pentru a ghida proiectantul către optimizarea opțiunilor de proiectare. Totuşi, conflictele ce pot apărea între diferite principii trebuie luate în considerare.

O interfață cu utilizatorul se poate baza pe ghiduri de stil şi standarde. Un ghid de stil descrie funcționalitatea şi aranjamentul diferitelor elemente de interfață. În mod normal, un ghid de stil este o caracteristică generală cu suport limitat în ceea ce priveşte dezvoltarea într-un anumit domeniu. Un ghid de stil situat pe un nivel mai evoluat, unde este prezentă şi cunoaşterea domeniului, poate fi mai detaliat şi mai eficient pentru procesul de proiectare [Gul1993] [Gul1996]. Trăsături importante ale *ghidului de stil specific unui domeniu* sunt elementele compozite de interfață corespunzătoare structurilor mai complexe de informație

din domeniu. Ghidurile de stil și standarde sunt necesare, dar uneori acestea pot restricționa flexibilitatea proiectantului în momentul realizării interfeței [Ols1993] [Gru1989].

Proiectare participativă

Proiectarea interfeței nu depinde doar de corectitudinea alegerii metodelor utilizate, ci și de competența celor ce iau decizii în cadrul proiectării. În cazul unor proiecte de dezvoltare de sisteme proprietare cei responsabili cu proiectarea interfeței sunt programatorii. Aceştia, chiar dacă sunt interesați de domeniul proiectării interfețelor, arareori au și competența corespunzătoare sau/și timpul necesar proiectării interfețelor utilizabile [Gre1991]. Soluția optimă la această problemă poate fi abordarea *proiectării participative* întrucât utilizatorii sunt experți în domeniul în care va fi utilizat sistemul chiar dacă nu sunt experți și în proiectare și utilizabilitate.

Pentru a realiza o interfață cu utilizatorul, pe lângă capacitățile artistice, sunt necesare cunoştințe din domeniul ingineriei software, psihologiei cognitive și utilizabilitate. Pentru că persoana responsabilă cu proiectarea trebuie să fie bine orientată într-o multitudine de domenii, ar fi de dorit ca proiectarea interfeței să fie realizată de un grup de *experți în proiectare*. O cale de îmbunătățire a abilităților individuale de realizare a unui proiect bun este de a crea cât multe interfețe și apoi de a analiza utilizabilitatea lor [Gre1991] [Lif1998]. Rolul experienței este foarte important pentru că proiectarea dezvoltării sistemelor necesită o cantitate considerabilă de timp, dar programatorii trebuie să acorde prioritate programării, rămânându-le puțin timp dedicat acumulării experienței de proiectare. Doar o persoană care s-ar putea concentra exclusiv asupra proiectării și testării utilizabilității ar avea posibilitatea acumulării unei asemenea experiențe.

Proiectantul interfeței cu utilizatorul poate fi comparat cu un arhitect. Arhitectul este în parte inginer, în parte artist. El trebuie să creeze o structură care este în același timp funcțională și armonioasă, care să țină seama de anumite criterii cum ar fi utilizarea de elemente standard, securitate sau menținerea într-un anumit buget. Proiectantul trebuie considerat ca făcând parte din dezvoltătorii de sisteme. Uneori ar trebui implicați proiectanți din domenii diferite, cum ar fi cel de proiectare a interfețelor (sau de interacțiune) și designer grafic.

Dependența de utilizator

Nielsen [Nie1993] identifică trei dimensiuni în care experiența utilizatorilor diferă: cunoştințe generare asupra calculatoarelor, experiență în utilizarea unui anumit sistem și

respectiv cunoştinţe asupra domeniului, ce au un impact important asupra proiectării interfeței cu utilizatorul. Un utilizator cu experiență în operarea calculatoarelor lucrează cu sisteme într-un mod diferit comparativ cu cei fără experiență. Este de aşteptat ca un expert de sistem să recurgă la „scurtături" de la tastatură, în timp ce un începător interacționează prin intermediul mouse-ului. Unui expert în domeniu îi este familiar limbajul domeniului, astfel că el poate avea afişată pe ecran o densitate mare de informație legată de munca sa.

Utilizatorii profesionişti sunt experți în domeniu. Aceştia utilizează sistemul informațional pentru interacțiune profesională, mai multe ore pe zi. La proiectarea unei interfețe pentru un astfel de utilizator, este mult mai important ca sistemul să fie eficient în utilizarea zilnică decât să fie uşor de învățat [Lif1998]. Astfel, utilizatorii trebuie să poată configura interfața conform experienței, paragraful 4.3.2 tratând în detaliu acest aspect.

Estetica în proiectarea interfețelor cu utilizatorul

Cel mai mare efort în IOC s-a depus pentru a spori eficiența şi utilitatea unui sistem. Totuşi, sistemul nu trebuie să fie doar funcțional, el trebuie să fie în acelaşi timp atractiv şi estetic [Shn1992]. Un produs trebuie să "transmită" clientului ce fel de produs este şi la ce se poate aştepta clientul de la el. Este important ca domeniul căruia îi este destinat sistemul să fie uşor de recunoscut (să se poată spune dacă sistemul va fi utilizat într-o bancă sau un spital). Modul de prezentare a unui produs poate fi esențial pentru ca acesta să fie confortabil în utilizare, devenind totodată un criteriu decisiv în competiția cu produsele rivale.

Majoritatea trăsăturilor constructive ce privesc estetica unei interfețe au fost dezvoltate în cadrul designului grafic. Specificațiile de acest tip nu pot fi luate în considerare la modul general pentru că ele depind de curente şi de modă. Totuşi, există câteva reguli ce se aplică, cum ar fi modul de utilizare a spațiului limitat pentru a crea o interfață echilibrată şi atractivă utilizatorului [Lif1998].

2.2.3. Evaluarea

Metodele de evaluare a interfețelor cu utilizatorul se divid în *metode de testare* a utilizabilității, unde sunt implicați utilizatorii, şi *metode de inspecție* a utilizabilității, unde utilizatorii *nu* sunt implicați.

- Metode de testare

O metodă tradițională pentru testarea de către utilizatori este *analiza performanței,* scopul acesteia fiind analiza măsurii în care s-a atins obiectivul utilizabilității. Performanța este analizată de obicei cu ajutorul unui grup de utilizatori de test ce efectuează un set predefinit de sarcini, colectându-se date cuantificabile asupra erorilor şi timpilor, uşurând compararea de diferite soluții de proiectare. Din păcate, în majoritatea proiectelor de dezvoltare a sistemelor nu este suficient timp, nu sunt suficienți bani sau lipseşte experiența de laborator pentru a se recurge la o asemenea metodă [Nie1993]. Un alt dezavantaj al testelor de laborator este aceea că, la începutul procesului de proiectare, testarea este dificilă în condițiile în care nu este încă disponibil un prototip funcționabil şi o bază de date suficient de completă. Alte inconveniente legate de procesul testării utilizabilității pot fi întâlnite în culegerea de date, probleme metodologice în planificare, validitatea şi siguranța datelor obținute [Hol1991]. În plus, evaluarea zilnică a eficienței necesită utilizatori abili.

În cazul *gândirii cu voce tare*, utilizatorii îşi exprimă verbal gândurile în timpul utilizării sistemelor [Lew1982]. Prin acest test, utilizatorii dau posibilitatea evaluatorului să înțeleagă modul de vizualizare a sistemului de calcul. Acest experiment identifică ideile greşite ale utilizatorilor despre sistem şi nu este costisitor. Utilitatea acestuia este probată atunci când este aplicat de proiectantul interfeței întrucât reacția vine direct de la utilizatori [Jør1990]. Dezavantajul metodei rezidă în situațiile în care utilizatorii nu-şi exprimă verbal gândurile fie pentru că nu sunt obişnuiți să o facă fie că sunt atât de specializați încât parțial activitatea lor este efectuată automat [Sch1977] [Shi1981].

Chestionarele sunt utile în special în cazurile în care se monitorizează satisfacții subiective şi posibile îndoieli ale utilizatorului [Nie1993]. Chestionarele pot fi uşor distribuite unui număr mare de utilizatori şi în plus este şi o metodă ieftină de sondaj. Totuşi, este foarte dificilă obținerea de rezultate obiective prin utilizarea chestionarelor, pentru că răspunsurile utilizatorilor se bazează pe ceea ce cred ei că fac şi nu pe ceea ce fac efectiv.

O metodă ce include utilizatorii, dezvoltătorii şi experții de utilizabililate, ce poate fi aplicată în etapele inițiale ale procesului de proiectare este *parcurgerea pluralistă* [Bia1991]. Reprezentanți ai celor trei categorii se întâlnesc şi discută despre problemele de utilizabilitate ce sunt asociate cu elementele de dialog din cadrul diferiților paşi ai scenariilor. Cazul parcurgerii pluraliste urmăreşte reacția utilizatorilor aflați în diferite situații. Parcurgerea pluralistă este o metodă eficientă în evaluarea uşurinței de învățare a unei interfețe cu

utilizatorul, dar nu este o metodă adecvată a evaluării eficienței în utilizarea zilnică, pentru că utilizatorii nu pot prezice cum se va interacționa cu sistemul atunci când vor avea experiență.

- Metode de inspecție

Există mai multe metode de inspecție. Una din aceste metode este *parcurgerea cognitivă* [Pol1992]. Cu această metodă, un evaluator examinează fiecare acțiune dintr-o cale de soluționare şi încearcă să creeze o descriere credibilă a motivelor ce-l determină pe utilizator să aleagă o anumită acțiune. Descrierea se bazează pe anumite presupuneri asupra experienței, cunoştințelor şi obiectivelor utilizatorului şi pe înțelegerea procesului de rezolvare a problemelor, ce permite utilizatorului să determine acțiunea optimă. Parcurgerea cognitivă este o metodă de inspecție ce se concentrează asupra evaluării unui proiect pentru uşurarea învățării, mai ales prin explorare. Este mult mai dificilă evaluarea eficienței în cazul utilizării zilnice. Problemele legate de conținutul interfeței sunt rareori identificate, datorită cunoaşterii limitate în domeniu a evaluatorului.

O altă metodă de evaluare este *evaluarea euristică* [Nie1990]. Evaluatorul utilizează un set de indicații (euristice) comparându-l cu interfața. Euristic se creează o listă de verificare pe care evaluatorul/utilizatorul o foloseşte în timpul lucrului. Prin evaluarea euristică este posibilă identificarea multor probleme de utilizabilitate şi este posibilă o evaluare timpurie în cadrul procesului de proiectare. Metoda este dificilă pentru utilizatorii finali fără cunoştințe în domeniul IOC. Metodele euristice nu sunt adaptate identificării problemelor de utilizabilitate legate de eficiența în utilizarea zilnică, totuşi, evaluarea euristică poate fi utilă la evaluarea stilului (prezentarea) interfeței.

2.3. Interfețe cu utilizatorul

Gradul de utilizabilitate al unui program este determinat în principal de calitatea interfeței cu utilizatorul. Aceasta permite comunicarea dintre utilizator şi program care se realizează în două sensuri şi în două moduri specifice: *grafic* sau *textual*.

Paragraful de față este dedicat cu precădere prezentării tipurilor de interfețe cu utilizatorul, propunându-se o clasificare a lor. De asemenea este evidențiat rolul important al utilizării de elemente speciale în creşterea eficienței proiectării asistate de calculator.

Interfața cu utilizatorul sau dialogul om-calculator reprezintă singura modalitate de comunicare între utilizatori şi sistemul CAD/CAM, conținând o colecție de comenzi pe care utilizatorul le poate folosi pentru a interacționa cu un sistem particular CAD/CAM.

Limbajul interfeței trebuie să fie suficient de *simplu* pentru a fi înțeles de utilizator. De asemenea, trebuie să fie *eficient* şi *complet*, şi trebuie să aibă o *gramatică naturală*, adică un număr minim de reguli uşor de urmat. Acest lucru conduce la minimizarea perioadei de instruire a utilizatorului şi permite acestuia să se concentreze pe problemele de rezolvat, permițându-i să *revină* în situația anterioară atunci când comite greşeli [Sav1997] [Gul1996].

Independent de tipul interfeței cu utilizatorul, structura generică a unei comenzi CAD/CAM constă din două părți: o parte de comunicare cu utilizatorul şi o parte de comunicare cu baza de date.

Partea de *comunicare cu utilizatorul* include dialogul purtat de utilizator pentru realizarea unor scopuri anumite.

Partea de *comunicare cu baza de date* include introducerea sau extragerea de date grafice din baza de date [Gul1996].

Proiectantul interfeței cu utilizatorul a unui sistem CAD/CAM trebuie să ia în considerare factori precum: manipularea datelor, caracteristicile bazei de date, factori umani, structura meniurilor [Gar1991]. Programele interactive au multiple secțiuni, executabile practic într-o infinitate de combinații, deci asigurarea integrității datelor şi a programului este complexă.

Fluxul de interacțiune între utilizatorul uman şi maşină poate fi divizat în unități individuale ce pot fi referite ca *tranzacții*. Există două forme similare de tranzacții, prezentate în figura 2-3.

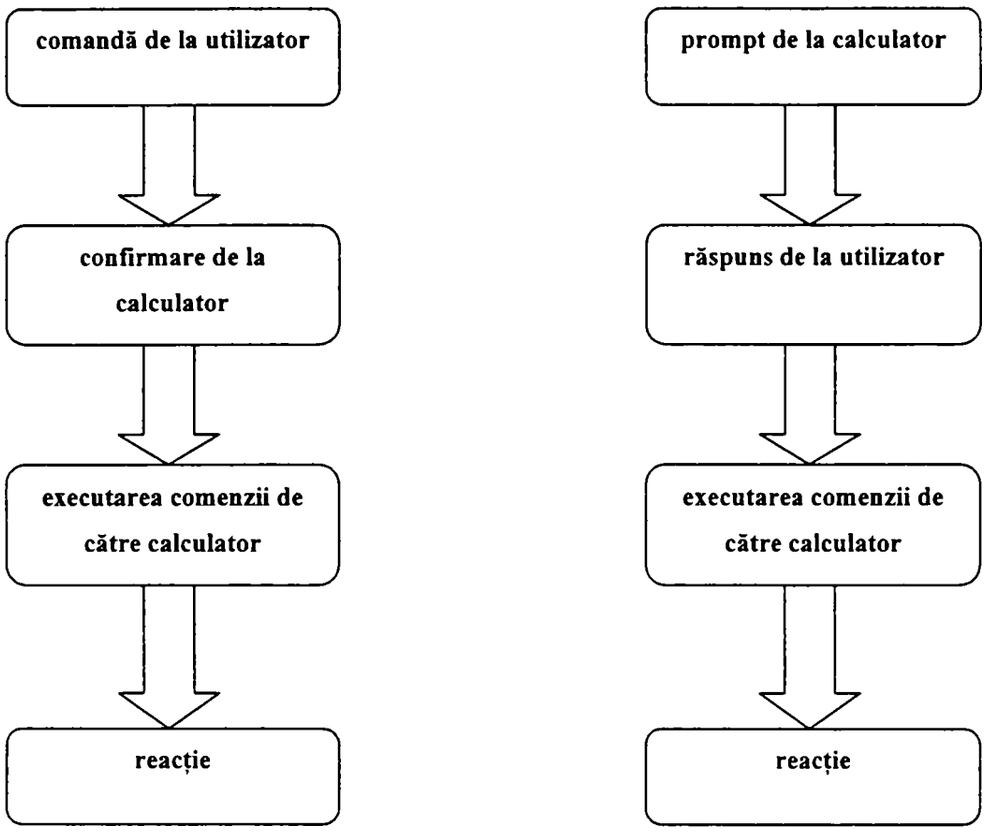


Figura 2-3. Tranzacții în interfața utilizator

La dezvoltarea interfețelor, trebuie să se țină cont de mai mulți factori:

- tranzacția trebuie inițiată şi încheiată în aceeaşi procedură;
- programul să reacționeze in timp util (timp de răspuns sub o secundă);
- interfața să permită restaurare/anulare;
- ordinea operanzilor afectează uşurința utilizării;
- mediul fizic în care lucrează utilizatorul, care influențează modalitatea de interacțiune utilizator-interfață.

Timpul de răspuns mic se referă la faptul că interfața trebuie să răspundă utilizatorului în timp util pentru a indica faptul că a fost *acceptată comanda* (nu necesar şi rezolvată) [Sav1997] [Shn1992].

Procesele de *restaurare* şi de *anulare* sunt necesare pentru a permite utilizatorului să anuleze (*undo*) efectele unor acțiuni pe care le-a comandat [Joh1992]. Înainte de executarea

unor acțiuni ale căror efecte sunt dificil de anulat, este indicat să se ceară confirmarea din partea utilizatorului.

Capitolul 4 dezvoltă elementele considerate în acest paragraf în cadrul unor tehnologii software utilizate pentru realizarea de interfețe cu utilizatorul.

Tipuri de interfețe cu utilizatorul

Principial, există două tipuri de interfețe cu utilizatorul: *bazate pe limbaje si grafice*. [Gul1996][Sav1997] Totuşi mai trebuie menționate si interfețele care asigura transferul bazelor de date intre aplicații. Deşi interacțiunea utilizatorului cu acest tip de interfețe nu este directa, pentru proiectantul unei aplicații aceasta reprezintă unul din dezideratele esențiale. Aceasta presupune o buna cunoaştere de către proiectant a aplicațiilor deja existente cu care se va interacționa, proiectantului revenindu-i sarcina de a realiza o interfață optima corespunzătoare comunicației între aplicații.

Interfețe bazate pe limbaje

Pentru comunicarea cu programul, utilizatorul introduce un mesaj sub forma unui şir de caractere (ASCII / ISO), de obicei pe o linie. Interfața va interpreta fiecare linie, interpretarea (*parsing*) constând din identificarea cuvintelor din mesaj şi ramificarea utilizând construcții de tip salt condiționat. Rezultatul va fi o cale specifică prin arborele tuturor combinațiilor posibile [She1989].

Prin analiză lexicală se stabileşte validitatea cuvintelor în cadrul limbajului, iar interpretorul (*parser*-ul) determină validitatea propoziției (mesajului) în cadrul limbajului.

Există mai multe tipuri de gramatici, din care pentru CAD/CAM se utilizează cu precadere gramaticile independente de context [Tay1992].

O *gramatică independentă de context* conține o mulțime de variabile sau entități nonterminale, descrise recursiv. Entitățile fundamentale din descrierea recursivă se numesc terminali, ele ne putând fii subdivizate în limitele gramaticii.

Regulile ce definesc o gramatică se numesc *producții*. Forma standard a unei producții este:

< a > –> < b > < c >

care se citeşte: nonterminalul din stânga poate fi înlocuit de nonterminalii din dreapta. "< >" indică variabile.

Interfețe grafice

Programele ce nu folosesc interfața de tastatură se bazează pe o formă de interacțiune prin ecran ca şi canal principal de comunicare cu utilizatorul. Această interacțiune se realizează cu tastele de control al cursorului, tablete grafice, mouse, ecran tactil, creion optic, rotițe (*thumbwheels*). În toate cazurile, programele se bazează pe o formă de interfață grafică cu utilizatorul (*graphical user interface* - GUI) şi un sistem de meniuri. GUI trebuie să lucreze ca gestionar de ferestre şi ca gestionar de meniuri [Fow1995].

Dispozitive logice de introducere

Standardele GKS şi PHIGS [Mye1990] prevăd şase clase de dispozitive logice de introducere de date: locatoare, tip flux (*stroke*), valuatoare, selectoare, interceptoare (identificatoare) şi şir de caractere.

Locatoarele servesc la specificarea unor poziții în spațiul modelului. Fizic, pot fi realizate prin mouse, joystick, tastatură, tabletă grafică (digitizoare), trackball, rotițe, creion optic etc. Ca răspuns (ecou) la activitatea locatorului, pe ecran se deplasează un cursor grafic.

Dispozitivele de tip **flux** furnizează o secvență de poziții la o singură operație (logică) de introducere (de exemplu pentru vârfurile unui poligon).

Valuatoarele au funcția de introducere a unor valori numerice. Fizic, pot fi realizate prin cursoare, rotițe, tastatură. În anumite condiții, dispozitivele locatoare pot fi folosite şi ca valuatoare (interpretând pozițiile sau lungimile ca valori numerice). De asemenea, se pot folosi interceptoare şi tastaturi numerice simulate pe ecran sau pe tabletă grafică.

Selectoarele permit alegerea unui element dintr-o listă de elemente grafice sau alfanumerice,. returnând numărul de ordine al elementului selectat.

Interceptoarele au funcția de selectare a unui element grafic afişat pe ecran. Selecția poate conține un element simplu sau un grup de elemente. Tipic, selecția de către interceptoare se realizează prin punctare. Interceptoarele furnizează identificatorul elementului grafic punctat şi cel al grupului din care face parte. Se poate realiza interceptarea mai multor entități simultan utilizând o fereastră (de regulă dreptunghiulară, indicată prin două vârfuri diagonal opuse), selecția cuprinzând entitățile strict interioare sau şi cele ce intersectează marginile ferestrei.

Dispozitivul tip **șir de caractere** are funcția de a furniza elemente de text. Realizarea fizică este prin tastatură, dar se poate folosi și un dispozitiv interceptor în combinație cu o tastatură simulată pe ecran sau pe o tabletă grafică.

Standardele GKS și PHIGS [Mye1990] prevăd trei **moduri de introducere**: la cerere (*request*), prin eșantionare (*sample*) și prin eveniment (*event*).

În modul **la cerere** aplicația așteaptă ca dispozitivul de introducere să furnizeze datele de intrare solicitate (un scalar, un vector etc.).

În modul **eșantionare** dispozitivul produce date în mod continuu, stocate într-un registru special de memorie, dar aplicația nu citește decât la anumite momente informația din acest registru.

În modul **eveniment** dispozitivul produce date în mod continuu, dar, spre deosebire de modul eșantionare, toate datele sunt memorate, aplicația putându-le citi la un moment convenabil.

Gestionarea ferestrelor

Gestionarea ferestrelor multiple este o funcție de nivel înalt ce este, de obicei, localizată într-un program separat, pentru a permite diferitelor ferestre să fie atașate la diferite programe. Un **gestionar de ferestre** este rareori creat de un programator CAD/CAM, care trebuie însă să cunoască gestionarul de ferestre al sistemului destinație [Sav1997] [Fow1995]. Cu toate că există standarde în domeniu, se întâlnesc și multe medii particulare. Interfețele grafice cu ferestre multiple sunt asociate de conceptul de multitasking (programe independente lucrând simultan).

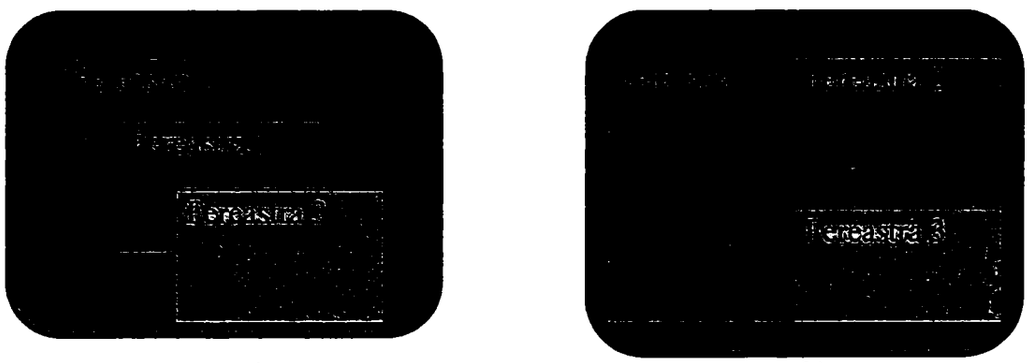
O altă cerință legată de ferestre apare în cazul imaginilor multiple afișate de același program - de exemplu vederi diferite ale aceluiași obiect [Opp1989]. Utilizatorul va avea posibilitatea de a aplica funcții *pan* și *zoom* independent pe vederi, chiar de a extinde una din vederi pe tot ecranul.

În mod similar, un program de simulare poate folosi ferestre aparte pentru selectarea parametrilor sistemului, pentru animație, pentru reprezentările grafice ale rezultatelor (răspunsurile sistemului). Se pot obține reprezentări ale mai multor variabile în mai multe ferestre.

În aplicațiile de dezvoltare de programe, se pot utiliza ferestre separate pentru editor, compilator, depanator, execuție program.

Opțiunile fundamentale ale gestiunii ferestrelor în cadrul interfeței cu utilizatorul sunt: *alăturarea sau suprapunerea ferestrelor și utilizarea de text sau de icoane* [Sav1997].

Prima caracteristică evidentă pentru utilizator este posibilitatea de acoperire a ferestrelor figura0 2-4 a) (*cascaded*), spre deosebire de situația din figura 2-4 b) (alăturate - *tiled*).



a) b)

Figura 2-4. Sisteme de ferestre suprapuse și de ferestre alăturate

Sistemul cu ferestre alăturate este indicat în cazul resurselor limitate (putere calculator și timp de programare) [Tay1992]. Sistemele avansate permit utilizatorului să aleagă oricare din variante.

Deoarece ferestrele au, de regulă, altă valoare a raportului laturilor față de cel al ecranului, apare problema evitării deformării imaginilor afișate în ferestre, mai ales la aplicațiile ce au sistemul de ferestre izolat de programul CAD/CAM [Fow1995] [Sav1997].

Pentru aplicații inginerești, ferestrele cu acoperire pot încetini mult execuția programului din cauza necesității recalculării întregii imagini dintr-o fereastră.

În orice moment doar una dintre ferestre este activă pentru introducere de date grafice. În unele sisteme, fereastra în care se află cursorul devine automat activă, la altele este nevoie de o metodă explicită de activare a ferestrei.

Icoanele servesc drept reprezentare grafică atât a ferestrelor, cât și a datelor. O fereastră închisă (invizibilă) trebuie să devină o icoană sau să apară într-o listă textuală de nume.

Unele funcții se realizează doar explicit, altele și implicit. De exemplu, selectarea poate realiza (implicit) și aducerea automată în față. Crearea și ștergerea sunt mai ales acțiuni ale programului, mai rar acțiuni explicite ale utilizatorului.

Gestionarea meniurilor

Un **meniu** este o listă de elemente (de exemplu indicând acțiuni), din care oricare poate fi selectat. În acest sens, meniul realizează o funcție de selector, dar poate include și elemente de tip valuator [Pfa1995].

Cele mai multe programe permit controlul lor de către utilizator printr-un sistem de meniuri interactive. Gestionarul de ferestre este, de obicei, un utilitar la nivelul sistemului de operare, iar gestionarul de meniuri - la nivelul programului individual. Există, însă, aplicații avansate CAD/CAM ce au sistem propriu de ferestre, pentru a putea lucra confortabil și sub sisteme de operare fără gestionar de ferestre.

Opțiunile privind sistemul de meniuri se referă la:

- meniuri pop-up sau pull-down;
- locație fixă sau mutabile;
- structură ierarhică (submeniuri);
- tratarea acțiunilor ilegale;
- utilizarea tastelor de comandă.

În ceea ce privește grafica meniurilor, opțiunile se referă la tipul lor: cu icoane, cu marcaje-cursor etc., iar în ceea ce privește textul, opțiunile sunt legate de: font, culoare text, culoare fond, stilul pentru evidențiere, stilul pentru element dezactivat.

Combinația între entitatea grafică și secvența de program ce implementează interacțiunile reprezintă un *widget* [Shn1992] [Gul1993].

Pachetul de gestiune a meniurilor trebuie să genereze entitățile grafice ce formează meniul, inclusiv structura ierarhică, datele textuale și datele iconice. De multe ori, nu toate acțiunile disponibile în program au corespondent în meniuri. Este de preferat să nu fie necesară amestecarea selectării din meniu cu tastarea (de exemplu la acțiunile ce necesită nume de fișier).

Crearea meniurilor este mult ușurată de programarea în mediu grafic orientată pe obiecte, deoarece meniurile au o mare cantitate de informații grafice ce trebuie specificate.

Sistemul de **gestiune a meniurilor** are două scopuri: crearea și utilizarea. Crearea implică folosirea sistemului de gestiune pentru generarea de noi meniuri sau modificarea celor existente.

Există două arhitecturi principale pentru sisteme de meniuri: *orientate pe date* şi *orientate pe program* (cod) [Pfa1995].

Ajutoare grafice în cadrul interfețelor grafice

Creşterea eficienței proiectării asistate de calculator este realizată şi prin utilizarea unor elemente speciale precum: modificatori geometrici, nume, straturi, culori, rețele de puncte, grupuri, trageri. Aceste elemente sunt cunoscute ca *ajutoare grafice*. Unele se referă la geometria modelelor, altele la managementul bazelor de date grafice [Tay1992] [Fol1995].

Modificatorii geometrici sunt utilizați pentru a uşura operațiile de introducere şi extragere a informațiilor grafice. Sunt comenzi care stabilesc regimul de poziționare a unei entități (de obicei punct) față de entități existente. De obicei se realizează o "prindere" ([*object*] *snap*) de entitățile existente. Principalele moduri utilizate sunt:

- **rețea**: cel mai apropiat punct al unei rețele fictive de puncte echidistante (*snap to grid)*;
- **capăt**: cel mai apropiat capăt al segmentului indicat (*endpoint*);
- **mijloc**: mijocul segmentului indicat (*midpoint*);
- **centru**: centrul cercului sau arcului de cerc indicat (*center*);
- **intersecție**: punctul de intersecție al unei perechi de entități grafice (*intersection*);
- **perpendicular**: piciorul perpendicularei din punctul indicat pe entitatea indicată;
- **tangent**: punctul de pe entitatea indicată ce realizează, împreună cu punctul indicat, o tangentă la entitate ;
- **cel mai apropiat**: cel mai apropiat punct de pe o entitate față de punctul indicat (*nearest*);
- **bază/referință**: punctul de referință al unui text sau bloc (*insertion, reference*);
- **cvadrant**: cel mai apropiat din cele patru puncte remarcabile ale cercului indicat (la 0, $\pi/2$, π, $3\pi/2$) (*quadrant*);
- **nod**: un punct nodal (*node*).

Nume. Pentru a ajuta referirea ulterioară, entitățile pot fi etichetate. Avantajul mare apare la programarea de macrocomenzi, minimizând interacțiunea cu utilizatorul. Nu mai este necesară punctarea, ci doar indicarea numelui (etichetei).

Straturi. Este avantajos pentru proiectant să grupeze informațiile legate de modele conform tipului lor. De exemplu, se realizează frecvent gruparea entităților auxiliare, de tipul cotelor, notelor tehnice. Pentru ca aceste grupe să poată fi tratate separat, se utilizează conceptul de *strat* (*layer*).

Straturile pot fi individual prevăzute cu atribute, cum ar fi vizibilitatea, tipul implicit și culoarea implicită a liniilor. Astfel, se poate realiza o vizualizare convenabilă a proiectului pentru diferite scopuri.

Culori. Culorile sunt utile pentru a putea distinge entitățile între ele. De obicei, fiecărui strat i se atașează (implicit sau de către utilizator) o culoare implicită, ce va fi utilizată la desenarea entităților de pe stratul respectiv.

Rețele de puncte. Pentru a ușura anumite operații grafice, se utilizează afișarea pe ecran a unei rețele de puncte echidistante (*grid*). La nevoie, rețeaua poate fi activată (vizibilă) sau dezactivată (invizibilă).

Rețeaua de puncte afișate poate fi definită pentru a se suprapune cu rețeaua de puncte pentru modificatori geometrici.

Rețelele pot fi paralele sau radiale. În mod obișnuit rețelele paralele sunt ortogonale, paralele cu axele de coordonate, dar se pot defini și rețele pe direcții oarecare, de exemplu, pentru a ajuta desenarea în planurile unei proiecții axonometrice izometrice.

Rețelele ajută la crearea de entități echidistante sau la distanțe multiplu de o valoare dată sau la mutarea în poziții ordonate.

Grupuri. În timpul realizării proiectelor apare necesitatea manipulării simultane a mai multor entități. Pentru a obține o tratare identică, entitățile pot fi asociate, temporar sau pe timp nedefinit, în grupuri, cărora li se pot asocia etichete. Operațiile de selecție se vor realiza rapid, indicarea unei entități din grup activând întreg grupul.

Dacă apare necesitatea prelucrării doar a unei entități din grup, acesta trebuie disociat.

Trageri. Tragerea (*dragging*) este o tehnică de mutare a unei entități cu ajutorul unui dispozitiv locator. După selectarea entității, pe timpul mișcării cursorului, entitatea selectată va fi mutată și ea, "agățată" de cursor (desenată punctat sau colorat diferit, eventual doar prin extensia de încadrare).

Un tip special de tragere este folosit la crearea entităților. Se simulează o linie elastică (*rubberband*) utilizând punctul anterior ce definește entitatea și poziția curentă a cursorului. Se aplică la toate tipurile de linii: segmente de dreaptă, arce de cerc, cercuri, elipse etc.

Prin tragere se realizează o reacție imediată pentru utilizator, pentru indicarea efectului pe care îl va avea comanda curentă.

2.4. Implicațiile ergonomiei software în proiectarea interfețelor cu utilizatorul

In general, în timpul dezvoltării unei aplicații se pune accent în principal pe criteriile și posibilitățile tehnice, ignorându-se alți factori esențiali, ce joacă un rol important în funcționarea sistemului. Pe lângă tehnologie - mediul, scopul și organizarea determină parametri activităților asistate de calculator. Obiectivul *ergonomiei* software îl reprezintă tocmai cooperarea optimă a acestor componente. De aceea, tehnologiei software i se poate atribui un puternic caracter pluridisciplinar [Maa1993].

Psihologia cognitivă, ergonomia, lingvistica, antropologia, sociologia precum și știința sistemelor și știința calculatoarelor se numără printre disciplinele care concurează în activitatea de proiectare a interfețelor cu utilizatorul. Contribuțiile pe care le au fiecare dintre componente sunt complexe. Fiecare disciplină în sine nu aduce un răspuns satisfăcător problemei ergonomiei software-ului. În acest caz soluțiile depind de contextul în care software-ul este utilizat [Opp1989] [Sch1983].

2.4.1. Direcții de bază în ergonomia software

În anii '70, de extensie a utilizării calculatoarelor în toate domeniile de activități, utilizarea aplicațiilor era însă dificilă deoarece necesita o muncă laborioasă, implicând comenzi complicate iar citirea informațiilor de pe dispozitivele de afișare era greoaie. Eforturi de îmbunătățire a acestei situații au început prin anii '80 cu unele sugestii, nesistematizate, obținute în cadrul cercetărilor asupra ergonomiei software. În același timp, sociologii au început să fie interesați de efectele ergonomiei software în cadrul economiei și administrației. Astfel ergonomia software și-a câștigat în timp rolul bine definit în contextul pluridisciplinar al domeniului [Maa1993] [Gul1993].

Latura tehnică

Dezvoltările hard- și software au fost pentru o lungă perioadă de timp orientate doar pe tehnologie, astfel că se punea accent doar pe îmbunătățirea laturii tehnice a sistemului. Viteza de operare a hardware-ului crescând, s-au dezvoltat posibilitățile de interacțiune cu utilizatorul (de ex. dispozitive grafice interactive). Simplificarea interacțiunii om-calculator a condus la enunțarea principiului „*manipulării directe*" [Shn1983], în care utilizatorul interacționează direct cu aplicația, având control și previzibilitate asupra interfeței utilizate.

Obiectele virtuale din cadrul sistemelor au început sa fie reprezentate grafic, astfel că ele erau mult mai uşor de înțeles. Utilizarea ferestrelor în cadrul interfețelor a condus la o mai bună imagine de ansamblu şi la o eficientizare a lucrului cu calculatorul prin operații simultane asupra obiectelor. Introducerea sistemului hipertext a fost următorul pas în cadrul realizării sistemelor cu interfață prietenoasă.

Pe lângă manipularea directă, reprezentarea informațiilor structurate mai complex este mult mai uşor de interpretat. Pe lângă text, în aceste structuri au fost introduse imagini, sunete şi filme/animație. Astfel calculatorul a devenit o unealtă multimedia. Termenul descrie un nou curent în cadrul ergonomiei software: *realitatea virtuală* [Gul1993] [Sav1997]. In acest caz utilizatorul interacționează cu senzori artificiali într-o lume virtuală.

Pentru a simplifica dezvoltarea aplicațiilor cu interfață prietenoasă trebuie realizate unelte software. Cu ajutorul acestora, dezvoltătorii de software nu mai trebuie să abordeze în toate detaliile interfața programului, ci doar să aleagă dintr-un set gata realizat.

Latura tehnică pune la dispoziție modele teoretice, ce descriu interacțiunile om-calculator. Astfel modelul Seeheim [Pfa1985] împarte aceste modele în componente distincte: *aplicație, dialog şi prezentare*.

Latura cognitiv-psihologică

La analiza sistemelor mai recente s-au utilizat modele psihologice de simulare ale percepției şi interpretării datelor de către oameni [Nor1986] [Lin1991b]. Astfel, reprezentarea figurată ar putea fi îmbunătățită după o analiză a aspectelor cognitiv-psihologice şi a datelor concrete cum ar fi nivelul de adâncime al meniului unei aplicații.

Cercetările în domeniul psihologicului au condus la noi sisteme de ajutor pentru utilizatori. Unul dintre procedee este în acest context explorarea individuală a sistemului de calcul.

Un model al interacțiunii om-calculator evidențiază două abordări fundamental opuse, *golful de execuție* („*gulf of execution*") şi *golful evaluării* („*gulf of evaluation*") [Nor1986]. Pe această bază se pot defini dificultățile pe care le întâmpină utilizatorul. În funcție de intențiile sale, utilizatorul îşi coordonează acțiunile asupra interfeței om-calculator (execuție) obținând un rezultat pe care trebuie să-l evalueze în funcție de intențiile sale (evaluare). Este

important ca utilizatorul să vadă posibilitățile pe care i le oferă sistemul precum şi modurile de introducere a datelor.

Cel mai cunoscut model psihologic asupra interacțiunii om-calculator este *modelul GOMS* adică scop, operatori, metode, reguli de selecție (*Goals, Operator, Methods, Selection rules*) enunțat de Card, Moran şi Newell în 1983 [Car1983]. Astfel, oamenii sunt priviți ca nişte unități de procesare a informațiilor ce sunt structurate similar cu calculatoarele şi rezolvă probleme prin ajungerea la condiția finală, pornind de la o condiție inițială şi ținând cont de o serie de operatori. Datele despre viteza de operare a oamenilor sunt apoi introduse în calculator pentru a minimiza timpul de operare.

Alte metode se leagă de procesul de gândire al oamenilor şi nu de gramatici. Cu ajutorul lor, interfețele existente pot fi analizate asupra capacității lor de învățare [Lin1991b].

Domeniul modelului GOMS se află în cadrul ergonomiei cognitiv-psihologice a software-ului ce implică metode cantitative. Modelul este criticat pentru că nu ia în considerare utilizatorii ocazionali pentru care ar fi necesară învățarea utilizării sistemului.

Pentru obținerea unei soluții optime, este nevoie şi de considerarea unei metode calitative, recurgându-se la experimente în cadrul cărora utilizatorii sunt chestionați asupra interacțiunea lor cu sistemul şi despre dificultățile întâmpinate.

Latura psihologiei muncii

Această abordare este prezentă mai ales în Europa, unde ergonomia software se prezintă preponderent în contextul organizării muncii umane, luându-se în considerare caracteristicile generale ale condițiilor de muncă [Sch1977].

O activitate lucrativă este ***umană*** *dacă ea nu afectează din punct de vedere al sănătății sau psihologic pe cel care o execută, adică corespunde necesităților şi calificării lui, si mai mult, îi stimulează personalitatea [Ulr1991].*

In anii 1975/1986 Volpert [Vol1975] şi Hacker [Hac1986] au dezvoltat *teoria psihologiei regularizării acțiunii*, care a devenit baza organizării umane a muncii. Astfel, munca este divizată pe mai multe niveluri, acțiunile sunt la rândul lor organizate pe niveluri, astfel încât să se treacă de la foarte complicat la foarte simplu.

Odată cu dezvoltarea sistemului trebuie acordată atenție şi păstrării libertății de acțiune, insistându-se asupra tuturor nivelelor de acțiune, în particular nivelelor mari. Pentru a realiza acest lucru, trebuie determinată mai întâi distribuția muncii persoanelor, în

continuare se determină ce porțiuni sunt suportate de către calculator şi ulterior se trece la realizarea concretă a sistemului.

2.4.2. Dileme ale ergonomiei software

În pofida căutărilor intensive în domeniul ergonomiei software, s-au au obținut relativ puține soluții practice, general valabile. Aceasta se datorează faptului că multe dintre întrebări sunt prea puțin precise şi nu se referă la contextul în care ele apar. Maaß dezvoltă această problemă pe baza unor abordări posibile ca de exemplu [Maa1993]:

- În principiu, comenzile trebuie să fie scurtate pentru a se evita erorile de introducere şi a eficientiza operarea cu calculatorul. Pentru abrevieri există mai multe abordări. Alegerea celei mai bune soluții pentru fiecare caz în parte depinde de numărul de comenzi şi de experiența utilizatorului. Abrevierile trebuie să fie clare pentru a evita posibilele greşeli, iar atribuirile trebuie să se încadreze in terminologia specializată a utilizatorului.

  O soluție consta in utilizarea meniurilor, ce reprezintă comenzi posibile în clar şi un mod alternativ mai rapid decât introducerea de la tastatură.

- Simplitatea unui sistem de calcul se poare referi la diferite componente. Un sistem poate fi simplu dacă el conține puține funcții sau există puține alternative pentru fiecare condiție care poate să apară. De asemenea el poate fi simplu dacă conține un număr mare de funcții transparente sau denumirile funcțiilor sunt sugestive.

  În contrast, se află complexitatea sistemului.

  În cadrul ergonomiei software, termenii interacționează între ei după cum urmează, în context cu organizarea umana a muncii.

  Sistemele complexe pot fi accesibile într-un mod simplu dacă ele sunt organizate şi prezentate într-un mod uşor de înțeles.

  Se ridică din nou întrebarea legată de organizarea umană a muncii. Simplitatea în cadrul aplicațiilor calculatoarelor nu trebuie luată în sensul de a subestima utilizatorii calificați prin restrângerea acțiunilor posibile (non-complexitate). Printr-o adaptare a sistemului la cunoştințele specializate ale utilizatorului sistemul poate să fie complex şi în acelaşi timp uşor de utilizat.

- Consecvența sistemului presupune o comportare uniformă, fără întreruperi. Un sistem este *consecvent intern* dacă de exemplu aranjarea ferestrelor, atribuirea tastelor sau

structura meniurilor din cadrul sistemului sunt în concordanță. *Consecvența externă* este realizată dacă sisteme diferite sunt echivalente structural astfel că utilizatorul poate trece ușor de la un sistem la altul.

Unii producători au introdus reguli de ghidare pentru consecvența internă și externă a sistemelor, cu toate că ele nu pot fi general aplicate fiecărui sistem individual.

De aceea trebuie efectuată o evaluare pentru a determina modurile în care creșterea complexității unui sistem ajută sau limitează utilizatorul în folosirea lui.

2.4.3. Condiții de studiu

În timpul organizării sistemului trebuie să se facă distincție între obiectul organizării și procesul de organizare.

Obiectul organizării

Au fost dezvoltate diferite modele pentru a ușura descrierea interfețelor cu utilizatorul. Pentru modelul nivelului practic este relevant modelul IFP [Ove1992]. Prin formalizarea utilizării interfeței se poate face diferențierea între aspectele organizării astfel că întreg sistemul poate fi dezvoltat, îmbunătățit sau analizat.

Firmele mari de software își clasifică produsele cu ajutorul unui set de reguli ce conțin principii generale de organizare și descrieri detaliate ale elementelor interfeței (de ex. sistemul de ferestre, imaginea butoanelor, seturile de caractere). Pentru aplicația software menționată, acestea sunt exact premisele de la care se trece în practică pentru organizarea exterioară. În contextul organizării umane a muncii, prin utilizarea unor asemenea premise tehnice nu trebuie uitat faptul că interfața sistemului încă nu este prietenoasă din punct de vedere al ergonomiei software.

Un alt element esențial pentru ușurința utilizării unui sistem este modul de prezentare al documentației. Atunci când există multe manuale, este foarte greu pentru un utilizator neexperimentat să găsească repede răspunsul despre un anumit element al sistemului. In acest caz este mai utilă o orientare a documentației în funcție de operații, ce oferă o privire de ansamblu începătorilor, la nevoie oferind și informații detaliate [Joh1992].

Procesul de organizare

În timpul procesului de organizare trebuie ținut cont de factori relevanți cum ar fi utilizatorii ulteriori, obiectivele și organizarea muncii lor precum și de facilitățile tehnice de

care dispun. Consultările cu viitorii utilizatori s-au dovedit importante. Acestea trebuie să se desfăşoare în condiții de *feedback* într-o manieră iterativă pe tot parcursul procesului de organizare. Utilizatorii pot testa prototipuri ale software-ului sugerând modificări. Sugestiile sunt luate în considerare la organizările ulterioare, dezvoltându-se versiuni noi ale programelor, care la rândul lor vor fi testate de către utilizatori pentru a verifica corectitudinea implementării sugestiilor [Bia1991] [Pol1992].

Includerea utilizatorilor în procesul de organizare are diferite efecte. Astfel, sunt luate în calcul calitatea şi uşurința operării, programatorul îşi poate forma o imagine legată de interacțiunea utilizatorului cu sistemul cât mai apropiată de realitate.

Pentru procesul de organizare exista două faze, cea cantitativă şi cea calitativă. În plus, deseori este utilă testarea sistemului în timpul dezvoltării nu numai din punctul de vedere al concepției cât şi din cel al utilizării, când se pot întâlni o multitudine de situații specifice. În cadrul proiectului EVADIS [Opp1989] s-a realizat un manual extensiv asupra evaluării sistemelor din punct de vedere al ergonomiei software-ului. Aici sunt evaluate metodic declarațiile utilizatorilor în funcție de construcția interfeței cu utilizatorul şi de principiile organizării sistemului.

2.4.4. Concluzii

Utilizarea calculatoarelor s-a răspândit exploziv la locurile de muncă şi acasă doar după ce în cadrul dezvoltării de software s-a luat în considerare şi gradul de uşurință în utilizarea aplicațiilor, învățarea operării unui sistem devenind mai simplă odată cu evoluția interfețelor cu utilizatorul. Astfel se poate spune că ergonomia aplicațiilor joacă un rol important în procesul de acceptare a calculatoarelor.

Un important pas a fost acela de a schimba scopul ergonomiei aplicațiilor de la un sistem stabil la acela de sistem flexibil, ce ia în considerare diferitele calificări ale utilizatorilor şi diferitele necesități complexe ale aplicațiilor, fiind utilizată o mai mare parte a potențialului.

Din (proprie) experiență s-a constatat că cea mai bună metodă de a învăța un sistem este acela de a-l explora în joacă. Astfel un sistem nou devine repede familiar utilizatorului evitând astfel supliciul pendularii constante între căutarea în documentația de pe hârtie şi aplicația de pe ecran (şi/sau asistența on-line). Documentația trebuie să aibă mai mult rolul de carte de referință decât un ghid. Premisa explorării în joacă forțează insa existența unor dispozitive de siguranță din cadrul sistemului, astfel că există protecții pentru cazuri cum ar fi

ştergerea neintenționată a datelor. Această metodă de familiarizare cu sistemul nu este oportună atunci când complexitatea aplicației este prea mare.

Un rol important în cadrul marketingului unui program îl reprezintă consecvența sistemului. Aranjarea similară a elementelor de interfață cu utilizatorul la aplicații provenite de la acelaşi producător produce utilizatorului un efect de confort şi familiaritate. În acest caz aşteptările şi încrederea utilizatorului asupra unui produs nou sunt strâns legate de experiența lui anterioară, pozitivă sau negativă, cu alte produse ale aceluiaşi producător. Cunoaşterea modului de utilizare a vechilor sisteme poate uşura modul de învățare al noului sistem. În realitate, cooperarea dintre proiectanții sistemului şi utilizatorii ce s-au confruntat cu situații concrete şi au colaborat la proiectarea sistemului se poare realiza în detaliu numai în cazul unor proiecte mari. În alte cazuri, este mai dificilă o participare intensivă a viitorilor utilizatori ai sistemului.

Calculatoarele nu pot fi total independente. Odată cu utilizarea lor în tot mai multe domenii, uşurarea lucrului cu calculatorul cade în sarcina software-ului. Unul din deziderate constă în crearea posibilității ca operatori fără o şcolarizare specială sau cunoaşterea unui volum mare de date tehnice să interacționeze uşor cu un sistem de calcul.

Realizarea de interfețe om-calculator corespunzătoare nevoilor psihologic-cognitive conform criteriilor ergonomice va fi posibilă doar când vor fi cunoscute natura reprezentărilor hipermedia şi altor tehnici ale interfeței om-calculator şi când realizarea lor va fi tehnologic posibilă. Viitorul ergonomiei software-ului va depinde esențial, ca şi până acum, de noile tehnologii pentru interfețele cu utilizatorul.

2.5. Concluzii

Obiectivul central al studiului IOC, o activitate pluridisciplinară, este înțelegerea interacțiunii dintre utilizator şi calculator, având ca rezultat proiectarea sistemelor pentru satisfacerea eficientă a utilizatorului în timpul luării deciziilor. Cunoaşterea modului de percepție al utilizatorului este esențială în luarea de decizii asupra organizării vizuale a interfeței, psihologia cognitivă şi analiza sistemelor furnizând un cadru teoretic pentru formularea, validarea şi analiza modelelor precum şi pentru analiza comportării sistemului studiat. Interfața cu utilizatorul trebuie proiectată astfel încât utilizatorul să o poată mânui automat, pe un nivel cognitiv mai scăzut, lăsând nivelele cognitive mai înalte pentru rezolvarea altor probleme legate de muncă.

Proiectarea interfeței este în principal o problemă de optimizare a interfeței cu utilizatorul pe baza cerințelor sistemului informațional, cu toate că ceea ce este optim într-un

anumit context poate fi dezastruos în altul. Pentru a obține cea mai bună soluție de proiectare este necesar ca proiectantul interfeței cu utilizatorul să posede cunoştințe avansate de utilizabilitate, să funcționeze ca o legătură între utilizatori şi programatori, în favoarea utilizatorilor, luând în considerare întregul lanț al analizei, proiectării şi evaluării.

Interfața cu utilizatorul sau dialogul om-calculator reprezintă singura modalitate de comunicare între utilizatori şi sistem. Descrierea şi analiza a şase tipuri de dispozitive de introducere, a trei moduri de introducere şi a modurilor de organizare a afişării informației a facilitat evaluarea celor mai utilizate forme de transfer a informației dintre utilizator şi sistem.

Creşterea vitezei de operare a hardware-ului a mărit posibilitățile de interacțiune cu utilizatorul, creându-se astfel un cadru propice cercetărilor din domeniul ergonomiei software de îmbunătățire a bunăstării omului şi a performanțelor globale ale sistemului prin optimizarea compatibilității om-sistem, luând în considerare factori fizici, cognitivi, sociali, organizaționali şi de mediu. Ergonomia software şi-a câştigat un rol bine definit în contextul interdisciplinar al proiectării interfețelor cu utilizatorul, având o contribuție importantă în procesul de acceptare a calculatoarelor. Viitorul ergonomiei software-ului va depinde esențial, ca şi până acum de altfel, de noile tehnologii pentru interfețele cu utilizatorul.

3. Analiza şi modelarea interfeţelor cu utilizatorul

O abordare care poate furniza informaţii utile proiectării interfeţelor este *analiza sistemică,* care, pe lângă analiza propriuzisă, se ocupă şi de construirea de modele, adică de *modelare*.

Un *sistem* este o parte limitată a lumii reale, definit de punctul de vedere şi de interesele observatorului. Celelalte părţi ale realităţii, care nu aparţin sistemului reprezintă mediul înconjurător (sau ambiant).

Sistemul este separat de mediu prin frontierele (limitele) sistemului care stabilesc care parte a realităţii este luată în considerare. Selectarea frontierelor este esenţială. După linia de demarcaţie care se trasează între sistem şi mediul ambiant se poate caracteriza sistemul prin relaţii intrare/ieşire.

Conform acestei abordări, calculatorul şi utilizatorul pot fi considerate ca fiind un singur sistem, sistem ce poate fi reprezentat de un model care poate fi modificat şi testat.

Testarea prin simulări fără a afecta lumea reală conduce la rezultate care pot fi luate în considerare la îmbunătăţirea sistemului.

Un *model* descrie aspecte relevante ale sistemului real, prezentând cunoştinţele asupra acelui sistem sub o formă utilizabilă, şi având în vedere un anumit scop, fără să fie o reprezentare completă a mecanismelor sistemului real [Pro2004]. Un model îşi justifică utilitatea atunci când reuşeşte să selecteze trăsături comune ale unui sistem, care să fie sesizate ca atare de mai mulţi subiecţi.

Analiza sistemelor furnizează un cadru teoretic cu privire la formularea şi validarea modelelor, modul cum pot fi ele analizate şi modul cum se pot trage concluzii legate de comportarea sistemului studiat.

Informaţiile obţinute prin modelare, simulare şi analiză sunt utilizate ulterior în priectare.

Modelele reprezintă un instrument important de lucru în dezvoltarea IOC. Interacţiunea utilizatorului cu calculatorul poate fi descrisă de un model. Utilizatorul, pe deoparte are un mod conceptual de gândire vizând sistemul informaţional, dar pe de altă parte şi cunoştinţe despre sistemul „real", care este în centrul interesului său într-o situaţie de lucru. Un sistem informaţional este testat de obicei cu un model (prototip).

Analiza sistemelor, precum şi utilizarea modelelor formale au un rol important în dezvoltarea sistemelor informaţionale. Analiza sistemelor implică analiza şi modelarea situaţiilor de lucru prezente şi viitoare. Rezultatul acestui proces constă în dezvoltarea de modele care să eficientizeze munca utilizatorilor cum ar fi modele de date, diagrame, cazuri de utilizare etc.

Fiecare model reprezintă o latură a sistemului analizat, nu este echivalent cu sistemul real şi este valid numai pe domeniul definit de scopul studiului. În cazul unei formulări corecte, în cunoştinţă de cauză şi verificată, modelele formale sunt utile în specificarea necesităţilor, documentarea utilizării informaţiei etc.

Înţelegerea interacţiunii utilizator - calculator reprezintă esenţa IOC. [Nor1986] a exprimat această interacţiune printr-un model ce implică unele activităţi orientate ale utilizatorului. Un utilizator efectuează o activitate cu intenţia de a atinge un obiectiv. Aceasta intenţie este concretizată într-o acţiune care este apoi executată. Acţiunea poate schimba starea în care se află calculatorul, schimbare care este percepută şi interpretată de către utilizator. În continuare, utilizatorul evaluează această stare în funcţie de obiectivele şi de intenţiile sale.

Această abordare permite analizarea interacţiunii om-calculator printr-o caracterizare de stare. Figura 3-1 reprezintă diagrama de tranziţii a stărilor calculatorului în urma interacţiunii cu utilizatorul pentru n stări [Dra2004]. T_{ij} reprezintă tranziţia din starea i în starea j a interfeţei în urma unei acţiuni a utilizatorului iar T_{ji} reprezintă revenirea la o stare anterioară în urma acţiunii de anulare a ultimei tranziţii. T_{ii} reprezintă acţiuni ale utilizatorului, cum ar fi modificări de configuraţie, în care starea calculatorului rămâne neschimbată.

Dificultăţile potenţiale ce apar atunci când se lucrează cu sistemele de calcul au fost descrise ca „golfuri" de execuţie şi evaluare. *Golful de execuţie* (*gulf of execution*) [Nor1986] este situaţia în care utilizatorul ştie ce obiective trebuie atinse dar nu ştie care variabile fizice şi/sau în ce mod să le ajusteze. *Golful evaluării* (*gulf of evaluation*) [Nor1986] apare atunci când sistemul s-a modificat, de obicei în urma acţiunii utilizatorului, dar acesta nu poate înţelege uşor modificarea în starea sistemului. Aceste dificultăţi sunt generate de modul de proiectare a interfeţei.

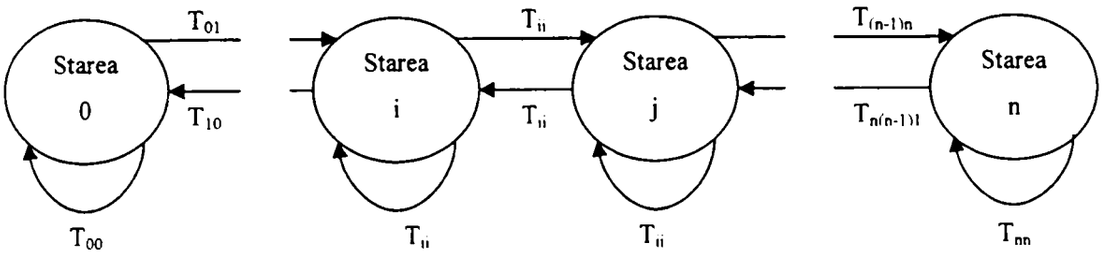


Figura 3-1. Diagrama de tranziţii a stărilor calculatorului în urma interacţiunii cu utilizatorul

Consideraţiile anterioare conduc în mod firesc la ideea că interfaţa cu utilizatorul poate fi considerată, modelată şi analizată ca un *sistem cu evenimente discrete*.

3.1. Sisteme cu Evenimente Discrete

Sistemele cu evenimente discrete constituie o clasă aparte de sisteme dinamice neliniare, care necesită pentru investigare instrumente proprii, complet diferite de cele utilizate în analiza sistemelor clasice (continuale/discrete).

*Dacă spaţiul stărilor unui sistem este descris printr-un set discret de valori, iar tranziţiile sunt observate doar la momente discrete de timp, atunci aceste tranziţii de stare sunt asociate cu "evenimente" iar sistemul este "**un sistem cu evenimente discrete**" (SED) [Cas2001] [Pas1997].*

Un *eveniment* poate fi identificat ca o anumită acţiune (apăsarea unui buton), poate fi privit ca o întâmplare accidentală sau poate fi rezultatul a mai multor condiţii îndeplinite simultan.

Din punct de vedere formal, SED sunt considerate, de regulă, ca sisteme dinamice caracterizate prin existenta unui spaţiu al stărilor şi al unui mecanism de tranziţie al stărilor.

SED se deosebesc în mod esenţial de celelalte categorii de sisteme prin faptul că, comportarea lor dinamică este determinată de producerea (realizarea) unor evenimente (*event driven*) şi nu este antrenată de timp (*time driven*) [Let1998].

In cadrul sistemelor cu stări discrete, tranziţiile pot apare la anumite momente de timp determinate - *sisteme pe bază de timp,* sau nedeterminate - *sisteme pe bază de evenimente*.

Iniţierea tranziţiilor de stare poate fi sincronizată cu un ceas sau poate fi asincronă. Sistemele pe bază de evenimente sunt mai greu de modelat şi analizat datorită faptului că există mai multe mecanisme de temporizare asincronă pe bază de evenimente, care pot fi specificate ca parte a sistemului modelat.

Principalele caracteristici ale SED sunt:

- stările au valori discrete si evenimentele se produc la momente discrete de timp;
- procesele sunt conduse de evenimente si nu antrenate de timp;
- procesele sunt tipic nedeterministe, funcţionând după un mecanism care nu este modelat de analistul sistemului, nefiind formulate caracteristici stochastice, interesând posibilitatea şi nu probabilitatea realizării evenimentului;
- procesele au în general o comportare dinamică internă, reacţionând şi interacţionând cu mediul;
- procesele operează concurent şi comunică unul cu altul;
- pentru o operare sigură, deseori trebuie precizate duratele de timp care descriu constrângerile pentru efectuarea unor activităţi; corectitudinea funcţionarii sistemului nu depinde doar de rezultatele logice ale comportamentului sistemului ci şi de timpul în care aceste rezultate sunt disponibile, sau unele activităţi sunt efectuate.

Sintetizând, se poate afirma că:

- *un SED este un sistem pilotat de evenimente, tranziţiile stărilor fiind determinate de apariţia evenimentelor;*
- *spaţiul stărilor pentru SED este un set discret.*

In [Cas2001] este formulată **definiţia**:

Un SED este un sistem cu stări discrete şi condus de evenimente (event driven) in care evoluţia stărilor depinde numai de apariţia asincrona a unor evenimente total independente de timp.

Evoluţia SED fiind determinată de evenimente, modelarea şi analiza lor necesită un formalism nou, diferit de cele clasice bazate pe ecuaţii diferenţiale sau ecuaţii cu diferenţe finite.

Scopul analizei SED *este dezvoltarea de modele* corespunzătoare, care descriu în mod adecvat comportamentul acestor sisteme şi oferă un cadru de utilizare a tehnicilor pentru proiectare, conducere şi evaluarea performanţelor.

In cadrul analizei evoluţiei în spaţiul stărilor a unui SED, o primă activitate este aceea de determinare a secvenţelor de stări vizitate şi a evenimentelor asociate, care iniţiază aceste tranziţii de stare.

Secvenţa de evenimente se poate descrie sub forma e_1, e_2, ..., e_n. O secvenţă de această formă specifică ordinea în timp în care apar evenimentele dar nu indică şi momentele exacte de timp asociate producerii acestor evenimente. Această reprezentare este atemporală, în care sistemul este *modelat printr-un limbaj*.

Una dintre modalităţile de studiu formal al comportamentului logic al SED se bazează pe *teoria limbajelor şi automatelor*. Punctul de plecare este acela că orice SED are asociat un set de evenimente E. Setul E poate fi considerat ca şi "*alfabetul*" limbajului iar secvenţele de evenimente ca şi "*cuvinte*".

*Un **automat** este un sistem capabil sa reprezinte un limbaj conform unor reguli bine definite.*

Cea mai simplă metodă de ilustrare a noţiunii de automat este reprezentarea sa sub formă de graf sau diagramă de tranziţii de stare [Cas2001].

In figura 3-2 este reprezentat un graf direcţionat în care nodurile reprezintă *stări* iar arcurile etichetate reprezintă *tranziţiile* dintre aceste stări. Acest graf reprezintă o descriere completă a unui automat.

Setul de noduri reprezintă setul stărilor automatului, X={x,y,z}.

Setul etichetelor pentru tranziţii este setul de evenimente (alfabetul) automatului, E={a,b,g}.

Starea iniţială este x (marcată de o săgeată).

Arcurile din graf sunt o reprezentare grafică a *funcţiei de tranziţie* a automatului definită sub forma:

$$f: X \times E \rightarrow X$$

$$f(x,a)=x \qquad f(y,a)=x$$

$$f(z,b)=z \qquad f(x,g)=z$$

$f(y,b)=y$ $f(z,a)=f(z,g)=y$

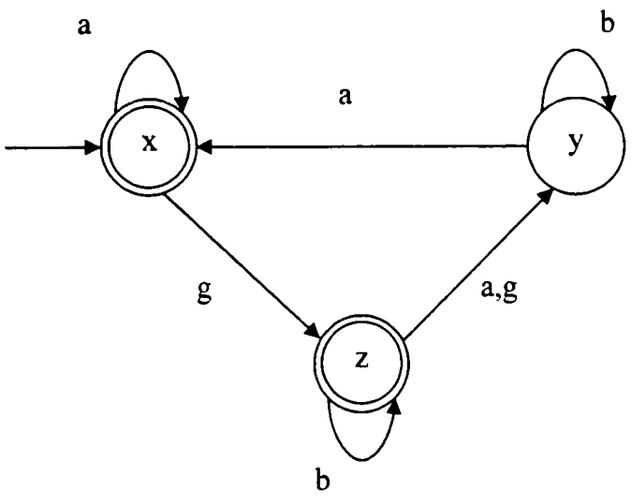


Figura 3-2. Diagrama tranziţiilor de stare ale SED

Notaţia: $f(y,a)=x$ semnifică faptul că dacă automatul este în starea y, la apariţia evenimentului a, va apare o tranziţie instantanee la starea x. Cauza apariţiei evenimentului a este irelevantă, ea poate să fie o interacţiune din exterior către sistemul modelat de automat.

Pot să apară următoarele situaţii:

- un eveniment poate apare fără să schimbe starea, ex. $f(x,a)=x$.
- două evenimente distincte pot apare la o anumită stare cauzând exact aceeaşi tranziţie, ex: $f(z,a)=f(z,g)=y$
- f este o funcţie parţială pe domeniul ei $X \times E$, adică nu trebuie să existe o tranziţie pentru fiecare eveniment din E pentru fiecare stare din X, ex: f(x,b) si f(y,g) nu sunt definite.

Pentru a avea o definiţie completă a unui automat, mai trebuie definită o stare iniţială x_0 şi un subset X_M al X care reprezintă *stările marcate* din X. Marcarea stărilor se face atunci când se doreşte asocierea de semnificaţii speciale unor anumite stări cum ar fi stările finale.

Un automat determinist, notat G, reprezintă sextuplul [Cas2001]:

$$G = (X, E, f, \Gamma, x_0, X_M)$$

Unde:

X - reprezintă setul *stărilor*

E - reprezintă setul finit de *evenimente* asociate cu tranziţiile din G

$f : X \times E \rightarrow X$ reprezintă *funcţia de tranziţie*: f(x,e) = y care semnifică faptul că există o tranziţie corespunzătoare evenimentului e din starea x în starea y; în general *f* este o funcţie parţială pe domeniul ei

$\Gamma : X \rightarrow 2^E$ este *funcţia eveniment activ* (funcţia eveniment fezabil); $\Gamma(x)$ reprezintă setul tuturor evenimentelor e pentru care există f(x,e) şi se numeşte *setul eveniment activ* al lui G în x

x_0 - reprezintă starea *iniţială*

$X_M \subseteq X$ reprezintă setul de *stări marcate*.

Dat fiind setul A, notaţia 2^A reprezintă super-setul lui A, adică setul tuturor sub-seturilor lui A.

Funcţionarea automatului se desfăşoară în modul următor: Porneşte din starea iniţială x_0 iar la producerea unui eveniment $e \in \Gamma(x_0) \subseteq E$ se realizează tranziţia către starea $f(x_0, e) \in X$. Procesul se repetă pe baza tranziţiilor pentru care f este definită.

Conexiunea dintre limbaje şi automate se face pe baza analizei diagramei tranziţiilor de stare a unui automat. Se iau în considerare toate căile ce pot fi urmate în cadrul diagramei tranziţiilor de stare pornind din starea iniţială, şi dintre acestea toate căile care se sfârşesc într-o stare marcată. De aici rezultă noţiunile de limbaje *generate* şi *marcate* de către un automat.

Limbajul generat de $G = (X, E, f, \Gamma, x_0, X_M)$ este

$$L(G) := \{s \in E^* : f(x_0, s) \text{ este definit}\}$$

Limbajul marcat de G este

$$L_m(G) := \{s \in L(G) : f(x_0, s) \in X_m\}$$

Astfel, un automat G este o reprezentare a *două* limbaje: $L(G)$ şi $L_m(G)$.

Două automate sunt *echivalente* dacă ele generează *şi* marchează aceleaşi limbaje. Formal, automatele G_1 şi G_2 sunt *echivalente* dacă

$L(G_1) = L(G_2)$ şi $L_m(G_1) = L_m(G_2)$

Un automat G poate ajunge într-o stare x, unde $\Gamma(x) = 0$ dar $x \notin X_m$. Această situaţie este considerată *deadlock* pentru că nu mai poate fi executat nici un alt eveniment. Un sistem se blochează atunci când intră intr-o stare *deadlock*, fără a termina execuţia sarcinii curente.

O altă situaţie ce poate apare este cea în care un set de stări nemarcate din G formează o componentă conectată puternic (aceste stări pot fi atinse de la una la alta) dar *fără nici o tranziţie în afara setului*. Situaţia în care sistemul intră într-un asemenea set se numeşte *livelock*. Cu toate că sistemul funcţionează (este *live*), în sensul că întotdeauna el poate executa un eveniment, el nu poate finaliza niciodată sarcina curentă pentru că nu există nici o stare marcată, iar sistemul nu poate părăsi acest set de stări.

Formal, un automat G este *blocabil* dacă

$$\overline{L_m(G)} \subset L(G)$$

în care incluziunea setului este corespunzătoare, şi *neblocabil* dacă

$$\overline{L_m(G)} = L(G)$$

Un exemplu de automat blocabil este ilustrat în figura 3-3 în care starea 5 este o stare *deadlock* iar stările 3 şi 4 formează un *livelock*.

Există două operaţii de compunere a automatelor: *produsul*, notat „×", şi com*poziţia paralelă* notată „||" [Cas2001]. Compoziţia paralelă mai este numită şi compoziţia co*mplet sincronă*.

Figura 3-4 ilustrează interconexiunile a două automate. La operaţia „×" participă doar evenimentele din $E_1 \cap E_2$ iar în operaţia „||" participă toate evenimentele din $E_1 \cup E_2$

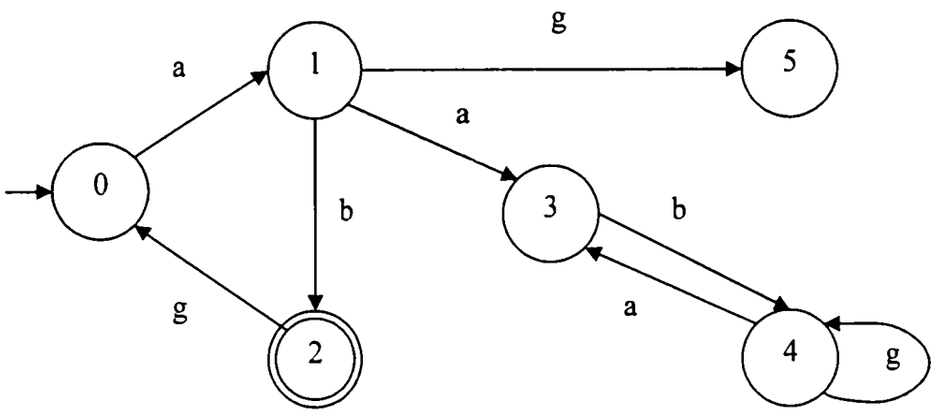


Figura 3-3. Automat blocabil

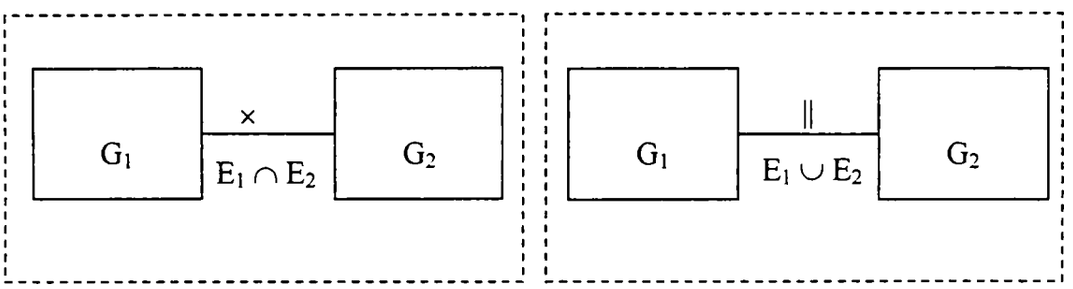


Figura 3-4. Interconexiunea a două automate

Considerând două automate:

$G_1=(X_1, E_1, f_1, \Gamma_1, x_{01}, X_{M1})$ şi $G_2=(X_2, E_2, f_2, \Gamma_2, x_{02}, X_{M2})$

produsul dintre G1 şi G2 este automatul

$$G1 \times G2 := Ac(X_1 \times X_2, E_1 \cap E_2, f, \Gamma_{1x2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

unde f este definită pe domeniul $e \in \Gamma_1(x_1) \cap \Gamma_2(x_2)$ ca:

$$f((x_1, x_2), e) := (f_1(x_1,e), f_2(x_2,e))$$

funcţia Ac(.) selectează setul de stări accesibile pornind din starea iniţială x_0 utilizând orice şir din $L(G)$ iar:

$$\Gamma_{1x2}(x_1,x_2) = \Gamma_1(x_1) \cap \Gamma_2(x_2)$$

Ca o consecinţă se observă faptul că tranziţiile din cele două automate sunt sincronizate cu un eveniment comun, din $E_1 \cap E_2$.

Rezultatul *compoziţiei paralele* a automatelor G1 şi G2 este automatul

$$G_1 \| G_2 := Ac(X_1 \times X_2, E_1 \cup E_2, f, \Gamma_{1\|2}, (x_{01}, x_{02}), X_{m1} \times X_{m2})$$

unde

$$f((x_1, x_2), e) := \begin{cases} (f_1(x_1,e), f_2(x_2,e)) & \text{pt. } e \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1,e), x_2) & \text{pt. } e \in \Gamma_1(x_1) \setminus E_2 \\ (x_1, f_2(x_2,e)) & \text{pt. } e \in \Gamma_2(x_2) \setminus E_1 \end{cases}$$

şi

$$\Gamma_{1\|2}(x_1, x_2) = [\Gamma_1(x_1) \cap \Gamma_2(x_2)] \cup [\Gamma_1(x_1) \setminus E_2] \cup [\Gamma_2(x_2) \setminus E_1]$$

In cazul compoziţiei paralele, un eveniment comun poate fi executat doar dacă ambele automate îl pot executa simultan. Astfel, cele două automate sunt "sincronizate" cu evenimentele comune. Restul evenimentelor din $(E_2 \setminus E_1) \cup (E_1 \setminus E_2)$, nu se supun acestei constrângeri şi pot fi executate atunci când este posibil. Atunci când $E_1 = E_2$ compoziţia paralelă se reduce la produs din moment ce toate tranziţiile sunt sincronizate forţat. Dacă $E_1 \cap E_2 = 0$, atunci nu există tranziţii sincronizate, astfel că $G_1 \| G_2$ se consideră comportamentul *concurent* al G_1 şi G_2.

3.2. Modelarea interfeţelor cu utilizatorul

Un calculator pe care rulează un sistem de operare şi un set de aplicaţii poate fi considerat un *sistem dinamic,* iar interfaţa sa cu utilizatorul reprezintă modul în care acesta prezintă informaţia şi interpretează *acţiunile* în timp ale utilizatorului.

Din punctul de vedere al *utilizatorului final*, starea dinamică a calculatorului îi limitează acţiunile ce pot fi efectuate, acest lucru afectând şi sarcinile pe care utilizatorul trebuie să le efectueze pentru a-şi atinge *scopurile*.

[Van1999] enunţă nouă tipuri diferite de modele care pot fi utilizate în proiectarea interfeţelor, prezentate în tabelul 3-1.

Tabelul 3-1. Tipuri de modele de interfaţă

Tip	Continut	Reprezentare
Sarcina	Ce face sau doreşte utilizatorul şi de	Există două tipuri principale de modele ale sarcinii: modele orientate pe ordine/secvenţă şi modele orientate pe proces/vehicularea datelor.

	ce	Primul tip conferă o importanţă crescută secvenţei sub-sarcinilor din cadrul unei ierarhii, cel de-al doilea datelor utilizate in cadrul sarcinilor pentru control si procesare. Modelele sarcinilor sunt utilizate la descrierea sarcinilor existente pentru îmbunătăţiri şi la conceperea de sarcini viitoare pentru care se va proiecta o interfaţă cu utilizatorul.
Domeniu/ date	Concepte, obiecte şi operaţii etc. ce descriu domeniul	Sarcinile utilizează datele unui domeniu şi operează în cadrul domeniului. Formalismele principale sunt entităţi cu atribute şi relaţii şi respectiv obiecte cu atribute, relaţii şi operaţii
Dialog/ conversatie	Structura dialogului dintre om şi calculator	Se utilizează abstractizări cum sunt interactorii, reţelele Petri, diagrame de flux, diagrame UML de activitate şi secvenţă şi respectiv diagrame de tranziţii de stări.
Interacţiune a concretă - Prezentare	Structura ieşirilor şi corelarea cu date şi operaţii	De obicei se reprezintă prin elemente de dialog generice dar concrete şi grafică
Interacţiune a concretă – Comporta-ment	Modurile de introducere a datelor şi iniţierea operaţiilor	Descrie desfăşurarea dialogului în funcţie de interacţiunea a utilizatorului şi include legătura dintre prezentare şi dialog, (ex: clicul unui buton pentru a deschide o vedere pop-up)
Comandă	Serviciile aplicaţiei de care depinde execuţia sarcinii	De obicei este o listă de comenzi sau operaţii de obiecte ce pot fi iniţiate cu pre-/post- condiţiile. Poate fi considerat ca o specificaţie funcţională a aplicaţiei
Platforma	Posibilităţile de introducere şi prezentare ale sistemului	Se pot defini atribute şi calităţi care pot fi referite de regulile de corelare a obiectelor de interacţiune.
Mediu	Contextul fizic şi	Descrieri informale

	cultural al interacţiunii	
Utilizator	Caracteristici şi abilităţi ale utilizatorului final, care nu sunt incluse în alte modele	Ierarhie de clasificare a stereotipurilor de utilizatori, care conţine atribute şi calităţi ce pot fi referite.

Reprezentările (modelele) formale sunt orientate spre descrieri concise, complete şi precise, caracteristice pentru o analiză obiectivă.

Se consideră însă că reprezentările formale sunt dificil de citit şi scris, fiind înţelese doar de cei experimentaţi în formalisme.

[Ban1989] împarte abordările de reprezentare în trei direcţii: teoretice de sistem, socio-tehnice şi critice. Ele corespund în mare măsură cu *funcţionalismul*, *relativismul social* şi *structuralismul radical*. Multe dintre formalismele actuale sunt axate pe analiza şabloanelor de sarcini şi a comportamentului uman, cu scopul de a fi utilizate în proiectarea interfeţelor cu utilizatorul.

Reprezentările formale utilizate ca şi mediu de comunicaţie între oameni, corespund cel mai bine abordării funcţionaliste, pe când în cadrul relativismului social sunt utile în măsura în care nu periclitează dialogul.

[Flo1987] prezintă două paradigme de dezvoltare a sistemelor: vederile orientare spre *produs* şi spre *proces*. Prima este caracterizată de accentul pus pe reprezentările produsului, unde dezvoltarea unui program este "... o tranziţie controlată pornind de la definiţia formalizată până la cod executabil...". Reprezentările formale sunt utilizate pentru a asigura tranziţia corectă, astfel încât această vedere permite nu numai utilizarea modelelor formale dar depinde în mare măsură de aceasta.

Printre avantajele reprezentărior informale se numără şi natura lor vizuală şi concretă. Cu toate că majoritatea limbajelor de modelare a interfeţelor cu utilizatorul sugerate sunt grafice, ele tind să fie relativ abstracte.

3.2.1. Modelele principale de interfaţă cu utilizatorul

Produsul primar al procesului de proiectare a interfeţei cu utilizatorul este o specificaţie a interfeţei cu utilizatorul, cu un anumit grad de utilizabilitate, care face posibilă implementarea şi care presupune crearea şi utilizarea mai multor tipuri de reprezentare. Limbajul de reprezentare poate fi privit ca un instrument de realizare a acestei specificaţii şi trebuie să fie capabil să includă elementele necesare proiectării unei interfeţe utilizabile.

Modele utilizate în proiectarea bazată pe modele prezentate în tabelul 3-1, reprezintă diferite abordări sau perspective ale domeniului proiectării interfeţelor cu utilizatorul. O parte dintre acestea pot fi considerate a fi orientate pe *probleme*, pentru că descriu cerinţe ale domeniului sau ale scopurilor ce se doresc a fi atinse, în timp ce altele sunt orientate pe *soluţii*, pentru că descriu aspecte ale produsului proiectat precum şi aspecte ale mediului în care va opera produsul [Ben1996]. Accepţiunile celor două abordări sunt:

- problemă/domeniu: caracteristicile diferitelor tipuri de *utilizatori*, *structura sarcinilor* pe care utilizatorul doreşte să le efectueze şi vederea utilizatorului asupra *domeniului*;
- soluţie/produs: *structura dialogului* dintre utilizator şi sistem, şi *interacţiunea concretă* ce detaliază vizualizarea şi interacţiunea.

Sintetizând, cele trei tipuri de modele principale de interfaţă cu utilizatorul sunt:

- *sarcină/domeniu;*
- *dialog abstract;*
- *interacţiune concretă.*

Limbajele propuse pentru modelarea interfeţelor tind să favorizeze doar câte o singură abordare. Tipul 'sarcină' este orientat pe structura muncii, tipul 'dialog' pe ciclul de viaţă şi pe activarea elementelor de dialog, iar tipul 'interacţiune' pe modul de prezentare a interfeţei cu utilizatorul.

In cadrul proiectării unei interfeţe, sunt utilizate mai multe limbaje pentru acoperirea tuturor abordărilor necesare.

Abordarea şi respectiv gradul de abstractizare a obiectelor definesc un spaţiu cu două dimensiuni (plan): problemă/soluţie şi abstractizare scăzută/crescută (figura 3-5).

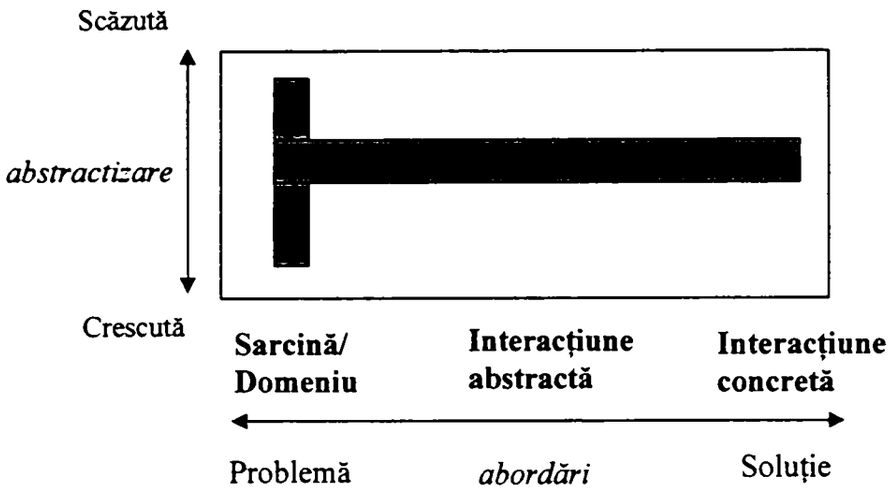


Figura 3-5. Spaţiul de reprezentare a proiectării: abordare şi nivel de abstractizare

Oricare reprezentare a unui proiect sau metodă/limbaj de modelare acoperă doar o parte a acestui plan. Reprezentările verticale acoperă obiecte din aceeaşi abordare cu abstractizări diferite, ceea ce este tipic pentru modelele formale. Reprezentările informale, cum sunt scenariile sau schiţele, precum şi limbajul natural, nu sunt potrivite acestor tipuri de abordări.

Lucrări de dată mai recentă privind proiectarea bazată pe modele a interfeţelor cu utilizatorul au reuşit să învingă reticenţele proiectanţilor în utilizarea reprezentătilor formale demonstrând importanţa acestora [Gul2004] [Van2001] [Sze1996]. Modelele utilizate (menţionate anterior) permit proiectarea şi evaluarea interfeţelor cu utilizatorul, exploatând avantajele reprezentărilor formale.

3.2.1.1 Modelarea sarcinii

Modelarea sarcinii poate fi privită ca o modalitate de formalizare a analizei sarcinii, scopul fiind înţelegerea mai buna a structurii utilizatorilor (cine?), a mediului lor de lucru (resurse), a scopurilor şi a sarcinilor lor lucrative (cum?).

In cazul evaluării interfeţelor existente, modelarea sarcinii va fi orientată mai mult pe soluţii, în timp ce în cazul proiectării de noi interfeţe cu utilizatorul modelarea sarcinii va fi orientată mai mult pe probleme.

Înţelegerea naturii şi structurii muncii utilizatorului este esenţială pentru realizarea cu succes sistemelor informaţionale (SI) dedicate unor „organizaţii" [Eis2000] din cel puţin două motive: 1) pentru ca un SI să fie inteligibil pentru utilizatorii respectivi, acesta trebuie să ţină cont de practicile curente de muncă ale „organizaţiei", 2) din cauză că SI determină într-o oarecare măsură ce sarcini pot fi efectuate şi modul cum pot fi efectuate, acesta trebuie să fie proiectat conform obiectivelor şi scopurilor „organizaţiei" respective.

[Mar1997] identifică patru elemente esenţiale care descriu o activitate: *structura acţiunii*, *actorii* ce o efectuează, *instrumentele* utilizate şi *informaţiile* necesare.

Conceptul esenţial este cel de *acţiune*. Acţiunile sunt cauzele tuturor schimbărilor şi constituie paşii ce trebuie efectuaţi pentru atingerea scopului. Pentru ca o acţiune să fie posibilă, trebuie satisfăcute anumite *precondiţii* implicite şi explicite, adică acţiunea să fie relevantă pentru scop iar resursele necesare, cum ar fi informaţia şi instrumentele, să fie disponibile. Precondiţiile unei acţiuni pot depinde de alte acţiuni, rezultând în mod natural o structurare pe dependenţe a acestora.

Pentru a evita complexitatea, structurarea unei acţiuni poate fi ierarhizată prin definirea de acţiuni compuse. Conceptul de acţiune corespunde conceptului de sarcină, iar ierarhia acţiunilor structurii sarcină/sub-sarcină, regăsindu-se în cadrul limbajelor de modelare a sarcinilor.

Actorii sunt utilizatorii ce efectuează intenţionat acţiuni. Ei au o stare (mentală) care le ghidează comportamentul (atingerea unui scop, responsabilităţi şi îndatoriri). Din punct de vedere tehnic, actorii pot fi priviţi ca nişte resurse cu anumite caracteristici şi abilităţi necesare acţiunilor [Mar1997]. Prin enunţarea caracteristicilor necesare, cum sunt drepturi şi privilegii, cunoştinţe şi abilităţi, acţiunile definesc implicit cine le poate efectua. Asemenea acţiunilor, actorii pot fi combinaţi rezultând *grupuri*. *Rolurile* definesc grupuri de actori prin enunţarea caracteristicilor lor comune, necesare acţiunilor. Cu toate că, de obicei, grupurile corespund structurilor formale de organizare, ele pot conţine indivizi arbitrari.

Obiectele sunt elemente materiale, mentale sau de reprezentare, care constituie domeniul în care sunt efectuate acţiunile. Obiectele sunt utilizate deopotrivă la reprezentarea informaţiei asupra căreia se acţionează, precum şi la modelarea informaţiei contextuale utilizate în cadrul acţiunilor.

Un *instrument* este un caz special din categoria *resurse*, care a fost considerat ca fiind necesar pentru efectuarea acţiunilor, inclusiv actorii care le efectuează, obiectele asupra cărora se acţionează precum şi alte informaţii contextuale necesare [Mar1997]. Instrumentele sunt de obicei aplicaţii sau componente din cadrul unor sisteme integrate. Un instrument poate fi specificat în mod concret, o anumită aplicaţie (AutoCAD), sau în mod abstract, o clasă de aplicaţii (aplicaţie CAD).

Instrumentele sunt elemente software care suportă efectuarea acţiunilor, care la nivelul sarcinii utilizatorului corespund cu *elemente de dialog*. Conceptul de instrument face legătura dintre *specificaţia* (orientată spre probleme a) funcţionalităţilor interfeţei cu utilizatorul, şi elementele de proiectare a interfeţei cu utilizatorul (orientate spre soluţie) care le implementează [Pri2001].

3.2.1.2 Modelarea Dialogului

Modelarea dialogului presupune analiza schimbului de informaţie, activarea şi secvenţierea, în timp ce interacţiunea concretă se concentrează asupra relaţiei dintre dialog şi dispozitivele de intrare şi ieşire, cum sunt ferestrele, mouse-ul etc.

Modelarea dialogului respectiv a interacţiunii concrete este realizată utilizând conceptul de interactor, considerat ca o abstractizare „matură" [Duk1994].

Un interactor specifică o soluţie de proiectare a unei interfeţe în contextul unei descrieri prin metafore. Tipurile de interactori se bazează pe o abstractizare a unui meta-obiect (obiecte, constrângeri, acţiuni, evenimente) în combinaţie cu una sau mai multe abstractizari de elemente de afişare şi comandă. Un interactor poate conţine atribute private, diagrame de tranziţii de stări şi rutine de tratare a evenimentelor [Cro2001].

Utilizarea conceptului de interactor permite proiectantului, pe de-o parte, formalizarea sub forma de diagrame de stare si pe de alta parte permite descompunerea unei interfeţe cu utilizatorul în componente similare, sau alternativ, compunerea unei interfeţe cu utilizatorul din componente. Aceste componente au aceeaşi structură generică, dar pot fi adaptate pentru a oferi funcţionalităţi diferite în cadrul unui spaţiu comportamental larg. Acesta este principalul motiv de utilizare a abstractizării sub forma de interactori. In plus, interactorii

oferă o vedere abstractă asupra obiectelor concrete de interacţiune existente în colecţiile de instrumente, şi prin urmare pot fi folosiţi pentru proiectarea şi analiza abstractă a structurii şi comportamentului interfeţei.

Se pot identifica trei roluri care corespund celor trei părţi principale ale comportamentului şi funcţiei unui interactor [Gul2004]:

- *medierea informaţiei*: interfaţa cu utilizatorul poate fi privită ca şi un mediator de informaţii dintre sistem şi utilizatorul său, precum şi un instrument de efectuare a acţiunilor. Fiecare interactor defineşte informaţia pe care o mediază şi direcţia în care aceasta este transferată.
- *comanda şi activarea*: interactorii au un spaţiu intern al stărilor care le controlează comportamentul, inclusiv transmiterea şi recepţia informaţiei, activarea altor interactori şi iniţierea efectuării unor funcţii specifice aplicaţiei;
- *structurarea compoziţională*: interactorii se pot compune în noi interactori, pentru a permite construirea unui comportament de nivel mai înalt, precum şi pentru abstractizarea detaliilor.

O interfaţă cu utilizatorul poate fi văzută ca şi un *canal de comunicaţie dintre utilizator si sistem*. După cum este ilustrat în figura 3-6, informaţia poate trece prin canal în oricare sens. Informaţia ce trece de la sistem către utilizator indică de obicei *starea sistemului*, inclusiv acţiunile disponibile utilizatorului. Informaţia de la utilizator la sistem se poate constitui *în acţiuni ce conţin date, cu menirea de a modifica starea sistemului*. Modelul transferului de informaţie depinde de modul de dialog al aplicaţiei, dar în mod obişnuit va alterna între sistem-utilizator şi utilizator-sistem.

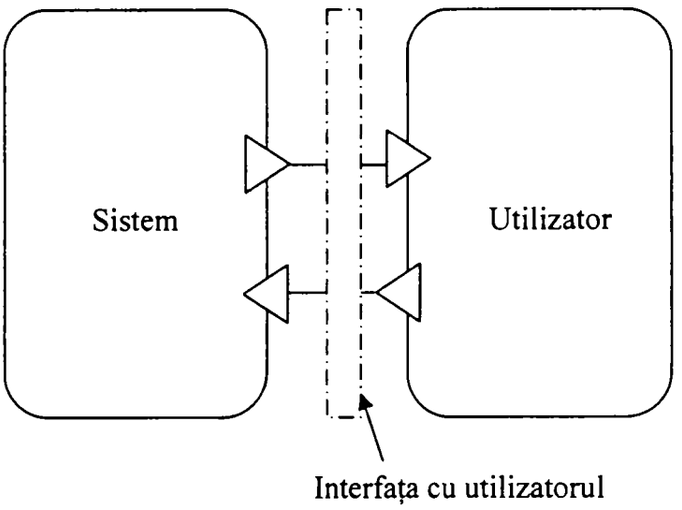


Figura 3-6. Comunicaţia sistem-utilizator

Un model al dialogului trebuie să descrie structura şi comportamentul interfeţei cu utilizatorul, astfel că este necesară explicitarea acestui canal de comunicaţie.

In figura 3-7, modelului i se atribuie un rol mult mai activ decât acela de *mediator* al informaţiei. Sistemul poate furniza informaţie utilizatorului prin transmiterea acesteia interfeţei cu utilizatorul, care la rândul ei o transmite într-o formă corespunzătoare utilizatorului şi dispozitivului de ieşire. Utilizatorul poate introduce informaţii în sistem prin transmiterea lor către interfaţa cu utilizatorul, care la rândul ei le va transmite sub o formă adecvată sistemului.

Transmiterea informaţiei dinspre şi spre sistem se bazează pe modelul sarcinii [Pri2001]. Totuşi, transferul informaţiei dintre interfaţa cu utilizatorul şi utilizator trebuie să se bazeze pe dispozitivele platformei de implementare: *de ieşire* (ieşirea poate fi text, grafică, imagini etc.), şi *de intrare* (introducerea de la tastatură, clicuri de maus etc.). Datorită faptului că acestea sunt în afara domeniului unui model abstract ele vor fi ignorate în modelarea dialogului, după cum este ilustrat în figura 3-8. Totuşi, trebuie modelat şi transferul informaţiei către şi dinspre *părţi ale* interfeţei în direcţia utilizatorului, pentru că acestea fac parte din structura şi comportamentul care trebuie cuprinse în modelul abstract de dialog.

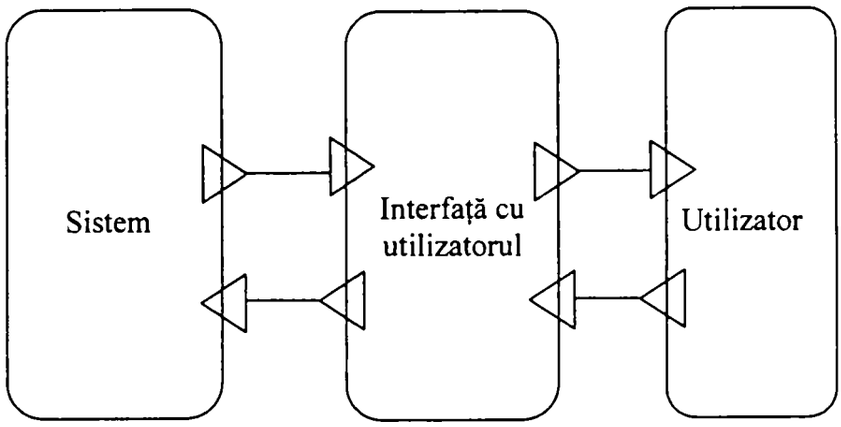


Figura 3-7. Medierea dintre sistem şi utilizator

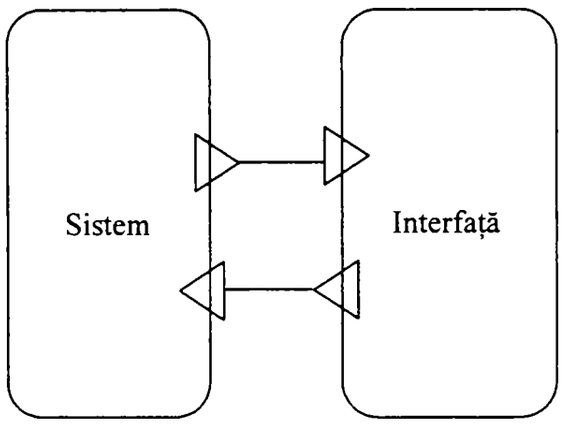


Figura 3-8. Interfaţa cu utilizatorul

Procesul de mediere a informaţiei realizat de către interactori şi respectiv de structurile de interactori este definit de:

- un set de porţi care definesc interfaţa externă a interactorului cu sistemul, utilizatorul şi alţi interactori;
- un set de conexiuni care transmit date între porţi;
- structura de comandă care iniţiază sau blochează transferul informaţiei între porţile interne, externe şi de-a lungul conexiunilor.

Împreună, acestea oferă transfer de informaţie între interactori, precum şi sincronizarea şi activarea interactorilor.

După cum se poate observa în figura 3-9, există 2x2 tipuri diferite de porţi, de transmitere şi recepţie, în fiecare din cele 2 sensuri. Denumirile şi rolurile lor sunt următoarele:

- *intrare/transmitere:* intrări ca rezultat al interacţiunii cu dispozitivele de

introducere, rezultă transmitere de informaţie dinspre interactor spre sistem;

- *ieşire/recepţie*: ieşire de informaţie din sistem spre interactor;
- *intrare/recepţie*: informaţia de la utilizator spre sistem, este transmisă către interactor pentru procesare/mediere;
- *ieşire/transmitere*: informaţia de la sistem este transmisă din interactor spre utilizator.

Interactorul reprezintă o agregare a unui set de porţi. Conceptul de *poartă,* în acest caz, se bazează pe conceptul de funcţie, ce preia un set de argumente de intrare şi determină o valoare de ieşire rezultat, pe baza unei atribuiri de funcţii [Mar1996].

3.2.1.3 Dispozitive interactor

Majoritatea obiectelor concrete de interacţiune (check-box, edit-box etc.) sunt *dispozitive interactor*, utilizate pentru introducerea şi extragerea de tipuri specifice de date, în interacţiunea directă cu utilizatorul.

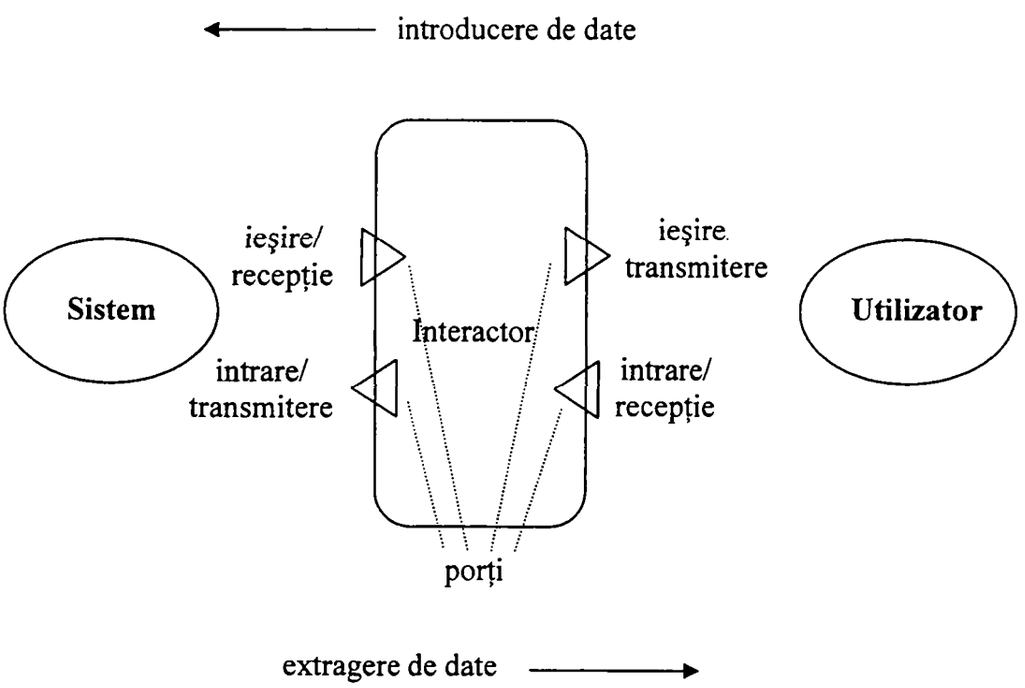


Figura 3-9. Interactor generic

Cel mai simplu dispozitiv interactor este cel de introducere şi extragere a valorilor booleene, pentru care echivalentul tipic este *check-box*-ul. După cum este ilustrat în figura 3-

10, *check-box*-ul are două stări care corespund celor două valori posibile, adevărat şi fals, indicate de obicei de o cruce sau un *check-mark*. Eticheta *check-box*-ului este utilizată la asocierea valorii booleene cu interpretarea în funcţie de modelul domeniului şi nu afectează comportamentul *check-box*-ului.

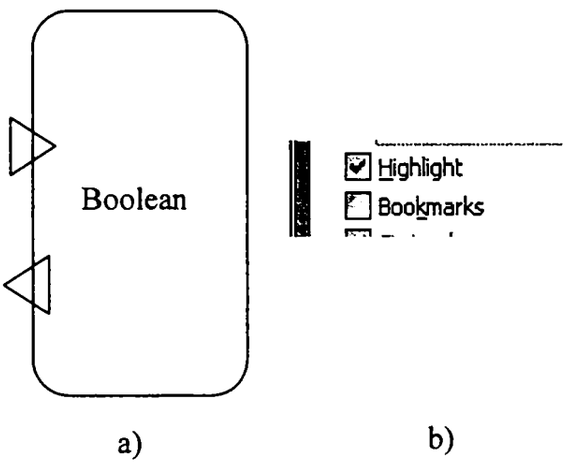


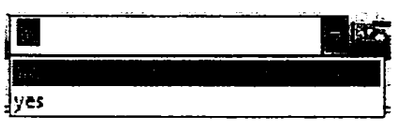
Figura 3-10. a) Dispozitivul interactor boolean şi b) obiectul de interacţiune concretă corespunzător

Aceeaşi funcţie cu cea a unui *check-box* poate fi obţinută prin utilizarea unui buton cu pictogramă, în care stările de apăsat şi depresat indică starea de adevărat sau fals. Butonul are acelaşi comportament comutator ca şi *check-box*-ul.

In figura 3-11, cele trei atribute (independente) de text - îngroşat, cursiv şi subliniat - sunt controlate printr-un set de butoane cu pictograme, fiecare buton implementând un singur interactor boolean. Principalul avantaj faţă de *check-box*-uri este acela al spaţiului redus de reprezentare, iar pictograma este mai uşor de recunoscut decât eticheta.

Butoane pictograme

Radio-Buttons



Drop-List

Figura 3-11. Elemente concrete care pot implementa dispozitivul interactor boolean

O soluţie alternativă este aceea de utilizare a unei perechi de *radio-button*-uri sau o listă drop-down cu două elemente, ca un caz special de selecţie a unei opţiuni din cadrul unui set de opţiuni (mutual exclusive).

In ambele cazuri, cele două stări sunt indicate de expresii diferite cu semnificaţii opuse.

Exemplul considerat este tipic pentru modul în care aceeaşi funcţie abstractă, din cazul modelului ilustrat în figura 3-10, poate fi implementată prin utilizarea de obiecte diferite de interacţiune concretă.

Alegerea unui anumit obiect de interacţiune concretă depinde de mai mulţi factori cum ar fi [Eis2000] [Mar1996]:

- modelul mental al utilizatorului cu privire la domeniu (ex.: este adevărat/fals mai intuitiv decât da/nu sau activ/inactiv?);
- capabilităţile utilizatorului (ex.: evitarea utilizării combinaţiei de culori verde/roşu de indicare a pornirii/opririi în cazul daltoniştilor);
- cerinţele de spaţiu (ex.: *check-box*-urile necesită mai puţin spaţiu decât *radio-button*-urile).

Toate aceste obiecte de interacţiune utilizează spaţiul de ecran şi clicurile de mouse ca şi resurse principale.

Structura de comandă este implicată în iniţierea transmiterii informaţiei şi respectiv activarea şi dezactivarea interactorilor.

Ca limbaj de modelare a ei s-a ales utilizarea diagramelor de stare. Fiecare interactor poate fi considerat o super-stare din cadrul diagramei de stare, în care se trece la activarea interactorului corespunzător. Alternativ, la accesarea unui interactor, acesta este considerat activ, şi poate răspunde la informaţia propagată prin porţile sale intrare/recepţie şi ieşire/recepţie şi poate emite date prin porţile intrare/transmitere şi ieşire/transmitere.

Interactorul poate fi interpretat ca un sistem cu evenimente discrete astfel că el poate fi descompus în stări conform limbajului diagramelor de stare. In figura 3-12 a) este

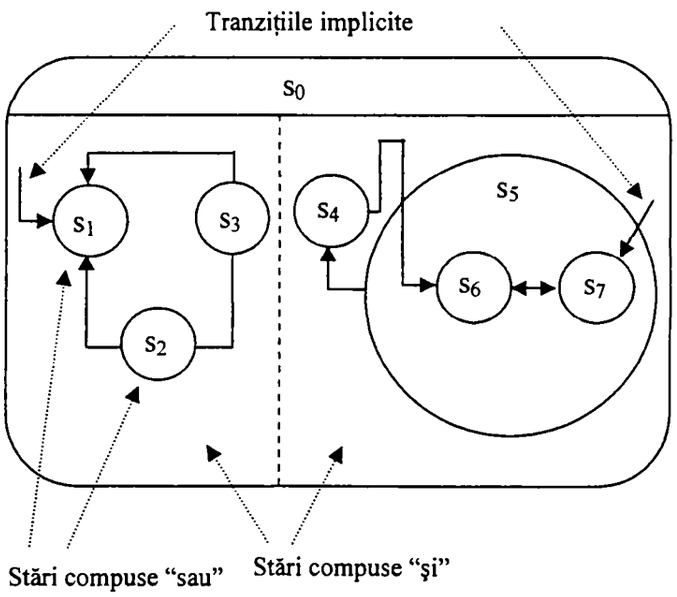
reprezentată descompunerea unui interactor în stările posibile iar în figura 3-12 b) sunt reprezentate aceleaşi stări sub forma unei diagrame arborescente de stare.

Tranziţiile pot fi etichetate cu evenimentele care le iniţiază, cu condiţia care controlează iniţierea sau cu acţiunile care sunt efectuate ca rezultat al parcurgerii tranziţiei (v. 2.5.1).

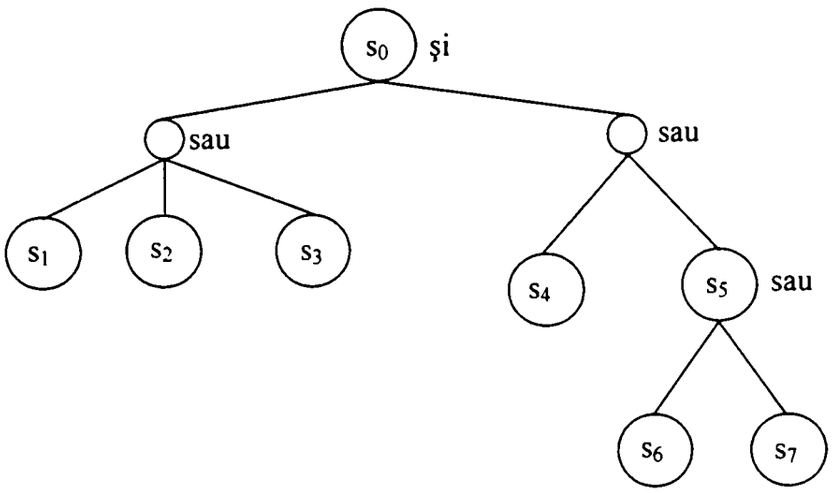
Notaţia utilizată este *eveniment[condiţie]/acţiune*.

Dacă nu este specificat nici un eveniment, atunci tranziţia se iniţiază întotdeauna în momentul în care condiţia este satisfăcută. Condiţia este implicit îndeplinită astfel că dacă nu se specifică nici o condiţie, evenimentul singur va iniţia tranziţia. Acţiunea este un eveniment care poate iniţia alte tranziţii din cadrul maşinii de stare, sau este o funcţie specifică aplicaţiei.

O tranziţie poate depăşi graniţele unei stări şi poate conduce doar la o sub-stare a unei compoziţii "sau". Stările pot fi referite şi în cadrul evenimentului şi în cadrul condiţiei.



a)



b)

Figura 3-12. Descompunerea unui interactor în a) stările posibile şi b) diagramă arborescentă de stare

Conform consideraţiilor anterioare, *check-box*-ul din figura 3-10 poate fi modelat prin diagrame de tranziţii de stare. Prin observarea lui vizuală şi comportamentală, se poate deduce modelul de (dispozitiv) interactor cu diagrame de stare din figura 3-13. Caracteristicile principale sunt următoarele:

- porţile ce reprezintă valorile de intrare şi ieşire sunt în partea stângă;
- starea de comandă în mijloc;
- reprezentarea vizuală in dreapta.

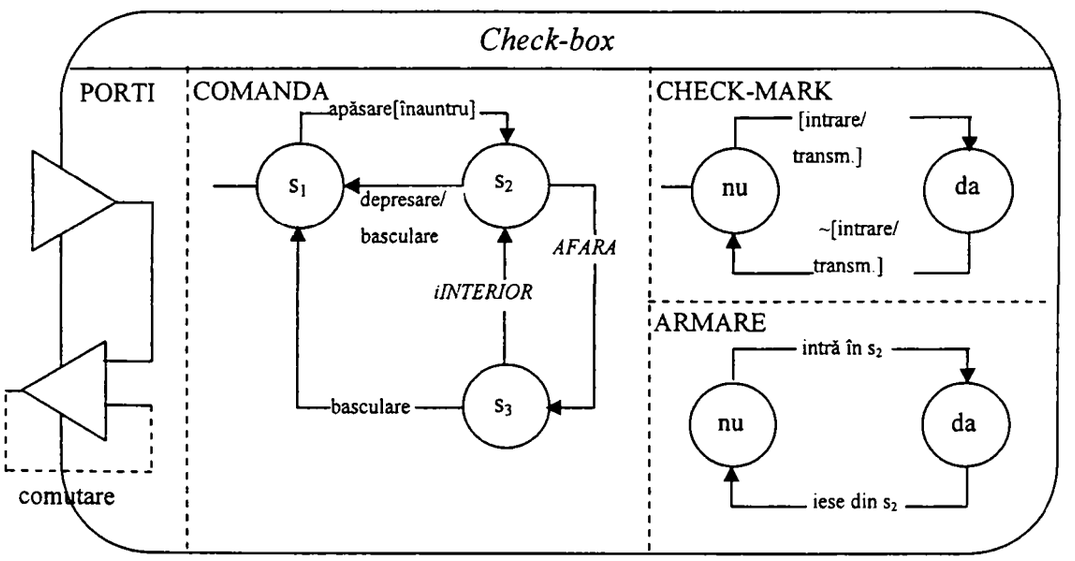


Figura 3-13. Modelul de (dispozitiv) interactor cu diagrame de stare pentru *check-box*

Cele trei componente ale interactorului sunt compuse "şi", adică operează în paralel şi se coordonează/sincronizează prin evenimente şi condiţii. Componenta cu porţi este obligatorie la toţi interactorii, în timp ce celelalte două componente sunt specifice fiecărui interactor în parte. Componenta PORTI defineşte modul în care sunt propagate şi sincronizate valorile de intrare şi ieşire, în cazul de faţă valoarea intrare/transm. este conectată, şi astfel egală, cu valoarea ieşire/recepţie. Dacă se efectuează acţiunea de comutare, se determină o nouă valoare de intrare de către o conexiune complementară (linia întreruptă semnifică un invertor). Acţiunea de comutare este iniţiată de starea de comandă, la apăsarea unui buton de mouse (tranziţia de la s_1 la s_2) şi în continuare depresarea butonului (tranziţia de la s_2 la s_1). Indicatorul mouse-ului trebuie să fie în interiorul *check-box*-ului în momentul apăsării şi depresării, dar poate fi afară între aceste evenimente.

In general o conexiune poate fi etichetată printr-un eveniment dar şi printr-o condiţie care indică momentul de timp în care se va propaga sau sincroniza o valoare. Evenimentele sunt utilizate la iniţierea tranziţiilor iar condiţiile la limitarea situaţiilor în care poate fi iniţiată tranziţia [Sti1999]. Reprezentarea vizuală este controlată de stările CHECK-MARK şi ARMARE, în care ultima este utilizată la indicarea comutării cu succes în momentul depresării butonului (diferenţa dintre stările s_2 şi s_3).

Două aspecte ale acestei diagrame de stare sunt implicite: semnificaţia evenimentelor şi condiţiilor externe, cum ar fi: apăsare, depresare şi în interior, precum şi corespondenţa dintre stările activ/inactiv şi reprezentarea vizuală a obiectului de interacţiune. Pentru a explicita un model şi a-l face executabil, este necesară conectarea diagramei de stare la dispozitivul mouse şi la funcţia de desenare în fereastră, moment în care acest model poate fi considerat o implementare completă a dispozitivului boolean de intrare/ieşire.

Acte şi funcţii

Un *act* reprezintă intenţia utilizatorului ca sistemul să efectueze o funcţie pentru a atinge un scop, sau mai concret, de aplicare a unei funcţii asupra unui set de argumente.

Un act corespunde de obicei scopului unei sarcini a utilizatorului, chiar dacă acest scop nu este modelat explicit, fie pentru că modelul sarcinii este prea detaliat, fie pentru că scopurile nu sunt modelate explicit.

Actele sunt utilizate pentru reprezentarea explicită a unei funcţii şi a unui set de argumente sub forma unui singur element, fie pentru că nu sunt specifice domeniului, fie

pentru că nu au relevanţă pentru utilizator. Acest lucru este tipic pentru serviciile standard de sistem, cum sunt interogări de director sau interpretarea şirurilor, sarcini "luate de bune" [Pri2001] [Mar1997].

Datorită faptului că fiecare act poate fi tratat de mai multe funcţii, funcţionalitatea efectivă a unui act este o combinaţie a mai multor componente ale unui model. Ordinea execuţiei acestor componente este importantă pentru rezultatul final, astfel că mecanismul de difuzare trebuie să fie predictibil.

In cazul diagramelor de stare clasice, evenimentele sunt difuzate către toate stările şi poate fi iniţiată orice tranziţie. Stările exterioare au *prioritate*.

3.2.2. Modelarea elementelor de interacţiune concretă

Dialogul abstract, reprezentat anterior printr-un model de interactor, poate fi privit ca o specificaţie de implementare în termeni de obiecte specifice de interacţiune. La trecerea de la specificaţia abstractă la implementarea concretă, funcţionalitatea dorită în cazul implementării concrete a interfeţei cu utilizatorul se obţine exploatând capabilităţile obiectelor de interacţiune [Van2001].

Modelul obiectului nu este obligatoriu o descriere completă a capabilităţilor lui, el reprezentând una din multele abstractizări posibile.

Obiectele informaţionale, reprezentând intrarea sau ieşirea unui interactor, sunt de obicei date ale domeniului, descrise în cadrul modelului sarcinii. Totuşi, nu doar datele domeniului ci şi meta-datele pot fi manipulate în cadrul interfeţei cu utilizatorul. Oricare din modelele interfeţei cu utilizatorul poate fi considerat ca o modalitate de introducere sau extragere de informaţii.

Denumirea acţiunilor utilizate în modelul sarcinilor sunt folosite pentru etichetarea elementelor orientate pe acţiuni [Ben1996], cum ar fi butoanele sau elementele de meniu. Barele de instrumente grupează instrumentele conform structurii sarcinii, iar "experţii" suportă sarcini ce au fost identificate ca importante în cadrul analizei sarcinilor.

Modelul domeniului este utilizat la identificarea datelor prezentate (ex.: în cazul completării formularelor, etichetele text sunt utilizate la prezentarea de nume de atribute din cadrul modelului domeniului, la stânga sau deasupra valorilor atributelor propriu-zise).

Cele două modele orientate pe soluţie, dialogul şi interacţiunea concretă, pot fi utilizate pentru personalizarea interfeţei (ex.: dimensionarea vederilor, barele de instrumente, scurtăturile de tastatură).

Butonul, obiect de interacţiune de bază

Etichetele text ca obiecte concrete de interacţiune sunt utilizate doar pentru prezentarea informaţiei. Butoanele sunt de asemenea elemente interactive, şi sunt utilizate pentru activarea sau dezactivarea unei funcţii, prin comutarea dintre stările de apăsat şi depresat. Atunci când sunt utilizate cu scopul de *comandă*, tranziţia este iniţiată de obicei de către un buton hardware (ex.: butonul stâng al mouse-ului, tasta *enter*).

Cele două stări pot fi utilizate şi la medierea informaţiei, unde stările de apăsat şi depresat corespund valorilor booleene adevărat şi fals, de exemplu pe baza valorii unui atribut, pe baza unei clasificări de elemente sau pe existenţa unei relaţii.

Funcţionarea unui buton poate fi descrisă printr-o diagramă de tranziţii a stărilor interne aşa cum este reprezentat în figura 3-14.

Unui buton îi corespund două stări de bază (apăsat şi depresat). In practică însă, sunt necesare câteva stări intermediare adiţionale utilizate pentru reacţia către utilizator. Figura 3-14 prezintă şi stările orientate spre prezentare: „interior" şi „bordură". Acestea sunt conduse de stările de „comandă", care limitează secvenţele admise de stări la cele care oferă comportamentul interactiv dorit. Resursa stare/condiţie „apăsat" coordonează starea „comandă" şi permite legătura cu o stare exterioară (buton de mouse, tastatură sau alt tip de buton fizic).

La o implemetare tipică a unui buton, resursa „apăsat" este legată de butonul stâng a mouse-ului şi de tasta spaţiu, ultima necesitând ca focusul tastaturii să fie pe buton.

Secvenţa tipică de stări este „inactiv", „activ.depresat", „activ.apăsat", „activ.depresat" şi „inactiv". Aceasta corespunde cu deplasarea cursorului de mouse deasupra elementului buton, apăsarea şi depresarea butonului de mouse şi deplasarea cursorului în afara butonului.

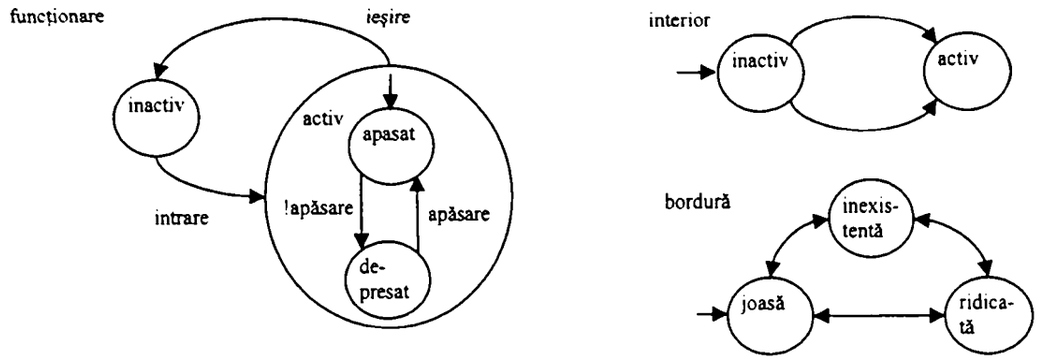


Figura 3-14. Diagrama de tranziţii de stare a butonului

Uzual, există două variante de reprezentare a unui buton:

- bordura este ridicată instantaneu după apăsarea butonului, revenind la starea „depresat". Această variantă este utilizată mai ales pentru iniţierea altor tranziţii la intrarea în starea „activ.depresat". Apăsarea butonului poate fi considerată ca un act, de exemplu "Salvare", orientat către aplicaţie sau către contextul unui buton (documentul curent);
- butonul simbolizează o valoare booleană care este comutată/inversată la depresarea butonului. Butonul rămâne în această stare până la următoarea apăsare sau la manipularea prin program. In acest caz butonul reprezintă starea domeniului, cum ar fi atributul "îngroşare" a unui bloc de text, iar apăsarea reprezintă comutarea stării. La modificarea contextului butonului, prin mutarea cursorului sau selectarea unei alte regiuni de text, butonul trebuie actualizat pentru a reflecta starea noului context.

Interpretarea acestor două variante ca fiind orientate pe comandă sau acţiune trebuie să se reflecte în cadrul modelului interactorului corespunzător.

Interactorul echivalent unui buton în care interpretarea informaţiei poate fi indicată prin adăugarea porţilor output/receive şi input/send este reprezentat in figura 3-15. Vârful porţii input/send trebuie conectat prin utilizarea unei conexiuni complementare la propria bază.

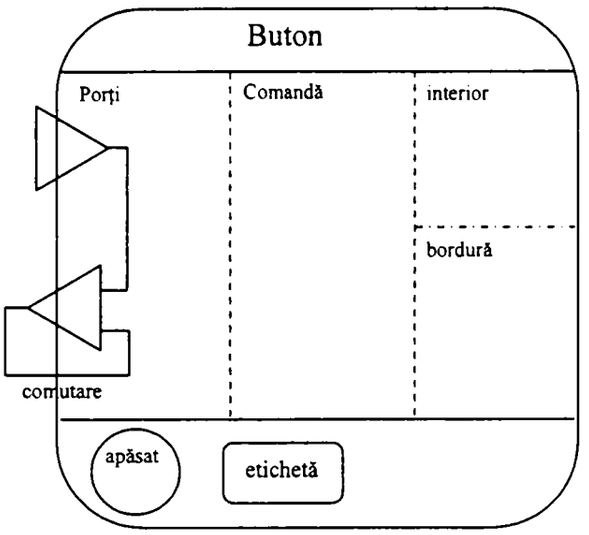


Figura 3-15. Model de interactor echivalent butonului

In tabelul 3-2 prezintă corespondenţele dintre comanda butonului şi stările aferente.

Tabelul 3-2. Comanda butonului şi stările aferente

comandă / valoare	**inactiv**	**activ.depresat**	**activ.apăsat**
adevărat	interior.nu bordură.scăzută	interior.da bordură.scăzută	interior.da bordură.scăzută
fals	interior.nu bordură.ridicată	interior.nu bordură.ridicată	interior.da bordură.scazută

3.3. Modelarea ca SED a unui proces standard de selecţie de obiecte

Modelarea sarcinii şi a dialogului permite proiectantului de interfeţe cu utilizatorul identificarea, analiza şi înţelegerea principalelor elemente şi caracteristici necesare proiectării unei interfeţe corespunzătoare, prin realizarea unei formalizări a interacţiunii cu utilizatorul.

Datorită faptului că interfaţa cu utilizatorul prezintă toate caracteristicile unui sistem dinamic cu evenimente discrete, aceasta poate fi modelată ca atare, rezultând o descriere formală (concisă, completă şi precisă/adecvată) a comportamentului acesteia.

In cele ce urmează, este prezentată modelarea ca SED doar a procesului standard de selecţie de obiecte din cadrul interfeţei cu utilizatorul pentru un sistem CAD, realizată în cap. 4. Pentru celelalte subsisteme din cadrul interfeţei cu utilizatorul (gestionarea unor cutii de

dialog, elemente personalizate de interfaţă, etc.) pot fi realizate modelări ca SED asemănătoare.

Standardizarea procesului de selecţie trebuie realizată atât din punctul de vedere al utilizatorului cât şi din cel al programatorului. In acest sens este necesar ca toate situaţiile care pot apărea în cazul unui proces de selecţie să fie luate în considerare şi tratate într-o manieră simplă şi uniformă.

Procesul de selecţie standard de obiecte permite selectarea a unui singur, sau a doi operanzi asupra căruia/cărora se va aplica o operaţie. Un operand este un set de obiecte valide, selectate pe ecran de către utilizator [Sav2004b].

In urma analizei procesului de selecţie (analiza sarcinii), prin aplicarea conceptelor definite în cadrul teoriei sistemelor cu evenimente discrete (v. paragraful 2.5.1) şi prin formalizare, utilizând tehnicile de modelare a sarcinii si a dialogului (abstract), acest proces poate fi modelat sub forma unui SED.

Pornind de la definiţia unui SED, în continuare se identifică structura elementelor sextuplului $(X, E, f, \Gamma, x_0, X_M)$.

Mulţimea stărilor (X) si semnificaţia lor este prezentată în tabelul 3-3:

Tabelul 3-3 Mulţimea stărilor (X) si semnificaţia lor

Stare	**Comentariu**
S1	Starea iniţială in care se afla SED
S2	obiectele active se adaugă primului grup de obiecte selectate
S3	se indică pe ecran, de către utilizator, primul obiect pentru primul grup de selecţie
S4	un posibil obiect candidat pentru primul grup de obiecte selectate este prezentat utilizatorului pentru confirmare
S5	obiectul confirmat se adaugă la primul grup de obiecte selectate
S6	starea finală, în care cele două grupuri de obiecte de selecţie conţin obiectele necesare efectuării operaţiei indicate anterior de către utilizator
S7	se indică pe ecran, de către utilizator, un obiect adiţional pentru primul grup de obiecte selectate
S8	un posibil obiect adiţional, candidat pentru primul grup de obiecte selectate, este prezentat utilizatorului pentru confirmare

S9	obiectele adiţionale confirmate se adaugă la primul grup de obiecte selectate
S10	se indică pe ecran, de către utilizator, primul obiect pentru cel de-al 2-lea grup de selecţie
S11	un posibil obiect candidat pentru cel de-al doilea grup de selecţie este prezentat utilizatorului pentru confirmare
S12	obiectul confirmat se adaugă la al doilea grup de obiecte selectate
S13	se indica pe ecran, de către utilizator, un obiect adiţional pentru primul grup de obiecte selectate
S14	un posibil obiect adiţional, candidat pentru cel de-al doilea grup de obiecte selectate, este prezentat utilizatorului pentru confirmare
S15	obiectele adiţionale confirmate se adaugă la al doilea grup de obiecte selectate
S16	cele două grupuri de obiecte selectate sunt golite

Tranziţiile între stările SED-ului sunt iniţiate de apariţia unor evenimente, în combinaţie sau nu cu anumite condiţii. Mulţimea evenimentelor (E) este prezentată în tabelul 3-4:

Tabelul 3-4 Mulţimea evenimentelor (E)

Ev.	**Comentariu**
e_1	active_elements.true AND selectionMode.oneGroup – iniţiază tranziţia din starea S1 in starea S2. Evenimentul este activat de îndeplinirea simultană a condiţiei de existentă a unor obiecte activate şi a modului de selecţie OneGroup
e_2	active_elements.false OR !selectionMode.oneGroup – iniţiază tranziţia din starea S1 in starea S3 dacă nu există obiecte active sau modul de selecţie nu este OneGroup
e_3	keyboard.ESC OR mouseRightClick – iniţiază tranziţia din S3 în S1, din S10 în S1, atunci când se apasă tasta ESC sau butonul din dreapta al mouse-ului
e_4	selectElem.invalid – iniţiază tranziţia din S3 tot în S3 dacă obiectul selectat nu corespunde criteriului de selecţie
e_5	selectElem.valid – iniţiază tranziţia din S3 în S4, din S7 în S8, din s10 în S11, din S13 în S14, daca obiectul selectat satisface criteriile de selecţie
e_6	browseSelectedCandidates.true – iniţiază tranziţia din S4 îin S4 sau din S8 în S8, din S11 în S11, din S14 în S14, dacă obiectul oferit spre confirmare este refuzat şi mai sunt obiecte candidate
e_7	browseSelectedCandidates.false – iniţiază tranziţia din S4 în S3, din S8 în S7, din

	S11 în S10, din S14 în S13, daca obiectul oferit spre confirmare este refuzat şi nu mai există obiecte candidate
e_8	mouseLeftClick – iniţiază tranziţia din S4 în S5, din S8 în S9, din S11 în S12, din S14 în S15, la apăsarea butonului stâng al mouse-ului
e_9	selectionMode.oneElem – iniţiază tranziţia din S5 in S6 daca modul de selecţie este OneElem
e_{10}	selectionMode.oneGroup – iniţiază tranziţia din S5 în S7 dacă modul de selecţie este OneGroup
e_{11}	selectionMode.onePair – iniţiază tranziţia dintre S5 şi S10 dacă modul de selecţie este OnePair
e_{12}	selectTwoGroups.true – iniţiază tranziţia dintre S7 şi S10 la apăsarea butonului drept de mouse şi modul de selecţie este TwoGroups
e_{13}	selectTwoGroups.false OR keyboard.ESC – iniţiază tranziţia din S7 în S1 dacă s-a apăsat butonul drept al mouse-ului şi nu este modul de selecţie TwoGroups sau s-a apăsat tasta ESC
e_{14}	!selectionMode.twoGroups – iniţiază tranziţia din S12 în S1dacă modul de selecţie nu este TwoGroups
e_{15}	selectionMode.twoGroups – iniţiază tranziţia din S12 în S13 daca modul de selecţie este TwoGroups
e_{16}	MouseRightClick – iniţiază tranziţia din S13 în S1 dacă s-a apăsat butonul drept al mouse-ului
e_{17}	Keyboard.ESC – iniţiază tranziţia din S13 în S16 dacă s-a apăsat tasta ESC
*	Condiţia de tranziţie este întotdeauna adevărata

Funcţia de tranziţie este definită în modul următor:

$f: X \times E \rightarrow X$

$f(S1, e_1) = S6$

$f(S1, e_2) = S3$

$f(S2, *) = S6$

$f(S3, e_4) = S3$

$f(S3, e_3) = S1$

$f(S3, e_5) = S4$

$f(S4, e_6) = S4$

$f(S4, e_8) = S5$

$f(S4, e_7) = S3$

$f(S5, e_9) = S6$

$f(S5, e_{10}) = S7$

$f(S5, e_{11}) = S10$

$f(S7, e_{12}) = S10$

$f(S7, e_{13}) = S1$

$f(S7, e_5) = S8$

$f(S8, e_6) = S8$

$f(S8, e_8) = S9$

$f(S8, e_7) = S7$

$f(S9, *) = S6$

$f(S10, e_3) = S1$

$f(S10, e_5) = S11$

$f(S11, e_6) = S11$

$f(S11, e_8) = S12$

$f(S11, e_7) = S10$

$f(S12, e_{14}) = S6$

$f(S12, e_{15}) = S13$

$f(S13, e_{16}) = S6$

$f(S13, e_{17}) = S16$

$f(S13, e_5) = S14$

$f(S14, e_6) = S14$

$f(S14, e_8) = S15$

$f(S14, e_7) = S13$

$f(S15, *) = S6$

$f(S16, *) = S1$

Funcţia de evenimente fezabile $\Gamma : X \rightarrow 2^E$ are următoarea definiţie:

$\Gamma(S1) = \{e_1, e_2\}$

$\Gamma(S2) = *$

$\Gamma(S3) = \{e_4, e_3, e_5\}$

$\Gamma(S4) = \{e_6, e_8, e_7\}$

Γ (S5) = {e_9, e_{10}, e_{11}}

Γ (S6) = 0

Γ (S7) = {e_{12}, e_{13}, e_5}

Γ (S8) = {e_6, e_8, e_7}

Γ (S9) = {*}

Γ (S10) = {e_3, e_5}

Γ (S11) = {e_6, e_8, e_7}

Γ (S12) = {e_{14}, e_{15}}

Γ (S13 = {e_{16}, e_{17}, e_5}

Γ (S14 = {e_6, e_8, e_7}

Γ (S15) = {*}

Γ (S16) = {*}

Starea iniţială este X_0 = S1.

Mulţimea stărilor marcate este X_M = {S1, S6} unde S1 este starea iniţială şi S6 stare finală.

Diagrama tranziţiilor de stare a procesului de selecţie standard modelat ca SED este prezentată în figura 3-16.

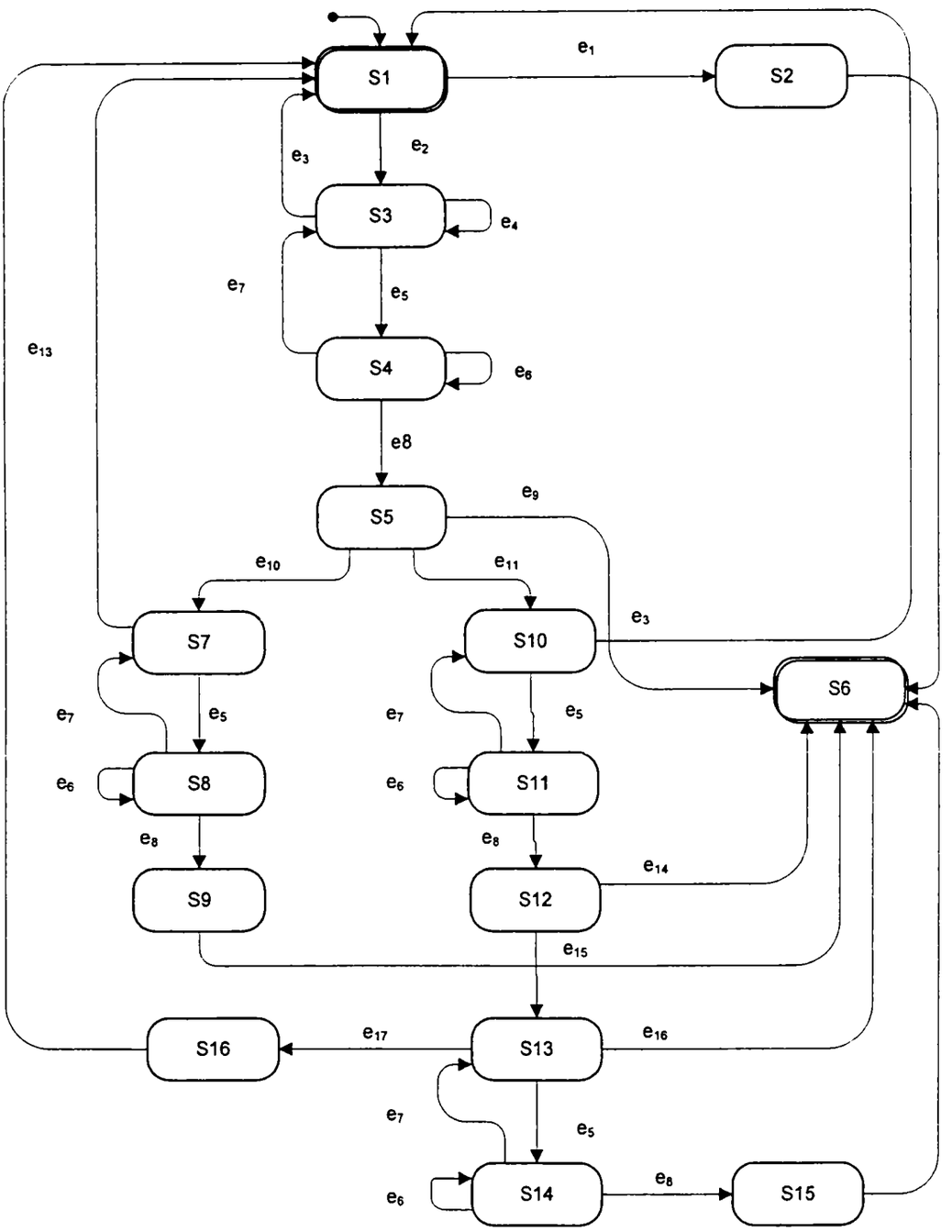


Figura 3-16. Diagrama tranziţiilor de stare pentru procesul de selecţie standard modelat ca SED

La implementarea funcţiei de selecţie în cadrul aplicaţiei dezvoltate în capitolul 4, diagramele schemelor logice au fost realizate pe baza diagramei tranziţiei stărilor a SED modelat din figura 3-16, prin corelarea stărilor, evenimentelor şi a condiţiilor cu apelurile de proceduri şi funcţii ale aplicaţiei, cu acţiunile utilizatorului şi respectiv cu stările variabilelor

interne ale aplicaţiei. Alegerea elementelor concrete de interfaţă se face pe baza specificaţiilor de implementare rezultate din modelul sarcinii, a modelelor de dialog abstract şi a modelelor elementelor de interacţiune concretă.

Modelări asemănătoare ca SED au fost realizate şi pentru celelalte subsisteme ale interfeţei cu utilizatorul pentru sistemul CAD Dietrich's prezentat în cap. 5.

3.4. Concluzii şi contribuţii

Modelarea reprezintă o componentă importantă a proiectării inginereşti şi este utilizată la analiza comportamentului, evaluarea performanţelor şi optimizarea sistemelor. Sistemele cu evenimente discrete constituie o clasă aparte de sisteme dinamice neliniare, care necesită pentru investigare instrumente proprii de modelare şi analiză, complet diferite de cele utilizate la sistemele clasice (continuale/discrete) bazate pe ecuaţii diferenţiale sau ecuaţii cu diferenţe finite. Scopul analizei SED *este dezvoltarea de modele* corespunzătoare, care descriu în mod adecvat comportamentul acestor sisteme.

Sintetizarea principalelor concepte din cadrul domeniului sistemelor cu evenimente discrete a demonstrat importanţa acestui domeniu ca suport de modelare ca SED a interfeţei cu utilizatorul.

Prezentarea critică a principalelor formalisme din cadrul modelării interfeţelor cu utilizatorul precum şi analiza critică a trei modele principale de interfaţă cu utilizatorul (sarcină/domeniu, dialog abstract, interacţiune concretă) şi a conceptului de interactor a permis descrierea aspectelor relevante ale interfeţei, prezentând cunoştinţele asupra interfeţei sub o formă abstractă utilizabilă.

Utilitatea acestei abordări este demonstrată prin modelarea ca SED a unei sarcini "selecţia standard de obiecte", prin identificarea stărilor, a stării iniţiale, a stărilor marcate şi a mulţimii evenimentelor, definirea funcţiei de tranziţie şi a funcţiei eveniment activ şi respectiv elaborarea diagramei de tranziţii de stare, constituindu-se ca etapă premergătoare realizării schemei logice de funcţionare dezvoltate în cadrul capitolului 5.

4. Tehnologii software utilizate în realizarea de interfețe cu utilizatorul

4.1. Introducere

In cadrul procesului de proiectare a unei interfețe performante cu utilizatorul este nevoie de o cunoastere în profunzime a domeniului tehnologiilor software actuale, în scopul de a identifica şi dezvolta acele elemente specifice care sunt cele mai adecvate aplicațiilor propuse.

S-au selectat trei tipuri de tehnologii, acestea constituindu-se în sursele cele mai cuprinzătoare privind soluțiile tehnologice destinate realizării de interfețe cu utilizatorul de înaltă performanță corespunzător obiectivului prioritar declarat al acestei lucrări:

- Programarea orientată spre obiecte.
- Programarea elementelor de interfață cu utilizatorul.
- Programarea vizualizării utilizând biblioteca grafică OpenGL.

Odată cu apariția limbajelor de programare, firesc s-a trecut la realizarea metodelor de analiză şi proiectare care au condus la optimizarea procesului de dezvoltare a produselor software.

Programarea structurată este orientată spre funcții, tratând entitățile de date separat de funcții. In urma creşterii complexității produselor software a fost necesară o restructurare a conceptelor ce stau la baza dezvoltării acestora. Astfel se explică apariția modelului de proiectare orientat spre obiecte.

Paragraful 4.2 justifica alegerea unui limbaj de programare orientat spre obiecte, sintetizand etapele esentiale de dezvoltare ale unui produs software: analiza domeniului, analiza cerințelor sistemului, proiectarea sistemului, analiza cerințelor software, proiectarea software şi implementarea. Secțiunea dedicată implementării software tratează în detaliu aspecte concrete privind utilizarea template-urilor, prototipizarea şi implementarea unei componente reutilizabile.

Elementele de interfață sunt diferențiate în funcție de tipul de informație pe care îl transferă între utilizator si sistem. Ca urmare, în cadrul procesului de proiectare s-au selectat şi, după caz, s-au adaptat, dezvoltat si proiectat elemente de interfață corespunzătoare

cerințelor produselor software dezvoltate în capitolul 5. Paragraful 4.3 tratează în detaliu, în cadrul interfețelor cu utilizatorul, probleme legate de utilizabilitate şi anume: cardinalitatea, feedback-ul, modele explicite de utilizator, suportul pentru planul de lucru, suportul tranzacțiilor, răspunsul la erori, internaționalizarea şi localizarea, revocarea şi anularea acțiunilor, compensarea tranzacțiilor, timpii de aşteptare şi redresarea după căderi ale sistemului. De asemenea se enunță criterii privind realizarea interfețelor grafice cu utilizatorul.

Biblioteca grafică OpenGL a fost creată cu scopul de a fi utilizată pentru afişarea şi manipularea obiectelor 3D [Wri2004]. OpenGL este o interfață procedurală mai degrabă decât descriptivă, o interfață software cu hardware grafic. O caracteristică importantă a OpenGL este cea de independență față de platforma hardware pe care rulează. Paragraful 4.4 este dedicat descrierii modului de implementare a vizualizării OpenGL cu detalierea următoarelor problematici: utilizarea modului de selecție şi feedback, desenarea în mod *wireframe* şi cu linii ascunse precum şi descompunerea şi triunghiularizarea poligoanelor complexe utilizate si dezvoltate in cadrul aplicației propuse in teză.

4.2. Programarea orientata spre obiecte.

Programarea orientată spre obiecte (OO) ar fi dificil de definit exhaustiv datorită complexității sale [Kak2003] [Lip2005] [Str1997] [Sht2000] [Ton2005]. Pentru a releva elementele esențiale ale unei definiții cât mai cuprinzătoare este realizată o analiză paralelă între proiectarea OO şi modul de organizare a realității. Departe de a ştii totul despre organizarea şi funcționalitățile sistemului social (societății umane), există studii în special referitoare la dirijarea şi menținerea funcționalităților într-un regim armonios [Kak2003] [Kul2003]. Pentru aceasta se impune existența unei persoane suficient de inteligentă, capabilă să prevadă şi să normalizeze variabilele care alterează unele funcții, în condițiile unei distribuții şi variații în limite extinse, având ca țintă permanentă obiectivul propus. S-a constatat până acum, că numai o organizare ierarhică poate face față unei asemenea provocări, ierarhizare care condiționează şi facilitează propagarea mesajelor de jos în sus şi de sus în jos.

Dezvoltarea programelor orientate spre obiecte nu este principial diferită de cazul expus. Ideea este că un software complex, care uneori poate consta din milioane de linii de cod, este similar unui sistem social în care fiecare individ este suficient de inteligent pentru a interpreta un mesaj primit de la un alt individ, generând o atitudine sau acțiune corespunzătoare. Mai mult decât atât, exact ca şi în cazul eficientizării funcționării societății umane prin descentralizare şi în cazul proiectării OO o organizare descentralizată a software-ului facilitează extinderea şi mentenabilitatea aplicației. Ca exemplu pot fi evocate în paralel două situații. Una în care în proiectarea aplicației s-a ținut cont de problema de rezolvat în ansamblu şi alta în care proiectarea s-a realizat printr-o organizare descentralizată (orientată spre obiecte). Este uşor de remarcat că în cazul apariției unor probleme cum ar fi necesitatea unor modificări, situația poate fi rezolvată cu mult mai mare uşurință când se impune acționarea localizată în comparație cu primul caz.

În sistemul social, tratarea problemelor care apar este mult mai clară şi mai eficientă când se realizează în raport cu diverse grupuri omogene adică, care sunt constituite din indivizi care au în comun un număr mare de caracteristici. Această analogie conduce direct la proiectarea orientată pe obiecte. Toate obiectele care posedă aceleaşi atribute şi manifestă acelaşi comportament sunt grupate într-o *clasă*. De fapt, clasa este mai întâi definită şi apoi se

creează obiectele individuale prin procesul de instanțiere. Toate obiectele care posedă proprietăți şi comportament impuse de clasa din care fac parte, însă în acelaşi timp manifestă în plus anumite proprietăți şi comportamente mai specializate, reprezintă o *subclasă* a clasei definite anterior.

În ceea ce priveşte oportunitatea şi eficiența programării orientate pe obiect, in timp a devenit stilul preferat de programare pentru interfețe grafice cu utilizatorul (GUI) [Han2005] [Gal2002] [Kak2003] într-o asemenea măsură încât, chiar şi atunci când se utilizează limbaje ce nu suportă orientarea spre obiecte (cum ar fi C standard) programatorii creează structuri software care simulează OO pentru programarea GUI. În [Kak2003] este evocat probabil cel mai renumit exemplu în acest sens şi anume *GNOME/GTK+ toolkit* pentru proiectarea GUI: unde, deşi programarea s-a făcut în C, stilul de programare şi structurare este foarte apropiat de cel orientat spre obiecte.

Strategii de proiectare orientată pe obiect

Pentru a realiza paradigma programării OO, în scopul rezolvării problemelor ce implică sistemele complexe tratate în cadrul tezei, se impune adoptarea unei strategii care implică trei direcții de dezvoltare.

- *cunoaşterea sintaxei specifice* limbajului de programare utilizat pentru a putea identifica şi utiliza uneltele puse la dispoziție de limbaj în acord cu necesitățile temei de proiectare. Astfel se creează capabilitatea de a putea selecta din documentația disponibilă cea mai adecvată şi bine sistematizată variantă în concordanță cu obiectivele propuse.
- necesitatea cunoaşterii *conceptelor specifice* de baza ale unui limbaj OO: *încapsulare*, *moştenire* şi *polimorfism*. Este de notat că, în funcție de limbajul utilizat, semnificațiile acestor atribute sunt uşor nuanțate.
- necesitatea cunoaşterii *unor aspecte specifice implicite proiectării OO*, întrucât nu este posibil să se enunțe principii de proiectare general valabile. Pentru identificarea soluțiilor adecvate unor probleme complexe experiența si documentarea la zi in domeniu este decisiva in alegerea celui mai bun cod OO scris. Se considera următoarea abordare optimă:

- cunoaşterea unui aşa numit *„meta" limbaj*, cum ar fi limbajul de modelare unificat UML, care permite vizualizarea proiectului la un nivel conceptual. [Boo2005] [Coo2000] [Fow2003] [Gam1995] [Fav2003].
- cunoaşterea de *design patterns* ce prezintă *soluții template (generice)* pentru un număr mare de sub-probleme ce pot apărea în cursul evoluției proiectării OO [YAC2003] [Öve2004] [Ale2001].

In cadrul capitolului de față sunt evidențiate elementele care au fost utilizate în cadrul sistemului software CAD propus şi dezvoltat.

4.2.1. Etapele de dezvoltare ale produsului

Pentru înțelegerea sistemului care urmează a fi dezvoltat este necesar să se realizeze o evaluarea critică a elementelor necesare proiectării OO:

- analiza domeniului şi a cerințelor sistemului şi respectiv proiectarea sistemului,
- analiza cerințelor software, proiectarea software şi respectiv implementarea aplicației software cu implicarea unui limbaj adecvat.

4.2.1.1 Analiza domeniului sistemului

Clasele şi obiectele cu care se operează în cadrul analizei domeniului simbolizează obiecte ale lumii reale. In consecință punctul de pornire îl constituie identificarea, definirea şi clasificarea acestora.

Pe baza cerințelor funcționale pot fi enumerate următoarele tipuri de clase [Sht2000] [NIE1995] [Far2001]:

- clase interfață dispozitiv,
- clase sistem extern,
- clase interfață utilizator,
- clase computaționale,
- clase rol (de modelare a unui rol cheie jucat de un subsistem în cadrul sistemului);
- clase de abstractizare a datelor.

Criteriile de evaluare a măsurii în care obiectele de nivel înalt (create în faza de analiză a domeniului) sunt adecvate ca entități abstracte trebuie să țină seama de următoarele aspecte [Sht2000] [Lip2005] [NIE1995] [FAR2001]:

- Dacă acestea sunt *entități ale lumii reale* atunci fiecare trebuie să aibă un corespondent în cadrul cerințelor domeniului problemei, iar denumirile vor fi asociate noțiunilor definite în cadrul cerințelor domeniului.
- Setul de obiecte identificate trebuie să acopere complet setul de cerințe ale domeniului problemei, conform criteriului de *completitudine*.
- Criteriul *dimensiune* impune tratarea în faza de proiectare a obiectelor de dimensiuni prea mici, detaliate şi tratate ca subsisteme reutilizabile sau descompunerea în abstractizări mai mici a obiectelor cu un nivel de complexitate prea mare.
- Fiecare obiect trebuie identificat pe baza uneia din următoarele *categorii de dezvoltare*: produs reutilizabil intern, produs comercial sau produs ce necesită dezvoltare proprie.
- Orice obiect va fi încadrat în *clasificarea obiectelor*, clasificare bazată pe cerințele funcționale.
- Un *atribut* a cărui descriere este prea generală va fi convertit într-un obiect individual care va reprezenta de obicei o cerință derivată.
- Fiecare din *operațiile* asociate unei clase în faza curentă de dezvoltare trebuie să conțină o descriere unică ce identifică o singură acțiune asociată unui obiect.

Produsele de lucru generate în etapa de analiză a domeniului sunt:

- *Specificația funcțională a sistemului*: acest document conține cerințele sistemului în contextul domeniului problemei.
- *Clasele şi obiectele*: sunt generate abstractizări conceptuale de nivel înalt ale principalelor elemente din cadrul sistemului.
- *Fişe pentru clase şi obiecte*: aceste fişe sunt utilizate ca instrument în evaluarea eficacității claselor şi obiectelor
- *Software existent*: pentru fiecare set de clase şi de obiecte se va întocmi o listă a entităților software existente ce corespund funcționalității dorite.
- *Experiența internă*: identificarea experților în diversele arii ale domeniului problemei este importantă pentru stabilirea posibilității de implicare a acestora în dezvoltarea proiectului.

- *Planul de reducere a riscului*: sunt identificate zonele de risc ridicat (comunicațiile interproces, toleranța la erori, proiectarea bazelor de date etc.) pentru a fi avute în vedere în activitatea de generare de prototipuri.
- *Planul preliminar de dezvoltare incrementală*: în procesul de dezvoltare a sistemelor mari funcționalitatea este dezvoltată incremental. Fiecare pas de dezvoltare include un set de funcționalități suplimentare. Acest plan constituie baza pentru analiza *thread*-urilor și pentru determinarea scenariilor [JAC1992] în etapa de analiză a cerințelor sistem.

Produsele de lucru rezultate în etapa de analiză a domeniului reprezintă date inițiale și pentru evaluarea efortului de dezvoltare software care va fi diferențiat în funcție de necesitatea asumării unor sarcini legate de *generarea de prototipuri* sau sarcini legate de *dezvoltarea sistemului*.

4.2.1.2 Analiza cerințelor sistemului

Este important ca structurarea sistemului să se realizeze în așa fel încât acesta să fie independent de mediul de implementare. În acest mod se poate determina un comportament optim al sistemului, o structură robustă capabilă să facă față modificării cerințelor apărute pe parcurs, iar limitările sale vor fi mai reduse. Structurile logice și stabile bazate pe obiecte independente și având interfețe bine definite determina robustețea sistemului. Realizarea acestui deziderat poate fi bazată pe diverse concepte, unul dintre acestea, utilizat în dezvoltarea aplicațiilor din prezenta lucrare, este conceptul de *scenariu* [HAR1987] [DEU1988].

Scenariul poate fi definit ca o entitate și reprezintă un descriptor al comportamentului sistemului în termeni ai aplicației. Acesta reprezintă atât o specificație cât și o implementare, conține evenimente de date, evenimente de control și condiții, conectează un stimul extern cu un răspuns intern pentru a descrie o cale comportamentală perceptibilă de către utilizator și poate fi implementat atât în hardware, cât și în software.

Utilizarea scenariilor facilitează crearea unei structuri care permite o *dezvoltare incrementală a sistemelor mari*, reprezentând în același timp o bază pentru activitatea de generare a prototipurilor.

Etapa inițială a analizei constă în *identificarea scenariilor fiind* urmată de *identificarea obiectelor* care participă la fiecare scenariu.

Pentru determinarea scenariilor se stabilesc mai întâi *interfețele* sistemului cu utilizatorul sau cu alte sisteme externe, se trasează activitățile declanşate în cadrul sistemului de către fiecare stimul şi se înregistrează secvența de evenimente rezultate.

In cadrul analizei cerințelor sistemului pentru proiectarea OO, obiectele rezultate din etapa de analiză a domeniului vor fi rafinate şi se vor constitui în *baza modelului de analiză.*

Se consideră un spațiu informațional tridimensional, reprezentativ pentru cele trei componente, reprezentate pe fiecare axă:

- prezentarea informațiilor asociate interfețelor externe ale sistemului;
- prezentarea informației care descrie stările interne ale sistemului;
- răspunsul sistemului la schimbările de stare.

Modelul de analiză se bazează pe identificarea obiectelor corespunzătoare spațiului informațional tridimensional descris. Scopul modelului de analiză este de a crea baza pentru etapa de proiectare a sistemului. Modelul de analiză se bazează pe modelul scenariilor, fiecare scenariu conținând un număr de obiecte de analiză.

Obiectele pot fi grupate în trei categorii [JAC1992]:

- *obiectele entitate*: reprezintă cerințele sistemului reflectate în obiecte ale lumii reale şi sunt asociate axei informației.
- *obiectele de interfață*: modelează informația reprezentând interfețele cu sistemele externe şi sunt aliniate axei de prezentare.
- *obiectele de control*: sunt responsabile cu tratarea schimbărilor stării sistemului şi sunt aliniate axei comportamentului.

Funcționalitățile sistemului care nu pot fi încadrate în cele două tipuri de scenarii prezentate mai sus sunt implementate pe baza *obiectelor de control.*

- *Obiectele entitate* sunt utilizate pentru modelarea informației ce va persista şi după ce scenariul şi-a parcurs întreaga secvență de evenimente.

- *Obiectele interfață* sunt asociate evenimentelor unui scenariu ce implică interacțiunea între sistem și mediul său exterior [Kul2003] [Öve2004] [JAC1992]. Etapele scenariului se desfășoară în felul următor: inițial se consideră un *actor* și se determină *funcția de interfață* între stimulul generat de acesta și o funcție sistem. Dacă sunt disponibile *obiecte de analiză a domeniului*, atunci aceste obiecte din categoriile interfață dispozitiv, sistem extern, interfață utilizator și rol vor fi transpuse direct în obiecte de interfață, care la rândul lor pot fi identificate plecând de la o *descriere textuală a scenariului* provenind direct de la modelul cerințelor.
- *Obiectele de control* rezultă din alocarea funcționalităților rămase neacoperite după identificarea obiectelor interfață și a obiectelor entitate. Informația asociată obiectelor de control este de scurtă durată și persistă de regulă într-un interval de timp relativ scurt în cadrul unui scenariu.

Anumite obiecte pot participa la mai multe scenarii, acest fapt fiind determinant pentru ordinea în care sunt dezvoltate obiectele.

Tipurile de scenarii pentru generarea modelului de analiză pot fi [JAC1992]:

- scenarii dependente în mod direct de mediul sistemului, care implica obiectele interfață;
- scenarii responsabile cu stocarea și manipularea datelor în mod independent de mediul sistemului care sunt implementate prin intermediul obiectelor entitate. Unele dintre aceste obiecte pot fi derivate din clasele computaționale și de abstractizare a datelor de la nivelul analizei domeniului.

Una dintre problemele importante în ceea ce privește dezvoltarea unui sistem utilizând tehnologia OO este legată de modul în care se face delimitarea între etapa de analiză și etapa de proiectare. Lipsa unei demarcații clare între ele poate duce la situații în care modificarea cerințelor are ca efect reproiectarea unei porțiuni semnificative a sistemului.

Etapa de analiză nu abordează decât în măsură redusă atributele și operațiile asociate obiectelor, cu excepția acelor obiecte ce pot fi asociate mai multor scenarii. Aceste elemente vor fi abordate în etapa de proiectare, îmbunătățind astfel delimitarea între faza de analiză și cea de proiectare.

Abordarea orientată spre obiecte a etapei de analiză a cerințelor sistemului implică un *factor de risc* pentru succesul procesului de dezvoltare software, risc ce se amplifică cu dimensiunea proiectului. Factorul de risc trebuie evaluat înaintea începerii dezvoltării unui proiect. Soluția oferită în cazul sistemelor mari este de grupare a obiectelor în subsisteme. Un posibil factor de risc îl poate reprezenta şi *insuficienta delimitare între etapa de analiză şi etapa de proiectare*. Există pericolul de a se pierde informația din categoria "ce", specifică etapei de analiză în favoarea informației din categoria "cum" specifică etapei de proiectare, generând pentru aplicație o structură dependentă de implementare.

Abordarea orientată spre scenarii considerată se caracterizează printr-o bună separare a etapei de analiză de cea de proiectare datorită criteriilor de încheiere a etapei de analiză.

4.2.1.3 Proiectarea sistemului

Cele două activități de bază, *partiționarea* şi *configurarea*, ce constituie etapa de proiectare a sistemului, realizează în fapt alocarea cerințelor sistemului componentelor hardware şi software. *Partiționarea* divizează setul de cerințe în subseturi ce vor forma baza arhitecturii sistemului. Prin *configurarea* sistemului se alocă cerințele din cadrul fiecărui modul componentelor hardware şi software. Cele două etape au uzual un caracter iterativ.

În funcție de perspectiva dezvoltatorului asupra sistemului proiectat se pot identifica metode adecvate de partiționare [NIE1995]. Acestea pot fi de natură: *pur analitică* (bazate pe diagrame de flux a datelor, diagrame de flux a controlului, specificații de proces şi specificații de control [HAT1987]), *pur intuitivă* (pe baza experienței anterioare) sau pot fi bazate pe *gruparea obiectelor* (analiza orientată spre obiecte efectuată în etapele de analiză a domeniului mai exact in cadrul analiză a cerințelor sistemului).

4.2.1.4 Analiza cerințelor software

Etapa de analiză a cerințelor software poate fi abordată imediat după etapa de analiză a cerințelor sistemului.

Analiza cerințelor software se axează pe cerințele fiecărei componente software şi nu pe sistemul global. Clasele si obiectele determinate reprezintă baza pentru transpunerea acestor elemente în entități de proiectare.

Principalul instrument care face posibilă tranziția de la analiză la proiectarea software este ***modelul*** *ca rezultat al etapei de analiză a cerințelor software. Modelele în*

funcție de metodele care le generează pot fi ***comportamentale, informaționale*** *sau* ***modele orientate spre obiecte***. Ultima variantă poate fi utilizată pentru a produce componente reutilizabile prin crearea de clase şi obiecte ale lumii reale.

În cadrul procesului de proiectare fiecare model poate fi utilizat după caz, existand si posibilitatea asocierii lor când acestea se completează reciproc.

Cerințele software orientate spre obiecte nu diferă de cele utilizate în etapa de analiză a cerințelor sistemului. Obiectul asupra căruia acționează cerințele este individual în acest caz şi nu global.

Avantajul primar al analizei software orientate spre obiecte este focalizarea încă din stadiul inițial asupra entităților funcționale, care ulterior pot fi implementate în calitate de entități software reutilizabile în limbajul de programare ales.

Codul reutilizabil trebuie să posede, pe lângă capacitatea de a fi folosit în mai multe proiecte sau pe platforme multiple şi capacitatea de reutilizare în cadrul aceleiaşi aplicații [Ale2001][Nie1995]. Gradul de reutilizabilitate al unei componente nu asigură integral corectitudinea sau claritatea unui obiect ci contribuie parțial la atingerea acestor deziderate.

Evoluția software-ului, prin adăugarea de caracteristici noi sau prin întreținerea de rutină poate conduce la devieri de la obiectivul inițial [Sht2000]. Aplicațiile proiectate corespunzător trebuie să faciliteze modificările ulterioare fără alterarea sistemului existent.

4.2.1.5 Proiectarea software

Prin proiectarea software se înțelege generarea structurii software a aplicației prin crearea de componente arhitecturale software şi a interfețelor acestora.

Un aspect semnificativ al etapei de proiectare software este toleranța la erori, care poate fi indeplinită apelând la ajutorul unei strategii de tratare a excepțiilor.

- Prima activitate din cadrul proiectării software este cea de *transpunere a entităților de analiză în entități de proiectare*. Această activitate se desfăşoară sub incidența unui set de reguli de transpunere. Odată transpuse entitățile de analiză în entități de proiectare se ajunge la un set inițial care va fi completat cu clase şi obiecte generate în cadrul etapei de proiectare software.

  Un exemplu de corespondență între entități corespunzător unui program de tip *single-thread*, ignorând aspectele legate de concurență este prezentat în [NIE1995]:

clase şi obiecte → clase

atribute → date membre

operații → funcții membre (metode)

relații → moştenire, agregare şi utilizare

diagrame de tranziție a stărilor → ierarhii de clase

scenarii şi obiecte entitate → obiecte entitate

modelul de flux al obiectelor → obiecte în cadrul subsistemelor

diagrama de trasare a evenimentelor → mesaje transmise între obiecte

Metoda de dezvoltare propusă nu ține cont de momentul abordării moştenirii în etapa de analiză, ca urmare relațiile clasă-obiect vor fi transpuse în relații de moştenire şi agregare în etapa de proiectare.

- A doua activitate din cadrul etapei de proiectare este cea de *structurare a proceselor*, cu alte cuvinte de abstractizare a unei entități care va fi implementată sub forma unui modul software susceptibil de a fi executat concurent cu alte procese. Procesul creat poate să conțină procese concurente care comunică prin date partajate sau prin mesaje şi procese distribuite care comunică prin mesaje.
- Al treilea pas al proiectării software este *proiectarea orientată spre obiecte* şi implică crearea componentelor arhitecturale şi a interfețelor acestora astfel încât să poată fi implementate cu ajutorul unui limbaj de programare orientat spre obiecte. Este de menționat faptul că proiectarea interfețelor are ca efect generarea de noi clase şi obiecte. Verificarea măsurii în care atributele şi operațiile satisfac liniile generale impuse de proiectare se face prin luarea în considerare a două noțiuni: *vizibilitatea* şi *nivelul de încapsulare şi de ascundere a datelor* [Lip2005][Sol2005]. Rezultă astfel diagrama de structură a proceselor însoțită de o descriere a lor, diagrama care fiind o oglindă a arhitecturii software pune în evidență relațiile dintre clase. Pentru o mai mare claritate este necesară descrierea mecanismului de comunicare interproces alături de justificarea soluției adoptate. De asemenea documentația mecanismului de comunicare interproces trebuie să includă toate restricțiile impuse, cum ar fi modul de conexiune, lungimea mesajelor, utilizarea datelor partajate, alocarea de spațiu tampon pentru coada de mesaje, identificarea mesajelor şi altele.

Tratarea excepțiilor precum şi tratarea erorilor sunt incluse în activitatea de proiectare OO.

Tratarea erorilor poate fi abordată prin utilizarea unor soluții diverse prin: ignorare, returnarea unei stări de eroare, setarea unei variabile de eroare partajate, utilizarea rutinelor de bibliotecă sau crearea de rutine de tratare a erorilor, definite de utilizator.

Tratarea excepțiilor excede tratarea erorilor şi constituie un element cheie pentru implementarea toleranței la defecțiuni a produsului software [Blu2003].

Categoriile de excepții care se recomandă a fi tratate în cadrul mecanismului general de toleranță la defecțiuni sunt [Blu2003] [Sht2000] [NIE1995]:

- *Condiții excepționale anticipate*: reprezintă excepții care pot rezulta din condiții (rare) care nu ar trebui să întrerupă programul.
- *Date de intrare eronate sau inconsistente*: pot fi detectate în cadrul unui modul software (server) care este apelat pentru a efectua un anumit serviciu.
- *Erori hardware*: funcționarea dispozitivelor hardware este legată de obicei de o anumită perioadă de viață.
- *Erori neanticipate (erori de programare)*: erorile de programare sunt frecvente în cadrul produselor software, fiind necesară detecția şi raportarea acestora.

În general, mecanismul de tratare a excepțiilor este proiectat ca parte a sistemului global. Aceasta este considerată a fi o *strategie de programare defensivă* care suportă toleranța la defecțiuni.

Orice plan de dezvoltare software trebuie să prevadă timp pentru *realizarea de prototipuri*, activitate care prezintă avantaje şi poate afecta în mod direct succesul dezvoltării unei aplicații comerciale [Sny2003] [Amb1998].

Regulile de bază în ceea ce priveşte activitatea de realizare a prototipurilor sunt următoarele:

- analiza inițială a părților mai dificile ale problemei;
- utilizarea practicilor adecvate de programare;
- documentarea prototipurilor în cele mai mici detalii pentru a îngloba ideile de proiectare;
- experimentarea prin utilizarea de elemente diferite ale limbajului de programare pentru a analiza modul în care acestea afectează proiectarea şi implementarea;
- utilizarea experienței dobândite în dezvoltarea unui prototip pentru accelerarea

dezvoltării următorului prototip;

- scrierea de unități de testare a prototipurilor;
- asigurarea conformității compilatoarelor pe diferitele platforme de dezvoltare;
- evitarea distribuirii prototipurilor în calitate de produse software finalizate.

Experiența a dovedit faptul că utilizarea de prototipuri în cursul procesului de dezvoltare favorizează terminarea produsului la timp [Rom2003], beneficiile acestei practici fiind următoarele:

- *Descoperirea la timp a problemelor*. Prototipurile ajută la dezvoltarea proiectului ce va genera produsul final, oferind posibilitatea de detectare a erorilor încă din faza inițială. Ele permit modificarea proiectului pentru evitarea greşelilor şi evidențiază mai clar cerințele necesare obținerii produsului final. Dacă descoperirea unei greşeli se face în timpul fazei de dezvoltare, acest lucru poate influența negativ atât conținutul produsului cât şi termenul de finalizare al lui;
- *Evaluarea performanțelor şi dimensiunea codului*. Scopul realizării prototipului nu este acela de a dezvolta produsul ci de a dezvolta idei şi un cadru pentru proiectarea aplicației. Prototipurile trebuie să implementeze doar o parte din întreaga soluție.
- *Asigurarea compatibilității între platformele de rulare*. Prototipurile sunt utile şi atunci când produsul trebuie să ruleze pe mai multe platforme sau pe un sistem înglobat. Prototipurile pot contribui la luarea deciziei în privința compilatorului şi a versiunii sale, fiind testată capacitatea compilatorului de a funcționa corespunzător.
- *Testarea de noi facilități ale limbajului*. Prototipurile sunt utile pentru testarea de noi concepte sau caracteristici ale limbajului. Acest lucru este util mai ales pentru studiul template-urilor. Forma finală a unui obiect template poate să nu fie vizibilă de la început, conturându-se de obicei în faza de proiectare software, iar prototipurile contribuie la găsirea soluției optime.

Utilitatea obiectelor şi funcțiilor *template*

În codul sursă al unei aplicații există părți care diferă doar prin tipurile de date utilizate [Van2002] [Daw2004] [Sht2000]. Acest lucru se observă cel mai bine în cazul

implementării algoritmilor cum ar fi sortările sau în cazul implementării de rutine de manipulare a datelor cum sunt listele înlănțuite.

În situația in care în care limbajul de programare utilizat nu oferă o facilitate specială pentru această situație, programatorul nu are la dispoziție alternative favorabile, dintre soluțiile posibile putându-se enumera următoarele:

- scrierea aceluiași comportament de mai multe ori, pentru fiecare tip de date care necesită comportamentul respectiv;
- scrierea de cod generic ce manipulează un tip de bază comună cum ar fi `Object` sau `void*`;
- utilizarea de preprocesoare de cod speciale

soluții care însă prezintă fiecare în parte dezavantaje specifice [Van2002].

O alternativă viabilă o oferă utilizarea *template*-urilor [Str1997] [YAC2003] [Öve2004] [Ale2001], care reprezintă funcții sau clase, scrise pentru unul sau mai multe tipuri nespecificate. La utilizarea unui *template*, tipurile sunt specificate ca argumente, explicit sau implicit. Fiind o caracteristică de limbaj, template-urile dispun de mecanismele de *type-checking*.

Există câteva concepte importante ce trebuie înțelese despre relația dintre clasele template şi tipurile de date [Tri2005]:

1. compilatorul utilizează clasele template pentru a crea tipuri prin substituirea parametrilor template. Acest proces se numeşte instanțiere;
2. tipul creat dintr-o clasă template se numeşte specializare;
3. instanțierea template-urilor se produce atunci când este necesar, adică compilatorul va crea specializarea atunci când va întâlni în cod o utilizare (locul respectiv se numeşte punctul de instanțiere);
4. pentru crearea unei specializări, compilatorul trebuie să cunoască nu doar declarația template-ului ci şi definiția;
5. sunt instanțiate doar definițiile funcțiilor care sunt utilizate.

Proiecatarea software se încheie printr-un proces de evaluare a proiectării.

Tehnicile de proiectare şi implementare POO sunt evaluate în funcție de următoarele caracteristici [Rom2003]:

- **comunicatia:** în cazul unei aplicații dezvoltate de un grup de programatori. Comunicația trebuie să persiste de-a lungul întregului proces de dezvoltare asigurând că: interfețele să fie concepute astfel încât să asigure integrarea perfectă a componentelor,; documentarea precisă şi exhaustivă a codului; aducerea la cunoştință a funcțiilor noi către testori şi către cei care realizează documentația aplicației;
- **extensibilitatea**: se impune ca adăugarea de noi funcții să se realizeze prin soluții simple şi rapide. Posibilitatea de extindere a codului pentru a implementa cerințe noi, după ce produsul a fost livrat, depinde în mare măsură de modul în care codul a fost proiectat, fiind recomandabil să se evite simpla implementare a cerințelor, fără a ține cont de posibile modificări ulterioare, compatibilitatea înapoi reprezentând o constrângere majoră;
- **mentenabilitatea**: Apariția erorilor este practic inevitabilă chiar şi după ce produsul a fost livrat către clienți, independent de numărul de module de testare realizate, de numărul de teste automate rulate sau de numărul de teste de funcționalitate efectuate. Proiectarea solidă a codului precum si claritatea stilului de programare astfel încât software-ul să fie întreținut eficient, fără riscul de al complica, reprezintă soluții favorabile mentenabilității;
- **inteligibilitatea**: Deseori, software-ul comercial conține interfețe software vizibile. Acest lucru nu presupune doar utilizarea unei nomenclaturi sugestive a funcțiilor ci, mai important, proiectarea codului precum şi utilizarea elementelor limbajului de programare, cum ar fi template-urile, să fie clară şi adecvată;
- **stabilitatea**: Software-ul comercial trebuie să fie stabil. El trebuie să poată rula perioade lungi de timp fără a genera erori fatale, scurgeri de memorie sau alte anomalii funcționale.

Revizuirea proiectului poate avea caracter *formal* sau *informal*. Evaluarea formală presupune revizuirea globală a rezultatului etapei de proiectare şi implică asigurarea beneficiarului de fiabilitatea şi performanța solicitată inițial. Revizuirile informale au loc intern, periodic asigurând echipa de lucru că activitatea de proiectare este îndreptată în direcția corectă sau informând despre eventuale devieri, oferind în acest fel posibilitatea reorientării activității de proiectare [Kul2003].

Evaluarea rezultatelor proiectării software din punct de vedere calitativ urmăreşte gradul de îndeplinire a unor *criterii de calitate* vizând structura proceselor, structura de clase, interfețele claselor, utilizarea moştenirii şi utilizarea excepțiilor. Evaluarea cantitativă este bazată pe utilizarea unor metrici software.

Evaluarea rezultatelor proiectării vizând **structura proceselor** va fi concentrată pe următoarele caracteristici [Sht2000] [Blu2003] [Ale2001] [Rom2003]: *numărul proceselor, interacțiunile între procese, elemente determinante ale proceselor ciclice, polling, date partajate* şi *excluziune mutuală.*

Structura de clase este evaluată în funcție de modul în care se încadrează o anumită clasă într-o ierarhie, presupunând şi verificarea setului sau de operații şi atribute: *categoria clasei, încapsularea şi nivelul de abstractizare, moştenirea, agregarea.* Operațiile asociate unei clase trebuie să formeze un set complet pentru abstractizarea încapsulată. Atributele selectate trebuie să reprezinte elemente de date asociate instanțelor clasei şi trebuie să fie în relație directă cu abstractizarea încapsulată.

Interfața unei clase este evaluată prin prisma operațiilor disponibile şi a nivelului de ascundere a informației. Un alt aspect al evaluării se referă la posibilitatea de a utiliza clasa pe post de clasă de bază pentru o clasă derivată. Criteriile de evaluare fiind: *ascunderea informației, operațiile disponibile (funcțiile manager, de implementare, auxiliare, de acces) şi trăsăturile claselor de bază.*

Evaluarea proiectării prin prisma **utilizării moştenirii** se realizează ținând cont de următoarele aspecte: *numărul de nivele în structura de moştenire, derivarea subclaselor din mai mult de o clasă de bază şi justificarea judicioasă a utilizării moştenirii multiple.*

Utilizarea excepțiilor va fi specificată uzual în faza de proiectare de nivel înalt şi evaluată în faza de proiectare detaliată. Sunt acceptate următoarele utilizări: *condiții excepționale anticipate, protecția software-ului server, raportarea eşecurilor hardware şi raportarea erorilor neanticipate.* Pentru asigurarea eliminării totale din program a excepțiilor care nu sunt neapărat necesare, o excepție trebuie tratată cât mai aproape de locul unde a fost detectată.

4.2.1.6 Implementarea

Arhitectura software derivată dintr-un set de cerințe software este creată utilizând facilitățile limbajului de programare ales. Implementarea reprezintă ultimul pas în procesul de dezvoltare în care activitatea de programare este susținută de o strategie de testare de

detectare a erorilor. Implementarea se realizează incremental pentru o anumită versiune care corespunde unei porțiuni bine definite a cerințelor conducând în final la implementarea pentru întreaga aplicație. Unul dintre instrumentele extrem utile în această fază este *debugger-ul simbolic* care poate fi adaptat unui anumit şablon de testare [Blu2003] [Ber2003], punctele de întrerupere facilitând examinarea porțiunii staționare a programului aflat în rulare. Entitățile de proiectare create în etapa de proiectare OO vor fi implementate utilizând facilitățile limbajului C++. Deciziile luate în etapa de proiectare vor fi implementate direct utilizând structuri adecvate, sau uzând de caracteristici ale limbajului de programare. Tranziția de la faza de entități de proiectare la faza de entități de programare cuprinde următoarele transformări [Rom2003] [Far2001]:

- clase → clase C++
- operații → funcții membre
- atribute → date membre
- ascunderea informației → segmente *private* şi *protected* în interfața clasei
- relaxarea ascunderii informației → utilizarea noțiunii *friend*
- ierarhia de moştenire → utilizarea noțiunilor de moştenire
- agregarea → declararea datelor membre ca şi obiecte ale altor clase
- modularitatea → separarea fişierelor header de fişierele sursă
- tratarea excepțiilor → noțiuni de excepții C++ sau *assert()*.

Datorită faptului că în timpul activității de programare se ajunge la necesitatea utilizării unor entități de programare diferite de cele puse la dispoziție de setul inițial, între etapele de proiectare şi implementare are loc un proces iterativ.

Limbajul ales pentru etapa de implementare este C++, prezentând suport nativ pentru moştenire şi suport run-time pentru polimorfism [Str1997] [SCH2004] [Lip2005] [Kak2003].

Clasele transpuse din faza de proiectare se implementează ca şi tipuri de date abstracte încapsulate sub formă de clase C++ cu un anumit nivel de ascundere al informației care a fost determinat tot în faza de proiectare şi va fi implementat sub formă de segmente *public*, *private* şi *protected*. Unele dintre tipurile de date abstracte cum ar fi stivele, cozile, listele se implementează utilizând meta-clasele (*template*) pentru a maximiza reutilizarea acestora în cadrul structurilor care diferă doar prin tip. *Ierarhiile de moştenire* sunt implementate utilizând trăsăturile obiectuale ale limbajului C++, cum ar fi structuri pentru crearea de clase

de bază şi clase derivate. *Relațiile de agregare* sunt implementate declarând clasele agregate ca obiecte membre în cadrul clasei care le conține.

Programarea modulară este utilizată pentru a facilita reutilizarea şi întreținerea codului. Trebuie concepută o structură de fişiere pentru a minimiza durata compilării în timpul dezvoltării, precum şi în cazul inevitabde e ilelor modificări ale cerințelor. Acest aspect este rezolvat în C++ de obicei prin utilizarea fişierelor *header* pentru definirea interfețelor [Rom2003] [Hoh2003]. Codul sursă C++ este plasat în fişiere diferite. Această separare a specificațiilor de implementare facilitează o strategie de compilare incrementală. Dacă interfețele nu se modifică softul client nu trebuie recompilat.

Detecția şi tratarea excepțiilor în faza de rulare reprezintă cheia realizării unui sistem tolerant la defecțiuni [Blu2003] [Daw2004]. Strategia de proiectare pentru toleranța la defecțiuni se stabileşte în etapa de proiectare software. Pentru implementarea mecanismului de tratare a excepțiilor pot fi utilizate următoarele metode [Str1997] [Sol2005]:

- utilizare funcției C *assert();*
- utilizarea mecanismului intern C++ de tratare a excepțiilor;
- combinarea celor două metode.

Mecanismul C++ de tratare a erorilor este disponibil în majoritatea mediilor de dezvoltare [Ber2003]. Efortul implicat de includerea mecanismului de tratare a excepțiilor se compensează prin evitarea propagării erorilor nedetectate care pot avea efecte fatale asupra aplicației.

Procesul de testare este legat de scenariul care se implementează. Procedurile de testare rezultă pe baza analizei thread-urilor, incluzând stimulii de intrare şi rezultatele aşteptate. Nu este fezabilă verificarea tuturor căilor posibile.

Procedurile de testare trebuie scrise pentru toate interfețele sistemului cu utilizatorul. Testarea claselor individuale (*testul de componente*) este deosebit de importantă înainte de a le utiliza împreună în cadrul unei aplicații (*testul de integrare*), ducând la reducerea timpului global de testare.

Mecanismul de tratare a excepțiilor stabilit în etapa de proiectare va fi verificat în cadrul procesului de testare. În cadrul fiecărei noi funcții poate fi plasat un cod sursă care să controleze generarea de excepții. Acesta va fi eliminat din versiunea finală a produsului.

Faza de depanare are loc în cadrul procesului de testare şi este necesară pentru a descoperi un număr cât mai mare de erori software. *Debugger*-ul este în mod uzual un instrument care trebuie să însoțească procesul de depanare facilitând examinarea unei porțiuni staționare a programului aflat în rulare.

Activitatea de depanare reprezintă o secvență de paşi care include testarea, inspectarea şi remedierea segmentului de cod care conține o eroare şi repornirea după recompilare şi link-editare.

4.2.1.6.1 Implementarea template-urilor

Funcțiile *template* oferă o funcționalitate ce poate fi apelată pentru tipuri diferite de date. O funcție *template* reprezintă, de fapt, o familie de funcții.

Parametrii funcției template se specifică utilizând următoarea sintaxă:

```
template < listă-de-parametri-separati-cu-virgulă >
```

Ca exemplu de funcție template se prezintă următoarea funcție ce compară dispunerea pe axa z a două solide prin compararea z minim şi z maxim:

```
template <class T> short compareVolSel(SortedObject<T> *vs1,
SortedObject<T> *vs2)
{
  if(vs1->minz == vs2->minz)
  {
    if(vs1->maxz == vs2->maxz)
      return 0;
    if(vs1->maxz > vs2->maxz)
      return 1;
    else
      return -1;
  }
  if(vs1->minz > vs2->minz)
    return 1;
  else
    return -1;
     return -1;
}
```

În acest caz lista de parametrii este **`typename T`**. Tipul T reprezintă un tip arbitrar ce este specificat atunci când se apelează funcția. Orice tip de date poate fi utilizat dacă oferă toate operațiile folosite de către funcția template. În cazul exemplului, tipul T trebuie să suporte operatorul < pentru că a şi b sunt comparate prin intermediul acestui operator.

Template-urile sunt compilate pentru fiecare tip de date utilizat, compilatorul generând câte un cod obiect pentru fiecare tip de date utilizat.

Instanțierea reprezintă procesul de înlocuire a parametrilor template-ului cu tipuri concrete de date, obținând ca rezultat o *instanță* a template-ului [Van2002][Str1997]. Instanțierea se produce automat la simpla utilizare a unei funcții template. Dacă se încearcă instanțierea unei funcții template prin utilizarea unui tip ce nu suportă toate operațiile implicate de către funcția template, se generează o eroare de compilare.

Template-urile sunt compilate în doi paşi:

- în lipsa unei instanțieri, compilatorul verifică doar sintaxa codului template-ului;
- în momentul instanțierii codului template-ului, este verificată validitatea fiecărui apel. În această etapă apelurile invalide sunt descoperite şi raportate de către compilator.

În momentul în care o funcție template este utilizată astfel încât să necesite o instanțiere a acesteia, compilatorul trebuie să cunoască deja definiția template-ului. Acest lucru diferă de stilul clasic de utilizare a funcțiilor obişnuite în care, la compilare, este suficient să se cunoască declarația funcției.

Moduri de implementare

Organizarea codului sursă a template-urilor se poate realiza în diferite moduri [Tri2005]:

- modul prin incluziune (*the inclusion model*)
- instanțierea explicită (*explicit instantiation*)
- modul prin separație (*the separation model*)

Modul prin incluziune

Organizarea codului sursă ce nu conține template-uri se face, în principiu, în felul următor:

- clasele precum şi alte tipuri de date sunt plasate în întregime în cadrul fişierelor header. Tipic, acestea sunt fişiere cu extensia **.h**;
- în cazul variabilelor şi funcțiilor globale, în fişierele header sunt specificate doar declarațiile iar definițiile sunt scrise în fişierele cu extensia **.cpp**.

Pornind de la această convenție, codul se poate scrie în felul următor:

```
// exemplu.h

#ifndef EXEMPLU_H
#define EXEMPLU_H

// declarata template-ului
template <typename T>
void print_typeof (T const&);

#endif // EXEMPLU_H

Implementarea funcției se face intr-un fişier .cpp:

// exemplu.cpp

#include <iostream>
#include <typeinfo>
#include "exemplu.h"

// implementarea/definitia template-ului
template <typename T>
void print_typeof (T const& x)
{
    std::cout << typeid(x).name() << std::endl;
}
```

Utilizarea funcției template se face astfel:

```
// exemplumain.cpp

#include "exemplu.h"

// utilizarea template-ului
int main()
{
    double ice = 3.0;
    print_typeof(ice);   // apelul functiei template pentru tipul
double
}
```

Compilatorul C++ acceptă un astfel de program fără a genera erori, în schimb *linker*-ul va genera o eroare prin care indică faptul că nu există o definiție pentru funcția

›rint_typeof(). Motivul acestei erori îl reprezintă faptul că definiţa funcţiei template ›rint_typeof() nu a fost instanţiată. Pentru ca o funcţie template să fie instanţiată, ompilatorul trebuie să ştie ce definiţie şi pentru ce argumente de template trebuie făcută nstanţierea. În exemplul anterior, cele două părţi de informaţie se află în două fişiere ompilate separat. Pe de-o parte, la apelulul funcţiei **print_typeof()** compilatorul ›resupune că funţia este definită în altă parte şi generează o referinţă către acea definiţie ›entru *linker*. Pe de altă parte, la compilarea fişierului în care funcţia template este definită ompilatorul nu cunoaşte tipurile concrete pentru care funcţia template trebuie să fie nstanţiată.

Există câteva soluţii recomandate [Tri2005] [Sol2005] [Van2002]:

- abordarea asemănătoare utilizării macrourilor şi a funcţiilor inline. Astfel, definiţia templateului se va include în headerul care declară respectivul template. În cazul exemplului prezentat se procedează la inserarea liniei

```
#include „exemplu1.cpp"
```

ı sfârşitul fişierului exemplu1.h.

- includerea lui exemplu.cpp in fiecare fişier **.cpp** care utilizează template-ul;
- renunţarea la fişierul exemplu.cpp şi rescrierea exemplu.h astfel încât să conţină toate declaraţiile şi definiţiile templateului:

```
//exemplu2.h

#ifndef EXEMPLU_H
#define EXEMPLU_H

#include <iostream>
#include <typeinfo>

// declaratia template-ului
template <typename T>
void print_typeof (T const&);

// implementarea/definitia template-ului
template <typename T>
void print_typeof (T const& x)
{
    std::cout << typeid(x).name() << std::endl;
}

#endif // EXEMPLU_H
```

Deşi modul prin incluziune prezintă dezavantajul unei cantități mari de cod compilat respectiv timpi îndelungați de compilare, este metoda cea mai utilizată.

În cadrul acestei metode este de remarcat faptul că funcțiile template non-inline diferă semnificativ de funcțiile inline şi macrouri. Ele nu sunt expandate în locul în care sunt apelate ci se creează o nouă copie a funcției doar atunci când sunt instanțiate. Din cauză că acest proces este automat, compilatorul poate crea două copii ale aceleaşi funcții în două fişiere diferite. Ca urmare, unele *linker*-e ar putea genera erori atunci când întâlnesc două definiții distincte ale aceeaşi funcții.

Instanțierea explicită

Modul prin incluziune asigură instanțierea tuturor template-urilor necesare după caz, întrucât compilatorul generează automat instanțierile respective în funcție de necesități [Van2002]. Standardul C++ oferă şi o directivă pentru instanțierea manuală a templateurilor: *directiva de instanțiere explicită.*

În continuare este prezentată o soluție de instanțiere manuală aplicată codului din exemplul anterior ce producea eroare la link. Pentru a corecta acea eroare se adaugă următorul fişier la program:

```
// exempluinst.cpp
#include "exemplu.cpp"
// instantierea explicita print_typeof() pentru tipul double
template void print_typeof<double>(double const&);
```

Directiva de instanțiere explicită constă în utilizarea cuvântului cheie **`template`** urmat de declarația în clar a entității care se doreşte a fi instanțiată.

Dezavantajul instanțierii manuale este acela că necesită urmărirea tuturor entităților ce trebuie instanțiate. Pentru proiecte mari acest lucru va face şi mai dificil procesul de dezvoltare.

Instanțierea manuală prezintă următoarele *avantaje* [Van2002] [Sht2000] [Sol2005]:

- instanțierea poate fi adaptată nevoilor programului;
- se evită utilizarea de fişiere header mari;
- codul sursă al definițiilor templateurilor poate fi ținut ascuns;
- pentru unele aplicații este util controlul locului din cadrul codului obiect în care se instanțiază templateul.

Aceste avantaje sunt greu de obținut în cazul instanțierii automate.

Modul prin separație

Standardul C++ oferă un mecanism alternativ de exportare a template-urilor, denumit *modul prin separație*.

Acesta se implementează cu ajutorul cuvântului cheie **export** prin care, într-un singur fișier, sunt marcate atât definiția precum și declarațiile template-ului [Van2002].

În cazul exemplului considerat, utilizarea modului prin separație conduce la următoarea declarație a funcției template:

```
// exemplu3.h

#ifndef EXEMPLU_H
#define EXEMPLU_H

// declaratia template-ului
export
template <typename T>
void print_typeof (T const&);

#endif // EXEMPLU_H
```

Template-urile exportate pot fi utilizate și fără ca definiția lor să fie vizibilă.

O implemetare posibilă a modului prin separație este cea în care se poate comuta între modul de incluziune și cel de separație cu ajutorul unui mecanism simplu, bazat pe directive de preprocesor.

Exemplul considerat devine astfel:

```
// basics/exemplu4.h

#ifndef EXEMPLU_H
#define EXEMPLU_H

// se utilizeaza export daca USE_EXPORT e definit
#if defined(USE_EXPORT)
#define EXPORT export
#else
#define EXPORT
#endif
// declarata template-ului
EXPORT
template <typename T>
void print_typeof (T const&);
// se include definitia daca USE_EXPORT nu e definit
#if !defined(USE_EXPORT)
#include "exemplu.cpp"
```

```
#endif

#endif // EXEMPLU_H
```

Prin definirea sau omiterea simbolului de preprocesor **USE_EXPORT** se poate selecta între cele două moduri. Modul prin separație este utilizat în cazul în care **USE_EXPORT** se defineşte înainte de a include exemplu.h:

```
// se utilizeaza modul prin separate:
#define USE_EXPORT
#include "exemplu.h"
```

Dacă **USE_EXPORT** nu este definit, este utilizat modul prin incluziune.

Particularitati ale modului de separatie:

- directiva export afectează ambele locuri, de instanțiere şi de definiție. Modificarea fişierului ce conține definiția implică recompilarea tuturor fişierelor ce instanțiază acel template. Această situație nu diferă mult de modul prin incluziune dar în cazul modului prin separație modificările nu sunt la fel de evidente, urmărind codul sursă. Din această cauză programele de management al dependințelor, cum sunt **make** sau **nmake**, nu mai funcționează corect;
- *template*-urile exportate pot conduce la consecințe semantice deosebite;
- experiența în utilizarea template-urilor exportate este limitată întrucât, după 4 ani de la publicarea standardului, doar o singură companie a implementat suportul pentru cuvântul cheie **export**.

Depanarea *template*-urilor

Problemele care apar când se încearcă depanarea template-urilor pot fi clasificate în funcție de două criterii:

- cele legate în principal de autorii template-urilor şi anume gradul de siguranță cu privire la funcționarea template-urilor la utilizarea oricăror argumente ce satisfac condițiile documentate;
- cele legate de programatori şi anume: capacitatea programatorului de a identifica argumentele greşite atunci când template-ul nu se comportă conform documentației.

Constrângerile ce pot decurge sunt de natură:

- sintactică, ce generează erori la compilare, uşor de identificat automat;
- semantică, mult mai dificil de detectat automat.

4.2.1.6.2 Prototipizarea

Pentru exemplificarea procesului de dezvoltare cu ajutorul prototipizării s-a ales realizarea unei aplicații de generare de pictograme pentru imagini.

Strategia de prototipizare este prezentată în figura 4-1:

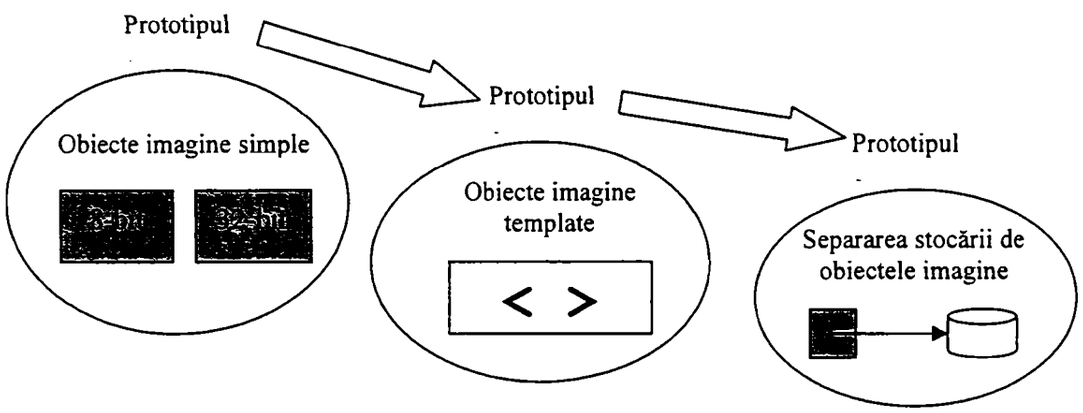


Figura 4-1. Strategia de prototipizare

S-a ales o strategie de prototipizare ce permite *analiza* a trei aspecte diferite ale problemei:

Prototipul 1

Este realizat pentru a analiza similaritățile dintre imaginile cu adâncimi de culoare diferite, considerând două tipuri de imagini monocrome: pe 8 biți şi pe 32 biți. Figura 4-2 prezintă strategia de realizare a clasei ce reprezintă o imagine în cazul prototipului 1.

O imagine pe 8 biți este reprezentată de un `unsigned char`. Similar, o imagine pe 32 de biți este reprezentată de un `unsigned int`. Fiecare prototip defineşte o clasă simplă pentru crearea unei imagini şi suportă o singură operație şi anume cea de generare a unei pictograme.

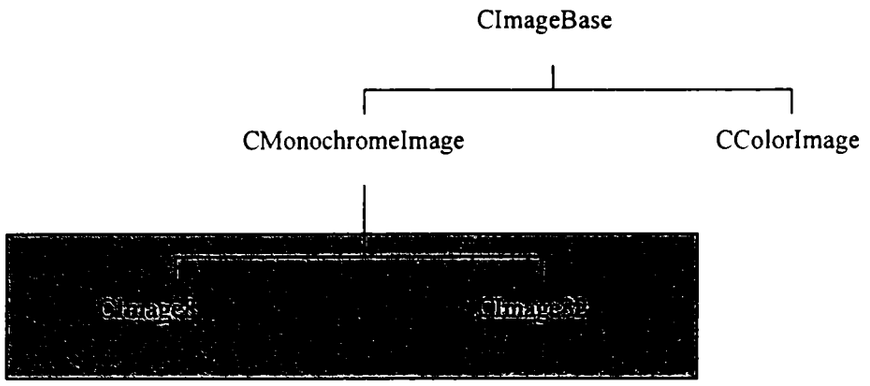


Figura 4-2. Strategia de realizare a clasei imagine în cazul prototipului 1

Declarația clasei **CImage8** implică:

```
typedef unsigned char Pixel8;

class CImage8
{
public:
  CImage8 ();
  CImage8 (int width, int height);
  // Creaza o imagine nula sau de dimensiunea specificata

  virtual ~CImage8 ();

  int width  () const { return m_width;}
  int height () const { return m_height;}

  const Pixel8* pixels () const { return m_pixels.data();}
  Pixel8*       pixels ()       { return m_pixels.data();}
  // Intoarce un pointer la inceputul datelor de pixeli

  Pixel8 getPixel (int x, int y) const;
  void setPixel (int x, int y, Pixel8 pixel);
  // Get/set a single pixel

  // Operatii asupra imaginii (doar una pentru prototip)
  virtual CImage8 thumbnail (int reduction) const;

  // constructorul implicit de copiere si atribuirea sunt ok
protected:
  CExAlloc<Pixel8> m_pixels; // Date de pixel
  int           m_width;  // dimensiunile imaginii
  int           m_height;
};
```

Declarația pentru **CImage32** este următoarea:

```
typedef unsigned int Pixel32;

class CImage32
{
public:
  CImage32 ();
  CImage32 (int width, int height);
  // Creaza o imagine nula sau de dimensiunea specificata

  virtual ~CImage32 ();

  int width  () const { return m_width;}
  int height () const { return m_height;}

  const Pixel32* pixels () const { return m_pixels.data();}
  Pixel32*       pixels ()       { return m_pixels.data();}
  // Intoarce un pointer la inceputul datelor de pixeli

  Pixel32 getPixel (int x, int y) const;
  void setPixel (int x, int y, Pixel32 pixel);
  // Citeste sau extrage un singur pixel

  // Operatii asupra imaginii (doar una pentur prototip)
  virtual CImage32 thumbnail (int reduction) const;

  // constructorul implicit de copiere si atribuirea sunt ok
protected:
  CExAlloc<Pixel32> m_pixels; // Pixel data
  int             m_width;  // Image dimensions
  int             m_height;
};
```

Informația de pixeli a imaginilor este gestionată de o clasă specială de gestiune a memoriei CExAlloc. Scrierea unui constructor de copiere sau a operatorului de atribuire nu este necesară deoarece funcțiile implicite sunt suficiente.

Funcția de generare a pictogramelor este următoarea:

```
CImage8 CImage8::thumbnail (unsigned int reduction) const
{
  CImage8 output (width()/reduction, height()/reduction);

  for (unsigned int ty=0; ty<output.height(); ty++) {
    for (unsigned int tx=0; tx<output.width(); tx++) {
      unsigned int sum = 0;
      for (unsigned int y=0; y<reduction; y++) {
        for (unsigned int x=0; x<reduction; x++)
          sum += getPixel (tx*reduction+x, ty*reduction+y);
      }
      output.setPixel (tx, ty, sum / (reduction*reduction));
    }
  }
```

```
  return output;
}
```

Prin compararea codului pentru **CImage8** si **CImage32** se constată că există o similaritate care permite:

- analizarea beneficiilor prin utilizarea derivării pentru simplificarea codului şi maximizarea reutilizării codului;
- analiza avantajelor pe care le-ar aduce utilizarea *template*-urilor pentru eliminarea codului duplicat.

În lipsa *template*-urilor, proiectarea claselor pentru imagini s-ar putea realiza prin derivarea fiecărui tip de imagine dintr-o clasă de bază comună. Clasa **CImage8** poate fi derivată dintr-o clasă **CMonochromeImage** care la rândul ei este derivată din **CImageBase**. Imaginile color pot fi reprezentate de o clasă derivată din clasa **CColorImage**.

Prototipul 2

În cazul prototipului 1 s-a demonstrat că, clasele ce reprezintă imagini de tipuri diferite sunt de fapt foarte asemănătoare. Derivarea reprezintă o soluție posibilă pentru tratarea acestei similarități dar astfel se impune ca toate clasele imagine să facă parte din aceeaşi ierarhie de clase. O altă posibilitate de abordare a acestei similarități o reprezintă template-urile cu scopul de a simplifica proiectarea precum şi de a maximiza reutilizarea codului.

Realizarea prototipului 2 permite *analiza* următoarelor aspecte:

- templateurile se utilizează pentru a rescrie clasa CImage pentru a accepta un parametru de template T ce reprezintă tipul pixelului;
- se introduce un handle astfel încât mai multe clase **CImage<>** să poată utiliza aceeaşi clasă de reprezentare **CImageRep<>**;
- se testează posibilitatea utilizării claselor proiectate la gestionarea de tipuri mai complexe de imagini, cum ar fi imaginile RGB.

Utilizarea templateurilor

Primul pas constă în analiza utilității abordării cu template-uri pentru rezolvarea problemei. Datorită similarității între clasele **CImage8** și **CImage32** se recurge la rescrierea claselor ca o clasă template așa cum este prezentat în figura 4-3:

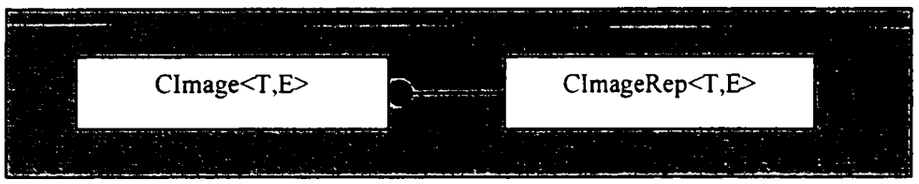


Figura 4-3. Proiectuarea cu ajutorul tremplate-urilor

Conversia clasei CImage într-o clasă template produce următoarea clasă:

```
template<class T> class CImage
{
public:
  CImage  ();
  CImage  (unsigned int width, unsigned int height);
  ~CImage ();

  const T* pixels () const;
  T*       pixels ();

  T getPixel (unsigned int x, unsigned int y) const;
  void setPixel (unsigned int x, unsigned int y, T pixel);

  CImage<T> thumbnail (int reduction) const;
protected:
  CExAlloc<T>   m_pixels;
  unsigned int m_width;
  unsigned int m_height;
};
```

Cele două clase **CImage8** și **CImage32** se definesc după cum urmează:

```
typedef CImage<unsigned char> CImage8;
typedef CImage<unsigned int>  CImage32;
```

Funcția de generarea a pictogramei devine:

```
template<class T>
  CImage<T> CImage<T>::thumbnail (unsigned int reduction) const
  {
    CImage<T> output (width()/reduction, height()/reduction);

    for (unsigned int ty=0; ty<output.height(); ty++) {
```

```
      for (unsigned int tx=0; tx<output.width(); tx++) {
        T sum = 0;
        for (unsigned int y=0; y<reduction; y++) {
          for (unsigned int x=0; x<reduction; x++)
            sum += getPixel (tx*reduction+x, ty*reduction+y);
        }
        output.setPixel (tx, ty, sum / (reduction*reduction));
      }
    }
    return output;
  }
```

În urma analizei funcției, şi anume a liniilor

```
        T sum = 0;
          ...
        sum += getPixel (tx*reduction+x, ty*reduction+y);
```

se constată că, în cazul în care T este **unsigned char**, precizia variabilei **sum** este insuficientă pentru a păstra suma a mai multor pixeli [Str1997].

Problema preciziei calculelor interne are ca soluție utilizarea unui argument suplimentar care specifică dimensiunea internă de pixel utilizată în calcule. Prin urmare, prototipul 2 defineste **CImage<T,E>** unde T reprezintă tipul pixelului iar E reprezintă tipul intern de pixel.

Utilizarea combinației clasa handle + clasa de reprezentare

Procedeul reprezintă practic un contor de referințe ataşat unei clase. Clasa de reprezentare, *clasa rep*, conține implementarea, toate funcțiile şi datele pe când *clasa handle* reprezintă practic un *pointer* la clasa rep [Rom2003].

Clasa handle

In cadrul prototipului 2, **CImage<T,E>** reprezintă clasa *handle* care referă o instanță a clasei **CImageRep<T,E>**:

```
template<class T, class E> class CImageRep; // Forward declaration

template<class T, class E> class CImage
{
public:
  friend class CImageRep<T, E>;
```

```
  CImage (); // Imagine null utila pentru atribuire ulterioara
  CImage (unsigned int width, unsigned int height);
  ~CImage () { m_image->subRef ();}

  CImage            (const CImage& src);
  CImage& operator= (const CImage& src);
  // constructorul de copiere si operatorul de copiere sunt
necesare

  const CImageRep<T, E>* operator -> () const { return m_image;}
  CImageRep<T, E>*       operator -> ()       { return m_image;}
  // Allow access to the rep object

protected:
  CImage (CImageRep<T, E>* rep);
  // Construirea unei imagini din instanta rep

  CImageRep<T, E>* m_image; // Datele propriuzise de imagine
};
```

Analiza secvenței conduce la concluzia că totul se rezumă la apelul metodelor **addRef()** și **subRef()** în cadrul clasei **CImageRep<T,E>**. Pe lângă definițiile constructorului/destructorului, cea mai importată funcție este operator->. Aceasta reprezintă modul de acces la metodele clasei rep [Rom2003]. Acest operator întoarce un pointer la clasa **CImageRep<T,E>** astfel că orice metodă publică poate fi accesată.

Clasa rep

Clasa rep **CImageRep<T,E>** este similară clasei template de imagine prezentate anterior:

```
template<class T, class E> class CImageRep
{
public:
  static CImageRep* gNull ();  // A null image

  CImageRep () : m_width (0), m_height (0), m_ref (0) {}
  // Creaza o imagine nula, utila pentru atribuire ulterioara

  CImageRep (unsigned int width, unsigned int height);
  ~CImageRep () {}

  unsigned int width  () const { return m_width;}
  unsigned int height () const { return m_height;}

  const T* pixels () const { return m_pixels.data();}
  T*       pixels ()       { return m_pixels.data();}

  const T& getPixel (unsigned int x, unsigned int y) const;
```

```
  void        setPixel (unsigned int x, unsigned int y, const T&
pixel);

  // Contorizearea referintelor
  unsigned int ref () const { return m_ref;}      // Number of
references
  void addRef () { m_ref++;}
  void subRef () { if (--m_ref == 0) delete this;}

  CImage<T, E> thumbnail (int reduction) const;

  // constructorul implicit de copiere si atribuirea sunt ok
protected:
  CExAlloc<T>   m_pixels; // datele de pixeli
  unsigned int m_width;  // dimensiunile imaginii
  unsigned int m_height;
  unsigned int m_ref;    // contorul de referinte

  static CImageRep* m_sNull;   // obiectul null de imagine
};
```

Se observă că în principal diferența între clasa rep şi clasa imagine este definiția imaginii *null*.

Funcția **thumbnail()** a clasei **CImageRep<T,E>** devine:

```
template<class T, class E>
CImage<T,E> CImageRep<T,E>::thumbnail (unsigned int reduction)
   const
{
  CImageRep<T,E>* output =
    new CImageRep<T,E> (width()/reduction,
                        height()/reduction);

  for (unsigned int ty=0; ty<output->height(); ty++) {
    for (unsigned int tx=0; tx<output->width(); tx++) {
      E sum = 0;
      for (unsigned int y=0; y<reduction; y++) {
        for (unsigned int x=0; x<reduction; x++)
          sum += getPixel (tx*reduction+x, ty*reduction+y);
      }
      output->setPixel (tx, ty, sum / (reduction*reduction));
    }
  }
  // conversia catra CImage prin apelul constructorului protected
  return output;
}
```

Această abordare diferă de prima încercare de conversie a funcției thumbnail() într-o funcție template. Clasele rep sunt alocate pe heap pe când clasele handle pot fi alocate oriunde. Din acest motiv, pentru alocarea clasei rezultate trebuie utilizat new.

Imaginile RGB

Din abordările considerate, clasele proiectate pot reprezenta doar imagini monocrome. O imagine RGB păstrează informația de pixel pe trei valori corespunzătoare celor 3 componente roşu, verde şi albastru.

O imagine RGB ar putea fi definită după cum urmează:

```
typedef unsigned char Pixel8;
  struct RGB {
    Pixel8 red;
    Pixel8 green;
    Pixel8 blue;
  };
```

Utilizarea clasei template definita anterior nu este suficientă, fiind necesare modificări pentru acomodarea operatiilor cu noul tip de pixel RGB.

Pentru a fi posibilă utilizarea imaginilor RGB este necesară definirea acestor operații precum şi a unui nou tip **RGBPixel32** utilizat în calculele interne pentru evitarea overflow-ului.

```
// tipul de culoare de baza (8:8:8 format)
struct RGB {
  Pixel8 red;
  Pixel8 green;
  Pixel8 blue;

  RGB (Pixel8 b=0) : red (b), green (b), blue (b) {}
};

// definitia interna pentru calcule (32:32:32 format)
struct RGBPixel32 {
  Pixel32 red;
  Pixel32 green;
  Pixel32 blue;

  RGBPixel32 (Pixel32 l=0) : red (l), green (l), blue (l) {}
};

RGBPixel32& operator += (RGBPixel32& s1, const RGB& s2)
{
  s1.red += s2.red;
  s1.green += s2.green;
  s1.blue += s2.blue;
  return s1;
}
```

```
RGB operator/ (const RGBPixel32& s1, int den)
{
  RGB div;
  div.red = s1.red / den;
  div.green = s1.green / den;
  div.blue = s1.blue / den;
  return div;
}
```

Astfel se poate defini o clasă ce reprezintă o imagine RGB:

```
CImage<RGB,RGBPixel32> image;
```

Prototipul 3

Gestionarea eficientă a memoriei devine o necesitate atunci când se manipulează imagini mari [Daw2004] [Sht2000]. În prezent clasa **`CExAlloc()`** realizează gestiunea memoriei utilizată la păstrarea informației de pixeli.

Pentru realizarea separării se creează clasa **`CStorageRep`** ce încapsulează stocarea iar **`CImage<>`** reprezintă clasa ce realizează procesarea imaginii. Conectarea celor două clase se realizează cu ajutorul clasei handle CImageStorage. Proiectul final pentru prototipul 3 este cel din figura 4-4:

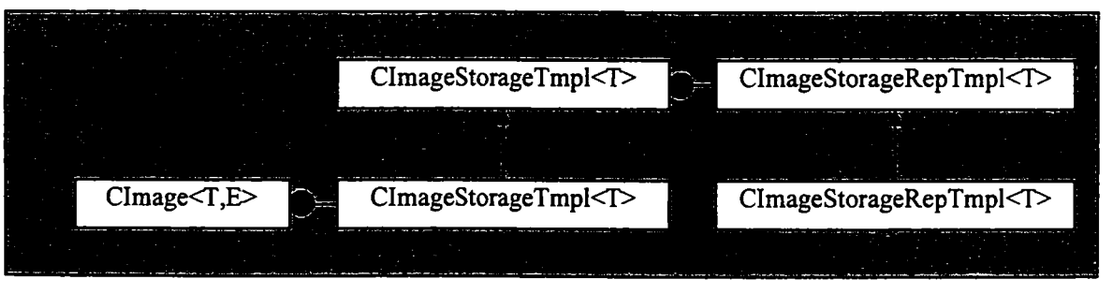


Figura 4-4. Clasa imagine si proiectul de separare a stocării

Astfel prin extinderea prototipului 2 se ajunge la figura 4-5:

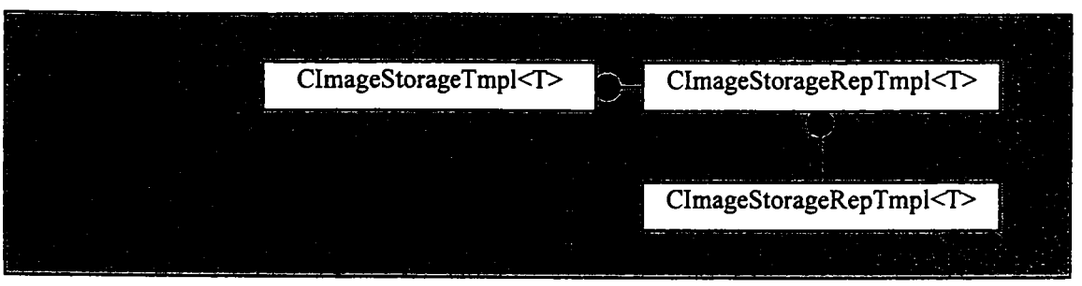


Figura 4-5. Evolutia proiectului (pasul 1)

Clasa comună de bază. Clasele rep de stocare a imaginii

Clasa rep aleasă, **CStorageRep**, conține definiții generice de stocare a imaginilor precum și funcționalitatea specifică unui handle. Prin compararea cu **CImageRep** din prototipul 2 se poate constata asemănarea dintre cele 2 clase. Principala diferență este aceea că **CImageRep<>** alocă memoria cu **CExAlloc<T>** unde T este un tip de date, pe când **CStorageRep** alocă memoria cu ajutorul CExAlloc<unsigned char>. Atunci când o clasă derivată din **CImageRep** va aloca memorie pentru stocarea unei imagini ea va trebui să specifice și numărul de octeți ce reprezintă un pixel.

Din moment ce **CStorageRep** nu este o clasă template, definiția sa se va scrie într-un fișier header iar implementarea într-un fișier sursă. Avantajul faptului că această clasă nu este template îl reprezintă controlul asupra modului său de compilare.

Prin scrierea majorității funcționalităților în cadrul clasei de bază, clasa template va fi una simplă, **CStorageRepTmpl<>**, ce redefinește funcția de acces la pixelii imaginii pentru a returna un pointer T* (corespunzător tipului pixelului).

Clasele handle pentru stocarea imaginilor

Clasa handle **CImageStorage** este asemănătoare cu clasa handle **CImage<>** definită anterior în cadrul prototipului 2.

```
class CImageStorage
{
public:
  CImageStorage (); // A null image storage
  CImageStorage (CStorageRep* rep);
  virtual ~CImageStorage ();

  CImageStorage            (const CImageStorage& src);
  CImageStorage& operator= (const CImageStorage& src);
  // suprascrierea constructorului de copiere si a operatorului
  // de atribuire

  const CStorageRep* operator -> () const { return m_storage;}
  CStorageRep*       operator -> ()       { return m_storage;}

protected:
  CStorageRep* m_storage;
};
```

Diferența majoră este aceea că **CImageStorage** nu este o clasă template ea fiind un handle pentru **CStorageRep**. In continuare se creează clasa template **CImageStorageTmpl<>** derivând **CImageStorage**.

```
template<class T>
class CImageStorageTmpl : public CImageStorage
{
public:
  CImageStorageTmpl () {}
  CImageStorageTmpl (unsigned int width, unsigned int height)
    : CImageStorage (new CStorageRepTmpl<T> (width, height))
  {}

  virtual ~CImageStorageTmpl () {}

  const CStorageRepTmpl<T>* operator -> () const
  { return static_cast<CStorageRepTmpl<T>*> (m_storage);}
  CStorageRepTmpl<T>*       operator -> ()
  { return static_cast<CStorageRepTmpl<T>*> (m_storage);}
};
```

Prototipul 3 reprezintă un design echilibrat: fiecărei clase de bază îi corespunde o clasă template derivată. Această simetrie indică apropierea de design-ul final.

Figura 4-6 prezintă relația dintre aceste clase.

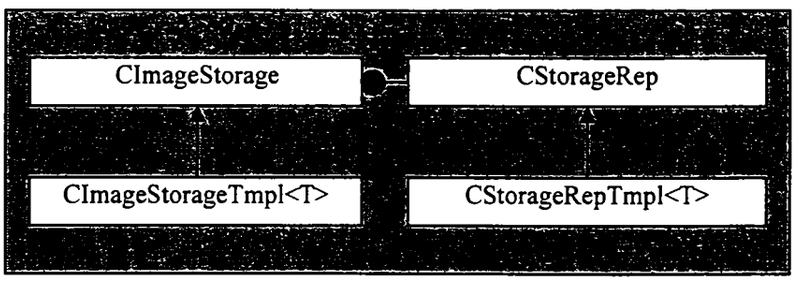


Figura 4-6. Proiectul de separare a obiectului imagine și a stocării

O nouă clasă template, **CImage<>** cu doi parametri de template, va realiza efectiv operațiile de procesare a imaginii.

```
template<class T, class E> class CImage
{
public:
  CImage ();
  CImage (unsigned int width, unsigned int height)
    : m_pixels (width, height) {}
  ~CImage () {}
```

```
  unsigned int width  () const { return m_pixels->width();}
  unsigned int height () const { return m_pixels->height();}

  const T* pixels () const { return m_pixels->base();}
  T*       pixels ()       { return m_pixels->base();}

  const T& getPixel (unsigned int x, unsigned int y) const;
  void     setPixel (unsigned int x, unsigned int y,
                     const T& pixel);

  // operatii asupra imaginii (doar una pentru prototip)
  CImage<T, E> thumbnail (unsigned int reduction) const;

  // constructorul implicit de copiere si de atribuire sunt ok
protected:
  CImage (CImageStorageTmpl<T>& storage);
  // se construieste imaginea din datele stocate

  CImageStorageTmpl<T> m_pixels;  // Datele propriuzise ale
                                  //imaginii
};
```

In cadrul acestui obiect, toate aspectele legate de stocarea şi manipularea pixelilor sunt parte a obiectului de stocare **m_pixels**. **CImage<>** expune doar acea parte a interfeței **CImageStorageTmpl<>** necesară accesului către **width()**, **height()** şi **pixels()**.

Metoda de generare a pictogramei devine:

```
template<class T, class E>
CImage<T,E> CImage<T,E>::thumbnail (unsigned int reduction) const
{
  CImage<T,E> output(width()/reduction, height()/reduction);

  for (unsigned int ty=0; ty<output.height(); ty++) {
    for (unsigned int tx=0; tx<output.width(); tx++) {
      E sum = 0;
      for (unsigned int y=0; y<reduction; y++) {
        for (unsigned int x=0; x<reduction; x++)
          sum += getPixel (tx*reduction+x, ty*reduction+y);
      }
      output.setPixel (tx, ty, sum / (reduction*reduction));
    }
  }

  return output;
}
```

Concluzii

Prin eliminarea referințelor la clasa template se remarcă asemănarea cu metoda clasica **thumbnail()** din cadrul prototipului 1. Această similaritate de organizare a claselor cu simplitatea implementării prototipului inițial demonstrează în fapt obținerea unei proiectari transparente a codului.

In cazul versiunii din cadrul protipului 2, rutinele de procesare a imaginii accesează imaginea curentă cu ajutorul pointerilor. Accesul la pixelii imaginii curente este posibil prin apelul unei metode, în timp ce accesul la o imagine nouă necesită un handle. In cazul prototipului 3 accesul este similar indiferent de tipul imaginii accesate.

4.2.1.6.3 Implementarea unei componente reutilizabile

Problema implică considerarea obiectului **string** din cadrul Standard Template Library. Clasa **std::string** este o clasă extrem de bine proiectată şi conține toate facilitățile necesare manipulării şirurilor de caractere [Lip2005] [Sol2005] [Kal1999].

Software-ul reutilizabil soluționează o problemă generală [Ale2001]. Clasa **string** din biblioteca standard este generică global. Ea poate manipula date de dimensiune **char** precum şi date cu dimensiune diferite pe caracter:

```
typedef basic_string<char> string;
typedef basic_string<wchar_t> wstring;
```

Pentru o mai bună înțelegere a modului de proiectare a codului reutilizabil se va considera o clasă nouă de manipulare a şirurilor. Funcționalitățile sau limitările clasei se determină în urma analizei cazurilor de utilizare.

Scrierea unei clase **string**, reutilizabilă, este un proces relativ uşor pentru că şirurile reprezintă un concept uşor de manipulat. Partea dificilă constă în luarea deciziei cu privire la numărul de facilități ce vor fi implementate. Dacă se implementează o clasă generică de manipulare a şirurilor în cadrul unei biblioteci utilizate de alți programatori, aceştia se pot aştepta la o clasă cu multe facilități cum este clasa CString din cadrul MFC, care are peste 100 de membri publici [Ber2003]. Pe de altă parte, se poate scrie o clasă string ce implementează minimul necesar:

```
class CExString
{
public:
```

```
  CExString  (int size);
  ~CExString ();
  CExString  (const CExString& src);
  CExString& operator= (const CExStringc& src);
  int size () const;
  char& operator[] (int index);
private:
  char* m_P;
};
```

Un astfel de obiect poate fi util pentru o anumită aplicație dar nu poate fi considerat reutilizabil.

Consideratii şi recomandări in completarea definiției codului reutilizabil:

- să ofere funcționalitate cat mai generică şi utilă;
- să poată fi descris cu ajutorul unui număr minim de propoziții, prin folosirea de termeni concreți. În cazul clasei **`string`** se poate spune că aceasta reprezintă un obiect ce manipulează şiruri de caractere, înlocuind funcțiile standard de manipulare a şirurilor.

O componentă reutilizabilă odată scrisă poate fi utilizată fără restricții economisind timp, efort financiar şi uman în dezvoltarea ei ulterioară. Dezvoltarea şi întreținerea codului reutilizabil este însă costisitoare [Sol2005].

Clasa şir binar

Metodologia de implementare a unei componente reutilizabile impune crearea unei clase de manipulare a şirurilor binare. Clasa şir binar este utilă manipulării stream-urilor de date, (de ex. pentru stocarea datelor pe suporturi nonvolatile). Aceste stream-uri pot reprezenta imagini sau conținutul unor obiecte.

Pentru a recunoaşte mai uşor tipul datelor conținute în stream, clasa gestionează date marcate (*tagged*). Acest lucru înseamnă că fiecare element scris în stream este compus din două părți. Prima parte reprezintă tag-ul ce specifică tipul de date din partea a doua. Prin marcarea datelor se asigură interpretarea corectă şi permite citirea stream-ului de date indiferent dacă acesta este cunoscut sau nu.

Obiectul m_BString poate indica următoarele tipuri de date:

- byte (cu semn, 1 octet)
- word (cu semn, 2 octeți)
- integer (cu semn, 4 octeți)
- unsigned integer (fără semn, 4 octeți)

- float (4 octeti)
- double (8 octeți)
- string (şir)
- data
- m_BString

Câmpul de marcare este de 1 octet şi precede datele. Modul de stocare a majorității tipurilor de date este similar cu modul de reprezentare din memorie, şirul fiind stocat ca şi lungime urmată de conținut. Obiectele **m_BString** pot fi înglobate permițând astfel încapsularea altor obiecte binare. Codul definiției clasei este următorul:

```
typedef unsigned char  Pixel8;     // 1-byte
typedef unsigned short Pixel16;    // 2-bytes
typedef int            Pixel32;    // 4-bytes  (unsigned)
typedef unsigned int   Pixel32s;   // 4-bytes  (signed)
class m_BString
{
public:
  m_BString  ();
  ~m_BString ();

  m_BString            (const m_BString& src);
  m_BString& operator= (const m_BString& src);

  size_t      size () const { return m_string.size();}
  const void* base () const { return m_string.c_str();}
  // intoarce un pointer si dimensiunea datelor

  void rewind () { m_offest = 0;}
  // resetarea la inceput a pointerului

  bool eof () const { return m_offest >= m_string.size();}
  // intoarce true daca stream-ul este la capat

  bool match () const { return m_match;}
  // intoarce true daca tipul cerut de date corespunde cu tipul
  // datelor pastrate

  const std::string& str () const { return m_string;}
  // accesul la datele de sir


// operatori de inserare
  m_BString& operator<< (Pixel8   b);
  m_BString& operator<< (Pixel16  w);
  m_BString& operator<< (Pixel32s l);
  m_BString& operator<< (Pixel32  l);
  m_BString& operator<< (float  f);
  m_BString& operator<< (double d);
  m_BString& operator<< (const std::string& s);
```

```
  m_BString& operator<< (const m_BString& bstr);

  void append (const void* data, long size);

  // operatori de extragere
  m_BString& operator>> (Pixel8&   b);
  m_BString& operator>> (Pixel16&  w);
  m_BString& operator>> (Pixel32s& l);
  m_BString& operator>> (Pixel32&  l);
  m_BString& operator>> (float&  f);
  m_BString& operator>> (double& d);
  m_BString& operator>> (std::string& s);
  m_BString& operator>> (m_BString& bstr);

  bool fetch (const void*& data, unsigned int& size);

  std::string dump ();
  // extragerea datelor de la pozitia curenta

private:
  std::string  m_string;
  int          m_offest;
  bool         m_match;

  enum   eTypes   {eNone=0,   ePixel8=1,   ePixel16=2,   ePixel32s=3,
ePixel32=4,
                eFloat=5, eDouble=6, eString=7, eData=8, eBstr=9};
  // tipuri de date suportate. Aceste date nu pot fi modificate
  // dar pot fi adaugate altele noi

  m_BString  (const void* data, unsigned int size);

  void add (eTypes type, const void* data, unsigned int size);
  // adauga datele specificate buffer-ului

  const void* extract (eTypes& type);
  // intoarce un pointer la urmatorul tip de date precum si a
  // tipului lor
  // intoarce null daca se incerca citirea in exces

  Pixel8         readPixel8   (const void* p);
  Pixel16        readPixel16  (const void* p);
  Pixel32s       readPixel32s (const void* p);
  Pixel32        readPixel32  (const void* p);
  float        readFloat  (const void* p);
  double       readDouble (const void* p);
  std::string readString (const void* p);
  // citire din sir

  std::string dumpBString (unsigned int indent);
  // extragere sub forma de dext a continutului unui BString
};
```

Şirul binar se păstrează ca şi **std::string** pentru că funcția **std::string** nu

ține cont de tipul datelor conținute. Pentru a permite utilizarea funcțiilor de prelucrare a șirurilor C se adaugă caracterul nul la sfârșitul șirului [Str1997].

In cazul utilizării tipului **Pixel16**, pentru adăugarea de date noi se pot folosi următoarele funcții:

```
m_BString& m_BString::operator<< (Pixel16 w)
{
  add (ePixel16, &w, sizeof (w));
  return *this;
}

void m_BString::add (eTypes type, const void* data,
                     unsigned int size)
{
  // adauga tipul datelor
  Pixel8 t = static_cast<Pixel8>(type);
  m_string.append (reinterpret_cast<char*>(&t), sizeof (Pixel8));

  // adauga datele
  m_string.append (reinterpret_cast<const char*>(data), size);
}
```

Operatorul de inserare, **operator<<**, apelează în fapt funcția **add()** pentru adăugarea efectivă a datelor. Operatorul de extragere a datelor, **operator>>**, este mult mai complex. În momentul începerii citirii de date din *stream:*

- se citeşte tag-ul pentru determinarea tipului de date ce vor fi extrase;
- realizează conversia la **Pixel16**.

Poziția curentă din cadrul șirului se păstrează permițând parcurgerea stream-ului fără modificarea acestuia.

```
m_BString& m_BString::operator>> (Pixel16& w)
{
  eTypes type;
  w = 0;

  bool match = false;
  const void* p = extract (type);
  if (p == 0) return *this;

  switch (type) {
  case ePixel8:
    w = readPixel8 (p);
    break;
  case ePixel16:
    w = readPixel16 (p);
    match = true;
```

```
    break;
  case ePixel32s:
    w = (Pixel16) readPixel32s (p);
    break;
  case ePixel32:
    w = (Pixel16) readPixel32 (p);
    break;
  case eFloat:
    w = (Pixel16) readFloat (p);
    break;
  case eDouble:
    w = (Pixel16) readDouble (p);
    break;
  case eString:
    w = (Pixel16) atoi (readString(p).c_str());
    break;
  default:
    // tip necunoscut
    break;
  }

  m_match &= match;
  return *this;
}
```

Concluzii. Clasa **m_BString** reprezintă o clasă ce implementează funcționalități de bază pentru manipularea datelor binare şi satisface toate definițiile pentru a fi un obiect reutilizabil. Ca dezavantaj se poate remarca faptul că, clasa este portabilă doar pe sisteme cu aceeaşi arhitectură. Astfel datele copiate pe un sistem în ordinea *litte-endian* sunt citite greşit pe un sistem *big-endian*. Prin urmare se poate afirma că, clasa **m_BString** nu este reutilizabilă pe tipuri diferite de sisteme: codul este portabil dar nu şi fişierele de date.

4.3. Proiectarea elementelor de interfață grafică cu utilizatorul

In cadrul paragrafului sunt tratate problemele referitoare la utilizabilitate respectiv stabilirea unor criterii de realizare a interfețelor cu utilizatorul, de o importanță primordială pentru proiectare.

4.3.1. Utilizabilitate

În standardul ISO/DIS 9241-11 [ISO1995], utilizabilitatea este definită ca fiind „măsura în care poate fi folosit un produs de utilizatori pentru a-şi atinge efectiv scopurile, cu eficiență dar şi cu satisfacție într-un anumit context de utilizare." In cadrul acestei definiții, efectivitatea se referă la gradul de atingere a obiectivelor, eficiența se referă la efectivitatea în relație cu resursele necesare efectuării sarcinii iar satisfacția se referă la acceptabilitate şi confort. Altfel spus, *utilizabilitatea* reprezintă măsura uşurinței de utilizare a unui produs în scopul efectuarii activităților pentru care acesta a fost creat [Coo2003] [Cog2004].Termenul de *‚utilizabilitate'*, în contextul dezvoltării de aplicații software, reprezintă abordarea în care utilizatorul, şi nu aplicația, se afla in centrul procesului de dezvoltare [Gal2002].

Această filozofie, denumită *proiectare orientată spre utilizator*, ia în considerare nevoile utilizatorului încă de la începutul procesului de proiectare şi le impune ca cerințe care trebuie luate în considerare în cadrul tuturor deciziilor de proiectare.

Figura 4-6 prezintă un *model stratificat* al utilizabilității:

- primul nivel reprezintă cei trei indicatori de apreciere, relativ greu de evaluat, eficiența, eficacitatea şi satisfacția, care caracterizează utilizabilitatea;
- nivelul doi prezintă un set de indicatori de utilizare ce pot fi evaluați în practică;
- modurile (nivelul trei) sunt utilizate în euristica utilizabilității şi ajută la îmbunătățirea indicatorilor menționați anterior. Cele mai bune rezultate din punct de vedere al utilizabilității se obțin prin folosirea optimă a modurilor, apelând la cele trei domenii de cunoştințe: modelul utilizatorului, cunoştințele de proiectare şi modelul activității.

Soluțiile de succes în ceea ce priveşte proeictarea interfetelor cu utilizatorul permit utilizatorului să-şi atingă scopurile în mod eficient şi cu un număr minim de erori [Wel2000]. Ele sunt utilizabile pe domeniul lor specific dar nu există soluții general valabile [Hoh2003]. Utilizabilitatea (*Usability)* este strâns legată de maniera de proiectare a sistemului software.

In continuare se analizează câteva din principalele domenii de interferență ale proiectării sistemului software cu proiectarea interfeței cu utilizatorul.

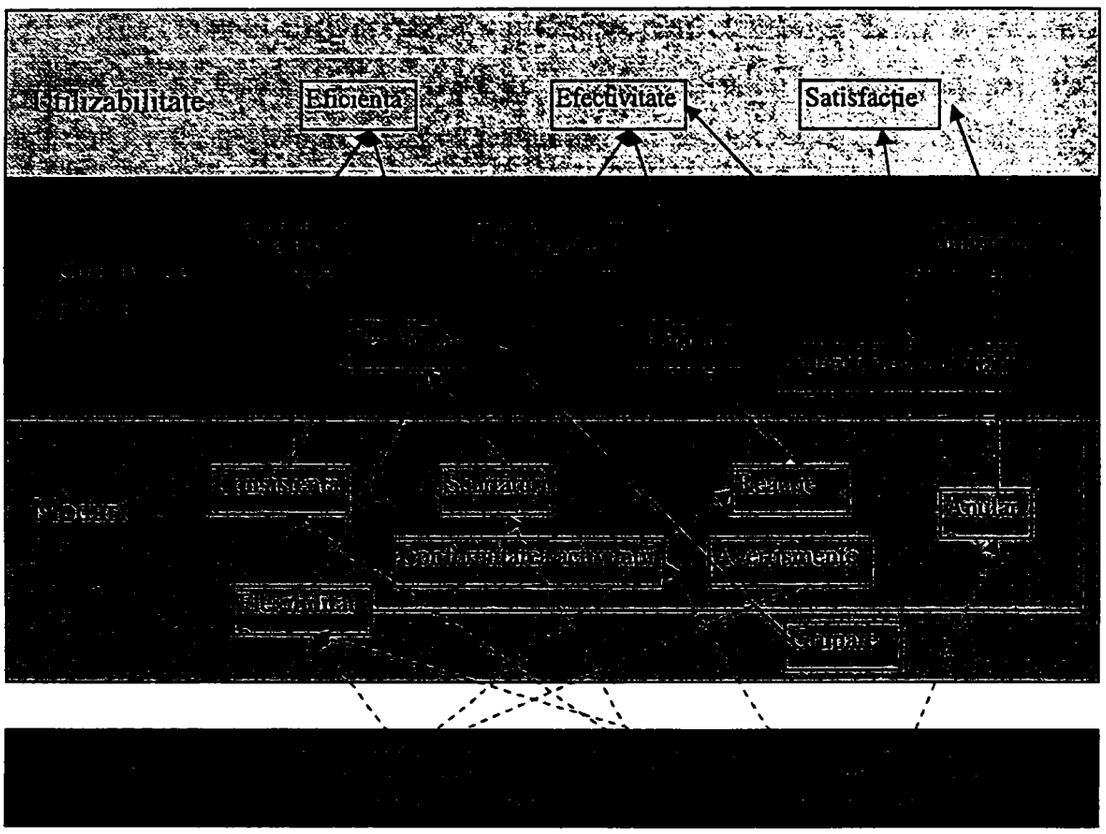


Figura 4-6. Vedere stratificată a conceputului de utilizabilitate

Cardinalitatea

Cardinalitatea descrie numărul entităților ce participă în cadrul unei relații (de exemplu numărul de elemente ale unei multimi, numarul de valori posibile pentru un atribut sau proprietate).

Cardinalitatea reprezintă un element important atât în cadrul general al proiectării software cât şi în proiectarea interfeței cu utilizatorul. Modurile de memorare, procesare şi reprezentare diferă de la caz la caz, de exemplu atunci când este implicat un număr mic de entități sau un număr mare de entități [Cog2004] [Joh2000]. Pe măsură ce cardinalitatea creşte, sporesc şi cerințele de reprezentare vizuală a datelor şi a interacțiunilor lor.

Reacția (Feedback)

Reactia *poate fi definită ca mesaj informațional - ca răspuns la acțiuni ale utilizatorului* - cum ar fi afişarea meniului ca răspuns la selecția utilizatorului, ca indicatoare de progres în cazul unei procesări de durată etc.

Una din cele mai importante euristici asociate cu crearea de sisteme utilizabile este reprezentată de moduri diferite realizare de reacții (feedback) către utilizator. Proiectarea sistemului software trebuie realizată ținând cont de cererea de reacție (feedback) [Coo2003] [Amb1998].

Pentru a exemplifica influența modului de proiectare a sistemului software asupra proiectării interfeței cu utilizatorul, se consideră un indicator de progres care sugerează faptul că sistemul procesează o cerere sau o tranzacție de durată. Porțiunea de sistem care efectuează cererea trebuie să fie proiectată astfel încât să raporteze periodic starea de procesare. Acest lucru poate fi uşor realizat în cazul unei operații asupra unui număr finit de elemente, unde se cunoaşte durata de procesare a unui singur element. In situații mai complexe, există operații pentru care nu se poate anticipa durata de procesare.

Aceste variabile decid alegerea tipului de indicator de progres.

O formă specială de reacție (feedback) o reprezintă *validarea în timp real*, în care sistemul validează parțial sau integral, în timp real, fiecare valoare introdusă. Un exemplu curent poate fi considerat un câmp de introducere a unei valori strict numerice. Un exemplu mai complex poate consta într-un ecran de introducere a datelor care validează parțial datele sau care activează/dezactivează diferite elemente de interfață pe baza datelor introduse anterior.

In general, dispozitivele utilizate pentru accesarea unui sistem influențează utilizabilitatea într-un mod neuniform [Gal2002]. *Caracteristicile fiecărui dispozitiv trebuie analizate individual pentru a asigura cel mai mare grad de utilizabilitate.*

Din experiență se cunoaşte faptul că un individ poate aprecia utilitatea unui obiect şi îl poate folosi ca instrument fără să-l cunoască în toată complexitatea sa [Coo2003]. Acest fapt este posibil datorită însuşirilor sale cognitive. Modelul creat este suficient pentru a explica interacțiunile om-instrument dar nu suficient de detaliat încât să explice funcționarea internă a instrumentului.

*Discrepanța dintre **modelele mentale** și cele de **implementare** este mai pronunțată în cazul aplicațiilor software, unde complexitatea implementării poate face aproape imposibilă perceperea de către utilizator a legăturii dintre acțiunile sale și reacțiile aplicației.*

Modele explicite de utilizator

*Un mod de a utiliza **modelele mentale** este acela de a crea un model explicit, corespunzător modului de înțelegere a utilizatorului raportat la sistem, și de a modifica în timp comportamentul sistemului urmând evoluția percepției utilizatorului.*

Această tehnică prezintă un grad înalt de generalitate, de exemplu: procesorul de text, care corectează automat greşelile de ortografie; magazinul virtual de cărți, care prezintă recomandări pe baza achizițiilor anterioare etc.

Suport pentru planul de lucru

Proiectarea software şi piața de profil interacționează permanent şi evoluează împreună [Amb1998]. Acesta este motivul pentru care inițial software-ul nu prezintă suport pentru planul de lucru (*workflow support*). Pe măsură ce piața şi utilizarea software-ului evoluează, echipele de dezvoltare pot capta în timp cele mai bune practici de utilizare pe care le pot implementa sub diferite forme simple (cum sunt experții sau sistemele de ajutor animate) de suport al planului de lucru. Suportul se bazează pe înțelegerea modelelor mentale ale utilizatorilor.

Suportul tranzacțiilor

Multe dintre operațiile efectuate într-un sistem pot fi considerate ca tranzacții [Amb1998], de exemplu cum ar fi introducerea unui caracter într-un program ce oferă anularea operațiilor. Inevitabil, tranzacțiile afectează interfața cu utilizatorul dar şi reciproc.

Răspunsul la erori

Modul de colectare, validare şi răspuns la erorile de introducere a datelor de către utilizator sunt determinate de interacțiunea complexă dintre arhitectura de bază a sistemului, interfața cu utilizatorul şi activitățile utilizatorului. De exemplu, un anumit compilator prezintă o listă a erorilor de compilare într-o fereastră separată. In urma selectării unei erori, editorul se va localiza pe linia de cod corespunzătoare din cadrul codului sursă [Ber2003]. Acest lucru nu poate funcționa în cadrul unei aplicații web, care trebuie să răspundă la erori

prin intermediul unei ferestre de notificare (câmpurile sunt marcate cu roşu sau indicate cu asterisc). Alegerile făcute în ceea ce priveşte modul de prezentare a erorilor afectează proiectarea software.

Internaționalizarea şi localizarea

Internaționalizarea produsului software dezvoltat (interfata cu utilizatorul) presupune ca sistemul software trebuie să fie capabil să suporte implementarea in diferite limbi (limba matena a beneficiarului) si respectiv localizarea formatarii specifice a datelor (formatarea valorilor monetare, a numerelor etc.). Sistemele de operare precum şi platformele pe care rulează prezintă infrastructura necesară determinării limbii utilizatorului [Coo2003] [Gal2002].

Localizarea unei aplicații trebuie să se țină cont de tot ce este adresat utilizatorului: mesajele de eroare, mesajele de informare, cutiile de dialog, formatele de introducere/afişare, fişierele jurnal şi chiar numele de funcții API externe. Toate acestea sunt candidate pentru internaționalizare.

Cateva din solutiile recomandate pentru evitarea erorilor de proiectare sunt:

- conversia cutiilor de dialog de dimensiune fixă;
- adaptarea elementelor de dialog pentru utilizarea de text de lățime variabilă;
- opțiunea utilizării de text scris şi de la dreapta la stânga;
- utilizarea de seturi de caractere pe 2 sau mai mulți octeți.

In general, aplicațiile localizate necesită un program de testare foarte riguros, realizat cu clienți şi/sau parteneri experți în limbile cu cea mai mare raspandire intre clienti. Majoritatea organizațiilor nu posedă diversitatea necesară testării traducerilor realizate de partenerul de localizare.

Revocarea acțiunilor

In cadrul majorității sistemelor software, utilizabilitatea poate fi îmbunătățită substanțial dacă proiectarea permite ca acțiunile care necesită timpi îndelungați de procesare sau consumă multe resurse să fie întrerupte. Acest element permite utilizatorilor să aibă mai mult control asupra sistemului software şi îmbunătățeşte performanțele globale ale sistemului (sistemul nu este angajat în procese inutile) [Amb1998] [Joh2000]. Opțiunea de revocare a

unei operații poate fi utilă în special administratorilor de sistem în activitatea de ajustare a performanțelor sistemului, mai ales în cazul sistemelor multi-utilizator.

In cadrul acestor sisteme, crearea de acțiuni revocabile impune o proiectare atentă a sistemului. Dacă un client inițiază o acțiune revocabilă, acțiunii trebuie sa i se atribuie un identificator cu care sa poata fi identificată ulterior. Serverul trebuie extins pentru a fi capabil saasocieze acțiunilor identificatori unici, sa asigure terminarea acțiunilor, precum şi sa recupereze resursele de sistem. Fiecare acțiune trebuie evaluată din punct de vedere al justificării implementării posibilității de revocare.

Anularea acțiunilor

Pe lângă opțiunea de revocare, utilizabilitatea poate fi îmbunătățită dacă utilizatorului i se oferă posibilitatea de a anula rezultatul unei acțiuni – *undo*. Proiectarea avansată a sistemelor, în special din domeniul managementului tranzacțiilor precum şi în cazul utilizării protocoalelor de tip publicare-subscriere, face posibilă anularea chiar şi în cazul unui sistem partajat (între mai multe componente tehnologice şi utilizatori) [Cog2004].

Posibilitatea de a anula acțiunile utilizatorului reprezintă principalul instrument de explorare a interfeței cu utilizatorul din cadrul aplicației [Amb1998].

Funcția de anulare a acțiunilor ar trebui implementată ca o funcție globală pe aplicație.

Acțiunile în acest sens sunt de două tipuri:

- incrementale – care includ modificarea datelor;
- procedurale – care operează asupra datelor dar nu le modifică.

Tipurile de anulare a acțiunilor sunt:

- singulară – la o apelare se anulează ultima acțiune. La o apelare dublă se anulează ultima anulare.
- multiplă – se pot anula succesiv mai multe acțiuni anterioare în ordine temporală inversă.

Compensarea tranzacțiilor

Anumite tipuri de tranzacții nu pot fi revocate sau anulate. In acest caz, sistemul trebuie extins prin utilizarea de tranzacții compensatorii.

Timpi de aşteptare

Sistemele ce necesită sesiuni cu un alt sistem, sau cu utilizatorul, presupun diferiți timpi de aşteptare. Alegerea corectă a valorilor implicite a timpilor de asteptare poate contribui semnificativ la perceperea utilizabilității aplicației. Alegerea unei valori greşite poate scădea gradul de utilizabilitate, poate consuma inutil resurse de sistem sau poate conduce la creşterea volumului de muncă depusă de utilizator pentru a-şi îndeplini obiectivele.

Redresarea după căderi ale sistemului

Sistemele pot cădea în diferite moduri. Alegerile din cadrul proiectării software pot mări sau reduce şansele de apariție a unor căderi. Daca s-a produs o cădere (crash) a sistemului, capacitatea lui de a se redresa afectează direct utilizabilitatea.

Indiferent de numărul de condiții de cădere tratate, utilizatorul trebuie atenționat asupra efectelora negative potențiale şi amplorii lor fiind necesar în acelaşi timp să i se recomande proceduri de redresare şi acțiuni viitoare de prevenire.

4.3.2. Criterii pentru realizarea interfețelor grafice cu utilizatorul

Interfata cu utilizatorul reprezintă singura modalitate de comunicare intre utilizator pe de o parte şi program pe de altă parte. Practic, o interfață cu utilizatorul constă într-o colecție de elemente grafice, cum ar fi butoanele şi check-boxurile din cadrul unei ferestre, pe care utilizatorul le manipulează pentru a comunica cu programul [Coo2003] [Tho2004] [Amb1998]. Din acest punct de vedere, interfața este deseori definită prin noțiunea *front-end* al unei aplicații iar programul însuşi este definit prin noțiunea *back-end*. Figura 4-7 prezintă locul interfeței în cadrul unei aplicații.

Figura 4-7. Locul interfeței în cadrul unei aplicații.

Câteva recomandări care se constiuie ca şi criterii destinate proiectantilor interfețelor grafice cu utilizatorul sunt formulate în continuare:

- să evite zgomotul vizual (elementele vizuale superflue, altele decat cele esențiale care comunică precis funcții ale aplicației) şi respectiv aglomerarea. In general

interfețele trebuie să utilizeze forme geometrice simple, contururi minimale şi culori nesaturate. Pe măsură ce interfața este proiectată, ea trebuie simplificată vizual cât mai mult;

- să utilizeze contrastul, similaritatea şi stratificarea pentru a face posibilă distingerea şi organizarea elementelor. Contrastul este utilizat la diferențierea elementelor active/manipulabile de cele pasive/nemanipulabile ale interfeței. O interfață vizuală se bazează pe modele vizuale. Diferențierea se poate face dimensional, prin nuanțare sau spațial. Prin stratificare se înțelege utilizarea unor repere vizuale pentru elementele individuale. Diferite atribute vizuale controlează perceperea straturilor – culorile închise, nesaturate, depărtează iar culorile deschise, saturate, apropie;

- să ofere o structurare şi un parcurs vizual pentru fiecare nivel de organizare. Interfețele sunt compuse din elemente vizuale şi comportamentale grupate. Gruparea poate fi făcută pozițional (proximitate), prin aliniere, prin culoare (intensitate, nuanță, temperatură, saturație), prin textură, prin dimensiune, prin formă. Alinierea elementelor vizuale este unul din modurile cheie prin care o interfață este organizată şi sistematizată. Simetria conferă de asemenea un echilibru vizual al interfeței;

- să afişeze imagini consistente, asociate contextului. Utilizarea pictogramelor şi a altor elemente ilustrative permite proiectarea unei interfețe sugestive, accesibilă utilizatorului. O bună cunoaştere a modelelor mentale ale utilizatorului constituie o bază solidă pentru generarea unui limbaj textual şi vizual adecvat într-o interfață.

  Cel mai dificil aspect este acela de a reprezenta un concept abstract într-un limbaj vizual, printr-o pictogramă. In acest caz trebuie să se recurgă la utilizarea idiomurilor şi a *tooltip*-urilor. La realizarea pictogramelor trebuie avute în vedere următoarele recomandări:

  - gruparea funcțiilor vizuale (spațial, cromatic) cu elemente comune menite să ofere un context;

- reprezentarea simultană a acțiunii şi obiectului pentru o mai bună înțelegere a pictogramei;
- evitarea metaforelor şi a reprezentărilor care pot induce confuzie;
- simplificarea pictogramelor, adică evitarea excesului de detalii;
- reutilizarea elementelor deja cunoscute, ori de câte ori este necesar, pentru reducerea timpului de învățare.

Elementele cu comportament diferit trebuie să fie vizual distincte.

In *Visual Display of Quantitative Information,* [Tuf2001] se stipuleaza şapte principii majore necesare unei afişari vizuale satisfăcătoare a informației:

1) Impunerea de *comparații vizuale*. Utilizatorii trebuie să aibă posibilitatea de a compara variabile înrudite sau stări inițiale şi finale. Comparația oferă un context ce face informația mult mai valoroasă şi mult mai inteligibilă pentru utilizator;
2) *Indicarea cauzalității*. Cauzele şi efectele trebuie corelate clar în cadrul informației grafice;
3) *Afişarea variabilelor multiple*. Afişarea datelor care oferă informații legate de variabile multiple trebuie făcută simultan fără a altera claritatea;
4) Afişarea pe acelaşi ecran a *textului, graficii şi a datelor*. Altfel, diagramele necesită legende pentru a fi decodate, fiind mai puțin eficiente şi solicită utilizatorului o procesare cognitivă adițională;
5) Asigurarea *calității, relevanței şi integrității* conținutului. Orice informație afişată trebuie să contribuie la activitatea utilizatorului menită atingerii scopului propus;
6) Obiectele să fie afişate *adiacent spațial*, nu temporal.;
7) Să nu se decuantifice *datele cuantificate*. Cu toate că utilizarea de grafuri şi diagrame îmbunătățeşte percepția informațiilor cantitative, este recomandabil ca valorile numerice reprezentate să nu fie ascunse.

In proiectarea interefetelor cu utilizatorul pot fi indentificate cateva elemente de reper care influențează decisiv calitatea şi performantele aplicatiei finale.

Modelarea cu ajutorul scenariilor este utilă în identificarea facilităților aplicației. Scenariile trebuie analizate din punct de vedere al:

- redundanței;

- gradului de specificitate;
- utilității;
- uşurinței de utilizare;
- completitudinii.

Paradigmele proiectarii interfețelor cu utilizatorul

Există trei paradigme dominante în cadrul proiectării conceptuale şi vizuale a interfețelor cu utilizatorul [Coo2003] [Amb1998]:

- *interfețe orientate pe implementare* Interfața reprezintă o reflexie a modului în care aplicația este construită;
- *interfețe metaforice* – se bazează pe conexiunile intuitive pe care le face utilizatorul între reperele vizuale din cadrul interfeței şi funcția corespunzătoare. Nu mai este necesară învățarea modului de funcționare a aplicației;
- *interfețe idiomatice*.

Utilizarea de idiomuri

Un *idiom* reprezintă o entitate care dacă este luată ca atare nu are nici o legătură cu contextul în care este utilizat [Cog2004]. Aplicațiile utilizează din plin idiomurile ce au o anumită semnificație pentru utilizatori (un exemplu de idiom este bara de aplicații (*taskbar*) din Windows).

Idiomurile pot fi şi sub formă de pictograme sau simboluri.

Utilizarea graficii si a pictogramelor

Utilizarea în exces a graficii reprezintă o încarcare contraproductivă a interfeței [Gal2002]. Pictogramele reprezintă soluția ideală de a minimiza cantitatea de text afişat pe ecran.

Echilibrul dintre text şi simboluri

In cadrul unei aplicații, mesajele textuale trebuie folosite cât mai restrâns şi concis. Se recomandă utilizarea de simboluri reprezentand imagini care pot sugera acțiuni sau procese.

Utilizarea Controalelor

Controalele sunt obiecte manipulabile situate pe ecran care permit comunicarea între utilizator şi aplicație [Coo2003].

Din punct de vedere al scopurilor utilizatorului, controalele se pot împărți în patru grupe de bază:

- controale imperative – utilizate la inițierea unor funcții;
- controale de selecție – utilizate la selecția de opțiuni şi date
- controale de introducere de date;
- controale de afişare – utilizate pentru manipularea vizuală directă a aplicației

Diagrame de stare pentru interfețe

In etapa de concepție a interfeței cu utilizatorul trebuie evitată descrierea textuală *prin recurgerea la reprezentări sub formă de diagrame de stare (v. paragraful 3.5). In esență, stările reprezentate sunt ecrane ale aplicației legate între ele prin săgeți care reprezintă diferite opțiuni. Figura 4-8 prezintă un exemplu de diagramă de stare pentru interfață.*

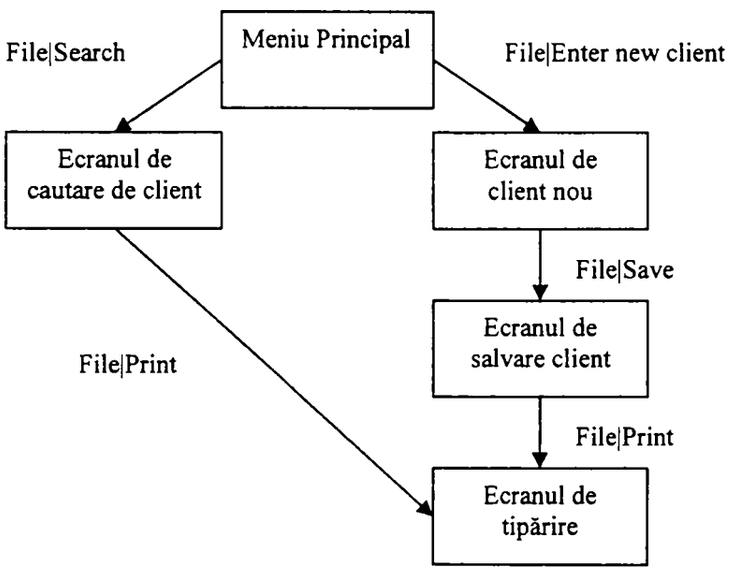


Figura 4-8. Diagramă de stare pentru interfață

Amplasarea ferestrelor şi a cutiilor de dialog

- Se consideră că o fereastră este bine concepută dacă:controalele din cadrul ferestrei urmăresc idiomurile standard;
- amplasarea ferestrei are sens în context;
- anumite părți ale ferestrei sunt accesibile utilizatorului;

Utilizarea cutiilor de dialog

Cutiile de dialog suspendă interacțiunea curentă om-aplicație [Coo2003]. Ea inițiază o conversație cu utilizatorul prezentând informație și așteptând un răspuns. După ce cutia se închide, aplicația își continuă cursul normal. Cutiile de dialog sunt utile în prezentarea funcțiilor și setărilor rar utilizate și de asemenea pentru concentrarea de informație asociată unui obiect cum ar fi proprietățile sale.

Utilizarea marcajelor

Pentru a eficientiza activitatea utilizatorului, aplicația poate deveni mai rapidă prin accelerarea operațiilor pe care le poate efectua utilizatorul. Acest lucru se poate realiza cu ajutorul marcajelor (*bookmarks*). Ele pot identifica:

- cele mai recente fișiere utilizate;
- serie de locații din cadrul unui document;
- un element recent utilizat din cadrul unui meniu;
- listă de documente deschise.

Marcajele trebuie:

- să indice clar elementul la care se referă;
- să fie configurabile și editabile;
- să existe posibilitatea de sortare;
- să poată fi organizate pe categorii;
- să fie persistente între sesiunile de lucru;
- să poată fi exportate și importate;
- să existe posibilitatea de căutare.

Navigarea în cadrul aplicației

Prin *navigare* se înțelege procesul de parcurgere a opțiunilor aplicației pentru a îndeplini o sarcină. Un software poate fi caracterizat ca eficient navigabil dacă permite utilizatorului să acceseze informația dorită printr-un număr minim de pași [Coo2003].

Pentru a obtine cursivitate, interacțiunea cu aplicația trebuie să fie transparentă. Pentru a realiza o interfață cât mai transparentă se pot lua în considerare următoarele recomandari:

- satisfacerea modelelor mentale;
- execuția directă, evitarea interacțiunii cu utilizatorul cum ar fi mesajele de confirmare;

- păstrarea instrumentelor cât mai accesibil;
- afişarea de feedback permanent.

Se pot formula urmatoarele recomandari pentru minimizarea acțiunilor auxiliare:

- utilizatorul nu trebuie obligat să meargă într-o altă fereastră pentru a efectua o funcție ce afectează fereastra curentă;
- utilizatorul nu trebuie să memoreze locurile unde păstrează informația;
- utilizatorul nu trebuie să fie nevoit să redimensioneze inutil ferestrele. Dacă o fereastră este deschisă, ea trebuie să fie automat redimensionată corespunzător;
- utilizatorul nu trebuie obligat să repoziționeze ferestre pe ecran;
- utilizatorul nu trebuie să reintroducă setările personale (font, culori etc);
- utilizatorul nu trebuie obligat să introducă arbitrar informații, doar pentru a completa un formular, acesta trebuie să poată omite detalii.

Navigația poate fi de mai multe tipuri şi pe mai multe nivele. In continuare sunt *prezentate critic* cele mai întâlnite tipuri de navigație:

- navigația în cadrul mai multor ferestre sau ecrane – aceasta poate dezorienta utilizatorii prin solicitarea unei atenții sporite;
- navigația în cadrul panourilor din cadrul unei ferestre – poate fi optimizată prin amplasarea adiacent sau suprapusă a panourilor;
- navigația în cadrul barelor de instrumente sau a meniului – minimizarea mişcărilor de mouse se poate realiza printr-o amplasare spațială adecvată. Instrumentele ce sunt utilizate frecvent în conjuncție trebuie amplasate apropiat. Meniurile necesită un efort suplimentar din partea utilizatorului pentru că, conținutul acestora nu este vizibil decât după selectie. Funcțiile frecvent utilizate trebuie oferite în cadrul barelor de instrumente;
- navigația în cadrul informației afişate într-un panou sau cadru – aceasta se poate efectua prin defilare (deplasare), legături (sărituri) şi mărire/micşorare. Mărirea/micşorarea şi deplasarea (*panning*) sunt specifice în cazul explorării informației 2D şi 3D. Atunci când sunt cuplate, aceste operații pot crea utilizatorului dificultăți majore de navigare, cum sunt dificultățile în perceperea tridimesionalității pe un ecran 2D.

Pentru îmbunătățirea navigației se propun urmatoarele *recomandări*:

- reducerea numărului locațiilor de navigare prin minimizarea paginilor, a ferestrelor, a controalelor şi a defilării astfel încât să fie suficiente pentru ca utilizatorul să-şi atingă scopul;
- oferirea de marcaje prin includerea de indicatoare/obiecte persistente pentru a ghida navigarea utilizatorului: meniuri, bare de instrumente, palete de instrumente etc.;
- oferirea de vederi de ansamblu grafice sau textuale care ajută la orientarea utilizatorului;
- asocierea corespunzătoare a controalelor la funcții, descriind relația dintre un element de interfata, obiectul afectat şi rezultatul dorit.;
- adaptarea interfeței la cerințele utilizatorului presupune organizarea interfeței pentru a minimiza navigarea, amplasarea celor mai frecvent utilizate funcții şi controale în locurile cele mai accesibile;
- evitarea ierarhiilor. Cu toate că ierarhiile sunt cele mai durabile instrumente ale unui programator, navigarea ierarhiilor abstracte de către utilizatori este destul de dificilă.

Selecția

Principala problemă în cadrul interfețelor cu utilizatorul o reprezintă ordinea executării comenzilor. Aproape fiecare comandă constă dintr-o operație (verb) şi unul sau mai mulți operanzi (obiecte). Se poate specifica întâi verbul şi apoi obiectul sau invers. In prima situație se pune problema modului de indicare a terminării identificării obiectelor, în timp ce pentru a doua, în care ordinea este obiect-verb, terminarea identificării obiectelor este clară, prin selectarea operației. Astfel se pot efectua chiar mai multe operații asupra aceluiaşi set de obiecte. In acest caz trebuie dezvoltat un mecanism de identificare, marcare şi memorare a operanzilor aleşi. Obiectele selectate trebuie să fie indicate clar utilizatorului.

Manipularea obiectelor 3D

Cea mai semnificativă problemă în cazul interacțiunii 3D pe un ecran 2D este lipsa unei paralaxe, abilitatea binoculară de a percepe adâncimea. O altă problemă este cea a obiectelor mai apropiate ce acoperă obiectele mai îndepărtate. In continuare este realizata o

prezentare critica a celor mai utilizate solutii dedicate acestui tip de probleme [Gal2002] [Zha2004] [Coo2003]:

- puncte de vedere multiple –este cea mai veche metodă de rezolvare a ambelor probleme dar este şi cea mai puţin eficientă din punct de vedere interactiv. Dezavantajul acestei soluţii constă în aceea că utilizatorul trebuie să privească în mai multe locuri simultan pentru a determina poziţia unui obiect;
- planul de bază, adâncime (*depth cue)*, umbre şi poli – reprezintă idiomuri care contribuie la o mai bună percepţie spaţială..
- ghiduri şi alţi indicatori vizuali
- cadre de sârmă (*wireframe)* şi cutii de delimitare (*bounding boxes*) care rezolvă problemele de vizibilitate a obiectelor.

Introducerea datelor în cazul aplicaţiilor 3D implică anumite probleme, cum ar fi:

- *deplasarea obiectelor* – una din problemele fundamentale ale manipulării directe într-o proiecţie 2D a unei scene 3D este aceea de translatare a mişcărilor 2D ale cursorului din planul ecranului într-o mişcare în spaţiul virtual 3D.
- *selecţia* – dacă obiectele sunt reprezentate în *wireframe*, identificarea obiectului selectat de către utilizator din multitudinea obiectelor suprapuse devine dificilă. Ca soluţie se poate oferi o listă sau o ierarhie de obiecte din care utilizatorul poate selecta;
- *rotirea obiectelor, respectiv mişcarea, rotirea şi zoom-ul camerei* – o altă problemă specifică aplicaţiilor 3D este numărul de funcţii de manipulare spaţială ce pot fi efectuate. Obiectele pot fi repoziţionate, redimensionate şi deformate pe 3 axe, pot fi rotite în jurul a 3 axe,. in plus, punctul de privire al camerei poate fi rotit în jurul său sau al unui punct de focalizare(pe 3 axe), câmpul de vedere al camerei putand fi mărit sau micşorat. Aceste facilităţi, în cadrul aplicaţiilor 3D, fac obligatorie utilizarea de meta-taste şi scurtături de tastatură.

Satisfacerea tuturor categoriilor de utilizatori

La proiectarea unei interfeţe cu utilizatorul trebuie avut în vedere ca:

- activităţile cele mai frecvente şi importante să fie cele mai vizibile;
- activităţile mai puţin utilizate să fie accesibile prin meniu sau bare de instrumente;
- utilizatorii să-şi poată personaliza meniurile şi barele de instrumente.

Deciziile cu privire la includerea de caracteristici cum ar fi *Experții* (*Wizards*) în cadrul aplicației rezultă din înțelegerea utilizatorului [Gal2002].

Stabilirea nivelului de complexitate în cadrul unei singure interfețe este o sarcină dificilă. Soluția poate fi găsită doar prin înțelegerea felului diferit în care utilizatorii mânuiesc conceptele şi activitățile noi.

O interfață bine echilibrată se orientează spre satisfacerea utilizatorului intermediar, ea trebuie să nu intimideze utilizatorul începător sau să subestimeze utilizatorul avansat.

Obiectele persistente din cadrul interfeței sunt acele obiecte care prin modificare pot influența considerabil navigarea în cadrul interfeței aplicației [Coo2003] [Gal2002]. Dintre acțiunile mai cunoscute, salvarea şi încărcarea pot fi considerate ca obiecte persistente. *Personalizarea* interfeței presupune decorarea obiectelor persistente. Una din formele de personalizare a interfeței constă în deplasarea pictogramelor din cadrul unei bare de instrumente. *Configurarea* interfeței presupune mutarea, adăugarea sau ştergerea de obiecte persistente. Proiectantul trebuie să ofere utilizatorilor posibilitatea de configurare a interfeței conform nivelui lor.

Navigarea meniurilor şi a barelor de instrumente

Secvența de cod asociată fiecărui punct de meniu sau buton trebuie să fie alcătuită doar dintr-un apel de funcție. Aceasta separă funcționalitatea interfeței de cea a aplicației şi reprezintă un avantaj în cazul includerii în aplicație a unui limbaj de macro-comenzi. Personalizările utilizatorului trebuie salvate pentru a fi utilizate din nou la reapelarea aplicației [Amb1998] [Joh2000] [Coo2003].

In proiectarea structurii barelor de instrumente, având în vedere faptul că ele vor fi utilizate cu precădere de către utilizatorii de nivel intermediar se recomandă ca ele să fie personalizabile şi multiple.

Fiecare buton dintr-o bară de instrumente trebuie să ofere sugestii (*tooltips*) pentru a da explicații referitoare la funcția asociată.

Pentru o navigabilitate eficientă a sistemului se pot formula urmatoarele recomandari:

- utilizarea de denumiri şi aranjamente standardizate;
- evitarea utilizării excesive a meniurilor în cascadă;
- includerea de scurtături de tastatură pentru punctele de meniu;

- evitarea modificării meniului prin adăugarea/eliminarea de puncte de meniu..
- dacă unui punct de meniu i se asociază un *check-mark*, rularea comenzii asociate schimbă starea punctului de meniu.
- opțional, textul unui punct de meniu ce va deschide o cutie de dialog va fi urmat de trei puncte (...);
- utilizarea de separatori în cazul meniurilor lungi;
- utilizarea unei pictograme comune pentru bara de instrumente şi meniu;
- meniurile *pop-up* să fie scurte (maximum 7-8 elemente);
- funcționalitățile din cadrul unui meniu *pop-up* să fie prezente şi în alte locuri;
- trebuie evitată utilizarea de meniuri in cascadă în cadrul meniurilor *pop-up*.

Date şi informații, rapoarte şi tipărituri

Datele se referă la valorile ce sunt stocate în memorie în timp ceinformația se referă la modul în care datele sunt interpretate de către utilizator [Coo2003]. După interpretare, punere într-un anumit context şi asociere de semnificații datele devin informație.

Intr-un raport, informația comunicată trebuie să fie:

- uşor accesibilă şi uşor de găsit;
- informația importantă să fie subliniată;
- cât mai completă;
- cât mai clară;
- să răspundă întrebărilor utilizatorului.

Tratarea excepțiilor

Erorile reprezintă situații neaşteptate care suprimă rularea unei aplicații (de exemplu împărțirea la zero) [Sol2005] [Sie2003].

Exceptiile reprezintă erori ce pot apare în timpul rulării aplicației (*run-time*) dar care pot fi corectate (de exemplu tentativa de a scrie un fişier pe un mediu de stocare plin) [Sol2005] [Sie2003].

O tratare corespunzătoare a erorilor constă în informarea utilizatorului la apariția ei. Intreruperea aplicației fără nici un avertisment trebuie să fie imposibilă prin proiectare. Utilizatorul trebuie să fie împiedicat să-şi continue activitatea după apariția unei erori fatale.

Pentru a rezulta o *proiectare robustă* a sistemului, se pot formula următoarele **recomandări**:

- Tratarea excepțiilor trebuie implementată astfel încât aplicația:
  - să nu moară;
  - să nu afişeze mesaje ostile utilizatorului;
  - să nu lase resurse deschise.
- Mesajele care anunță apariția unei erori trebuie să prezinte informații utile pentru determinarea cauzei;
- trebuie evitată ‚aruncarea' de excepții în cadrul constructorilor. In acest caz este mai indicată utilizarea unei variabile de stare;
- Nu trebuie scris cod în cadrul constructorului. Operațiile necesare trebuie scrise în cadrul unei funcții tip Init() ce poate fi apelată; Astfel excepțiile pot fi ‚aruncate' iar destructorul poate fi apelat în siguranță;
- Pentru ‚prinderea' problemelor din cadrul unui constructor se poate utiliza un bloc *try/catch*. Curățarea se poate face în cadrul blocului *catch* unde excepția este ‚aruncată' din nou pentru a permite ‚prinderea' excepției în cadrul codului apelant.
- Utilizarea excepțiilor în cadrul destructorului unei clase trebuie evitată sau tratată în cadrul destructorului cu un bloc *try/catch*.

Notificările şi confirmările

Asemenea mesajelor de eroare, notificările şi confirmările opresc cursul normal al aplicației, dar ele nu raportează disfuncționalități. O *notificare* informează utilizatorul despre acțiunile aplicației iar *confirmările* permit utilizatorului evitarea unei acțiuni [Joh2000] [Coo2003]. Utilizarea în exces a acestor tipuri de mesaje nu este de dorit, ele trebuie înlocuite, cand este cazul, cu idiomuri mai utile.

Notificările - In cazul în care aplicația generează o notificare, aceasta trebuie să fie afişată în aceeaşi fereastră în care se desfăşoară acțiunea şi nu separat, într-o cutie de dialog.

Confirmările - Mesajele de confirmare vin din partea aplicației şi nu a utilizatorului, astfel ele *reprezintă o reflexie a modelului de implementare* şi nu fac parte din scopurile utilizatorului. Pentru a le face funcționale, confirmările trebuie să apară doar atunci când este foarte probabil ca utilizatorul să aleagă 'Nu' sau 'Revocare'.

Eliminarea confirmărilor este oportună în următoarele situații:

- executarea unei acțiuni fără a fi necesară confirmarea;

- reversibilitatea acțiunilor;
- oferirea de reacții modeless care pot ajuta utilizatorul în evitarea greşelilor.

4.3.3. Reguli de proiectare a elementelor de interfață personalizate

Prin utilizarea de elemente grafice personalizate se urmăreşte o uniformizare a modului de reprezentare a elementelor de interfață cu două motivații principale:

- utilizatorul se va simți într-un mediu familiar indiferent de modulul sau programul utilizat din cadrul sistemului;
- se obține o relativă independență a interfeței cu utilizatorul față de sistemul de operare pe care rulează aplicația în cazul portării acesteia precum şi o delimitare netă între elementele specifice aplicației şi cele ale sistemului de operare.

Regulile prezentate în continuare au fost concepute de către autor şi stau la baza realizării elementelor de interfață personalizate din cadrul capitolului 5.

Crearea sau editarea unei cutii de dialog este posibilă prin utilizarea *editorului de resurse* din cadrul mediului de programare (Microsoft Visual C++) (Figura 4-9). Pentru fiecare element de interfață personalizat, modificările asupra proprietăților standard sunt ilustrate într-o figură reprezentând cutia de dialog de proprietăți a editorului de resurse.

- fiecărui element de interfață i se atribuie un identificator unic (ID) care îi sugerează funcția. Datorită faptului că ID-urile elementelor trebuie să fie diferite doar în cadrul aceleiaşi cutii de dialog, se recomandă utilizarea unor ID-uri generice, reutilizate în fiecare cutie de dialog, ex. IDC_BUTTON1, IDC_BUTTON2, IDOK, IDCANCEL etc.

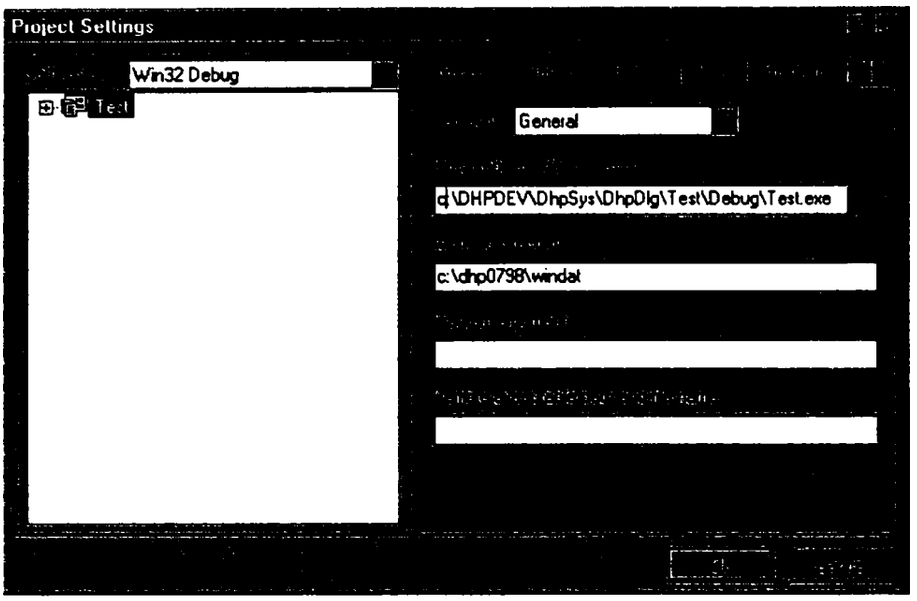


Figura 4-9. Configurarea parametrilor de proiect ai mediului de programare

- la creare se specifică identificatorul cutiei precum şi limba comunicării. Datorită faptului că selectarea limbii în care sunt afişate mesajele nu este automată, în funcție de parametrii de compilare sau de sistemul de operare pe care va rula aplicația, se va alege tipul neutru (implicit) de limbă (Figura 4-10). Selectarea limbii se face prin program.

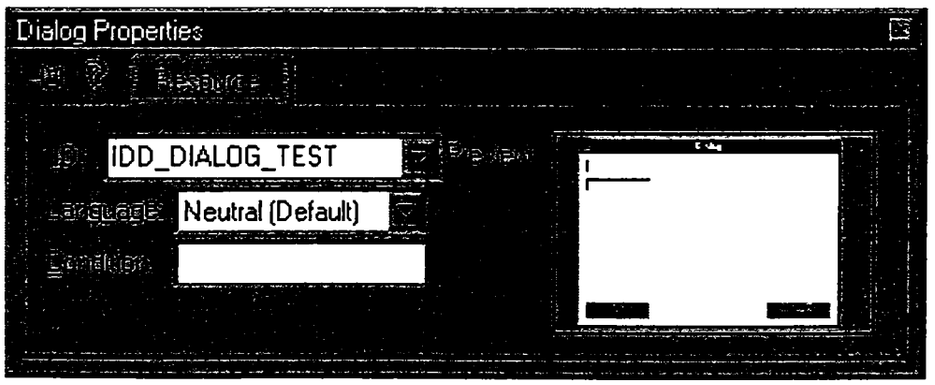


Figura 4-10. Proprietățile unei cutii de dialog

- adăugarea butoanelor se face în mod obişnuit, cu ajutorul editorului de resurse. În secțiunea de stiluri a proprietăților butonului se va indica faptul că desenarea se realizează prin program şi nu de către sistemul de operare (Figura 4-11). Un model de dispozitiv interactor echivalent unui buton este sintetizat în paragraful 3.5.2.

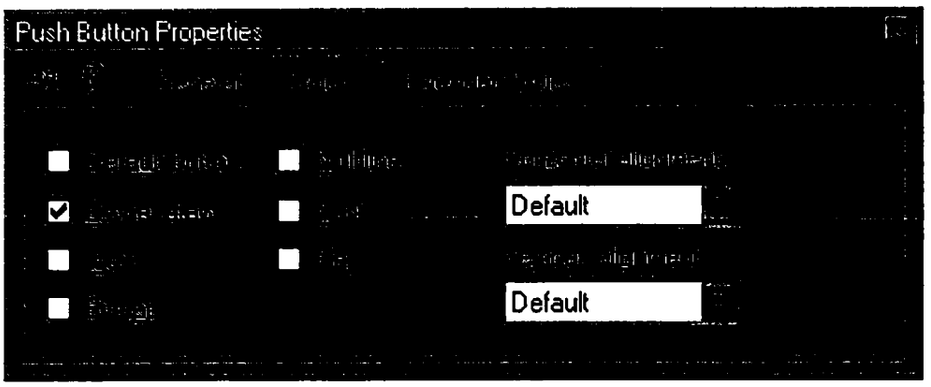


Figura 4-11. Proprietățile unui buton

- pentru afişarea unei imagini în locul textului de pe un buton se selectează *check-box*-ul corespunzător. Afişarea imaginii dorite se va realiza mai târziu prin cod. Textele prezente în cadrul cutiei de dialog sunt utilizate la inițializarea fişierelor ce vor fi traduse în alte limbi (Figura 4-12).

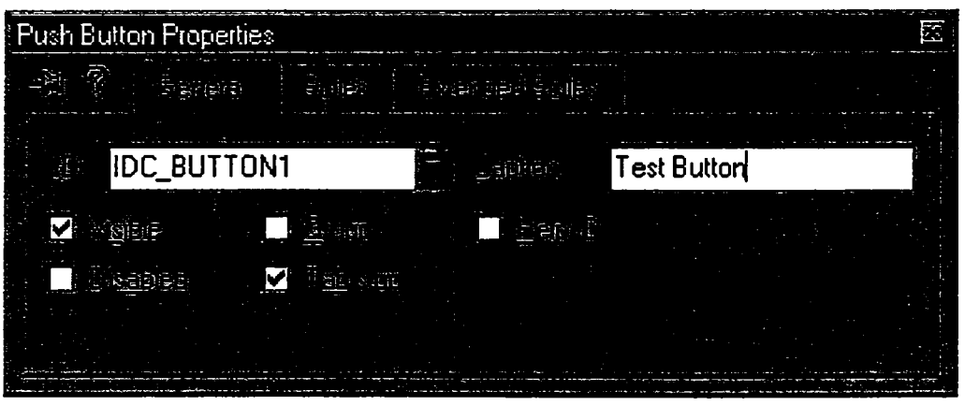


Figura 4-12. Setarea identificatorului şi etichetei unui buton

- comparativ cu echivalentul lor în Windows, elementele de introducere de date (*edit controls*) utilizate prezintă o funcționalitate mult diferită. Acestea au fost plasate în cadrul cutiei de dialog ca elemente personalizate (*custom control*) (Figura 4-13). Pentru identificarea lor se utilizează identificatori de tipul IDC_EDIT1, IDC_EDIT2 etc. Textul asociat nu prezintă importanță pentru că el nu va fi afişat. Pentru identificarea şi ataşarea de cod elementului, acestuia i se atribuie clasa CDhpEdit. Stilurile şi stilurile extinse sunt cele implicite.

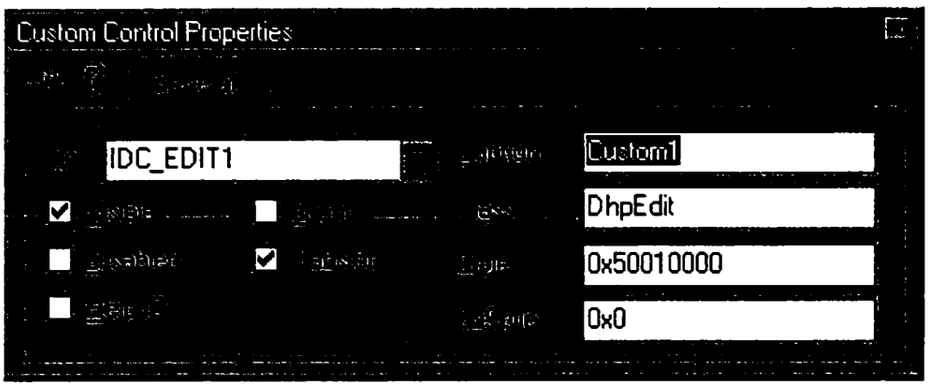


Figura 4-13. Elementul de introducere de date

- *drop-list*-urile utilizate diferă semnificativ prin imagine şi funcționalitate față de echivalentul lor Windows. Acestea, precum *edit-control*-urile, sunt definite ca elemente personalizate. Diferențierea se realizează prin tipul ID-urilor atribuite IDC_DROPLIST1, IDC_DROPLIST2 etc. şi prin codul stilului (Figura 4-14). Clasa este tot DhpEdit.

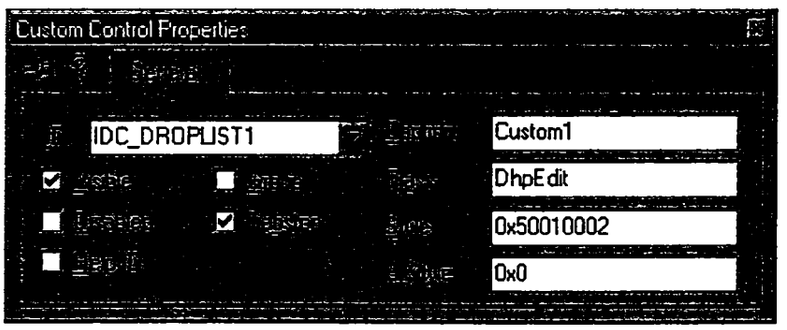


Figura 4-14. Elementul *Drop List*

- adăugarea unui element *dropdown* este identică cu cea a unui *droplist* diferența fiind codul (Figura 4-15).

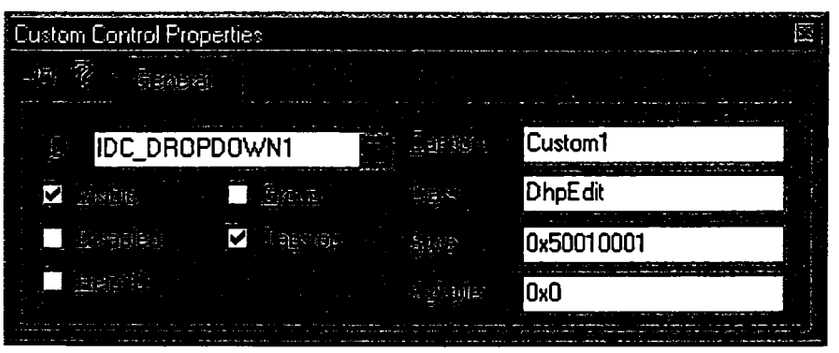


Figura 4-15. Definirea elementului *Drop Down*

- s-au creat funcționalități și stiluri adiționale noi care pot fi atașate unui element. Acestea sunt:
  - buton ajutător pentru măsurători pentru fiecare element de interfață. Se obține prin adăugarea valorii stilului elementului. Ex. un *edit control* va avea valoarea sau un *droplist*;
  - buton de ajutor;
  - buton *Browse*;
  - buton de apelare a calculatorului;
  - evitarea numerotării elementelor dintr-un droplist;
  - ascunderea liniilor de contur: de sus, jos, stânga și dreapta. Această facilitate este foarte utilă construirii de tabele în care elementele de la marginile tabelului prezintă linii de contur doar spre exteriorul tabelului;

- butonul de ajutor, de măsurare, de *browse* și cel de invocare a calculatorului nu pot fi utilizate în același timp la același *edit control*.

- mesajele text statice nu necesită o abordare specială întrucât sunt utilizate cele standard de sistem. Identificatorii lor sunt de tipul IDC_STATIC1, IDC_STATIC2 etc.

- imaginile ce pot fi afișate în cadrul unei cutii de dialog pot fi de tipul bitmap sau meta-fișier. ID-urile utilizate sunt de aceeași formă cu cele ale mesajelor text statice (Figura 4-16). Tipul ales al imaginii din cadrul parametrilor generali este de *Enhanced Metafile* sau *Bitmap*. Pentru cazul imaginilor *bitmap* se selectează și opțiunea de centrare a imaginii.

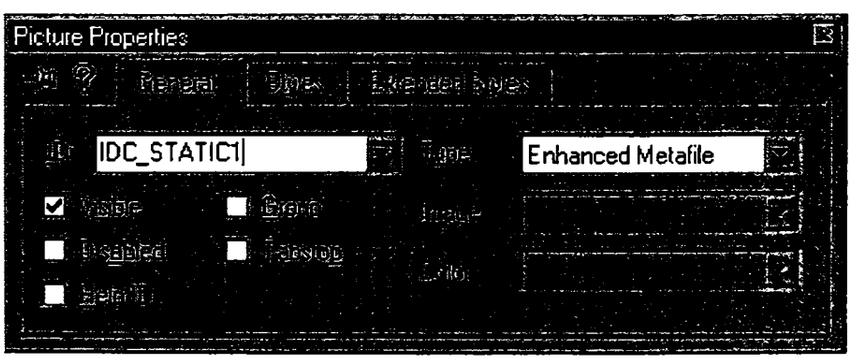


Figura 4-16. Proprietățile unui element de tipul imagine

- *check-box*-urile sunt diferite de cele standard ale sistemului de operareastfel că se utilizează din nou un element personalizat (*custom control*) (Figura 4-17). ID-ul utilizat este de forma IDC_CHECKBOX1, IDC_CHECKBOX2 etc. Clasa elementului este DhpCheckBox. Un model de dispozitiv interactor echivalent unui *check-box* este sintetizat în paragraful 3.5.2.

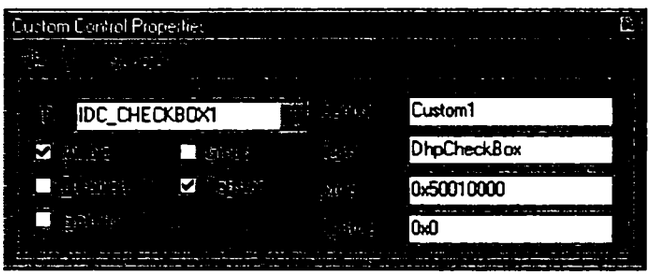


Figura 4-17. Element de tip *check-box*

- tipurile de *check-box*-uri au fost specificate cu ajutorul codurilor de stil:
  - *Check box – text*: text aliniat la dreapta;
  - *Text - check box*: text aliniat la stânga;
  - *Check box – text*: text aliniat la stânga;
  - *Text – check box*: text aliniat la dreapta;
  - *Check box* cu trei stări;
  - *Check box* asemănător unui buton.

4.4. Programarea vizualizării utilizând biblioteca grafică OpenGL

Pentru a crea o imagine ce reprezintă o scenă tridimensională este necesară parcurgerea a două activități: *modelarea* și *randarea (rendering)*.

*Prin **modelare** se generează un model care reprezintă descrierea unui obiect utilizat de către sistemul grafic.*

*Prin **randare** se primesc modelele ca intrări, și ca ieșiri se generează valori de pixeli pentru imaginea finală.*

De aceasta ultimă problemă se ocupă OpenGL și ea este abordată în cadrul paragrafului curent, care realizează un studiu și o analiză ale elementelor esentiale din cadrul programării OpenGL, necesare realizării aplicației propuse în capitolul 5. Soluțiile considerate pentru problemele abordate în acest paragraf sunt prezentate sub forma unor algoritmi ce vor fi implementati în cadrul aplicației. Elementele sintetizate în cadrul paragrafului se pot constitui ca un ghid util proiectanților de interfețe care utilizează OpenGL.

Biblioteca grafică OpenGL a fost creată de compania SGI [Wri2004] și este utilizată cu precădere în cadrul aplicațiilor științifice și de proiectare asistată de calculator.

Dintre motivele care au stat la baza alegerii OpenGL ca și limbaj grafic pentru implementarea facilităților din cadrul aplicației se pot mentiona:

- oferă un set complet de funcții de desenare 3D;
- funcționează eficient pe o gamă largă de arhitecturi grafice;
- este intuitiv și uşor de utilizat;
- este independent de platforma hardware sau de implementarea *driver*-ului pe care rulează aplicația;
- există pe toate platformele mai importante: Apple Macintosh, toate versiunile de Microsoft Windows, aproape toate variantele de Unix inclusiv Linux sau chiar și platformele mobile prin OpenGL ES.

4.4.1. Utilizarea OpenGL sub Microsoft Windows

OpenGL reprezintă un set de funcții API, unde interacțiunea cu utilizatorul și afișarea într-o fereastră pe ecran se realizează de către mediul gazdă. Pentru a facilita acest parteneriat, fiecare mediu oferă extensii ce fac legătura între OpenGL pe de o parte și

gestiunea ferestrelor şi interacţiunea cu utilizatorul pe de altă parte [Shr2003] [Mcr2005]. Aceste extensii asociază comenzile de desenare OpenGL cu o anumită fereastră. De asemenea sunt necesare funcţii pentru selectarea modurilor de *buffering*, adâncimii de culoare precum şi a altor caracteristici de desenare.

In cazul sistemului de operare Microsoft Windows, în cadrul Windows API a fost adăugat un set nou de funcţii [Ast2004]. Acestea au prefixul **`wgl`** şi sunt exportate din biblioteca **`opengl32.dll`**. In Windows există mai multe implementări ale OpenGL [Seg2004] [Shr2003] [Shr2004]:

- Implementare generică. Aceasta reprezintă o implementare software ce nu utilizează hardware 3D.
- *Installable Client Driver (ICD)*. Reprezintă interfaţa hardware originală oferită pentru Windows NT. Această implementare trebuie să realizeze întregul *pipeline* OpenGL utilizând o combinaţie între software şi hardware-ul specific pentru care a fost scrisă.
- *Mini-Client Driver (MCD)*. Reprezintă un compromis între implementarea software şi cea hardware.
- *Mini-Driver*. Acesta realizează o conversie între API-ul OpenGL şi API-ul 3D proprietar hardware-ului. Acesta nu este un display-driver propriu-zis;
- *Extended OpenGL*. Implementarea Microsoft a OpenGL API oferă un set limitat de funcţii ce sunt apelate în cazul în care nu există hardware 3D instalat. Acest set de funcţii implementează doar specificaţia OpenGL 1.1. Accesul la funcţiile specifice versiunilor ulterioare necesită paşi adiţionali.

Randare în ferestre multiple

La desenarea obişnuită în cadrul unei ferestre se utilizează funcţii GDI iar pentru identificarea ferestrei se utilizează ca argument al funcţiei API un *device context* (DC). Funcţia **`WinMain`** realizează iniţializarea iar funcţia **`WndProc`** procesează mesajele de fereastră. Desenarea propriu-zisă se realizează la primirea mesajului **`WM_PAINT`** prin încadrarea codului de desenare intre apelurile funcţiilor **`BeginPaint`** şi **`EndPaint`**.

Pentru identificarea contextului de desenare, OpenGL utilizează un identificator de *rendering context* (RC). Acesta este similar în multe privinţe cu DC din cadrul GDI [Ast2004].

Conceptul de DC în Windows este limitat în cazul graficii 3D pentru că a fost proiectat pentru aplicații grafice 2D. Natura unui DC depinde de natura dispozitivului. Orice fereastră sau dispozitiv ce va afișa grafică 3D are mult mai multe caracteristici decât adâncimea de culoare, mai ales în cazul dispozitivelor hardware de afișare [Fin2004] [Han2005]. Înainte ca OpenGL să poată desena într-o fereastră, aceasta trebuie configurată corespunzător cerințelor de randare cum ar fi: randare software sau hardware, randare cu *single* sau *double buffering*, utilizarea unui *buffer* de adâncime. După selectarea acestor caracteristici, fereastra nu mai poate fi modificată.

Caracteristicile 3D ale unei ferestre se setează o singură dată, imediat după creare. Aceste caracteristici se reunesc sub denumirea de format de pixel şi sunt grupate în structura **PIXELFORMATDESCRIPTOR**, având următoarea componență:

```
typedef struct tagPIXELFORMATDESCRIPTOR {
WORD nSize;             // dimensiunea structurii
WORD nVersion;          // versiunea structurii (trebuie sa fie 1)
DWORD dwFlags;          // proprietatile bufferului de pixeli
BYTE  iPixelType;       // tipul datelor de pixeli (RGBA sau Color
Index)
BYTE   cColorBits;          // numarul de biti de culoare pentru
bufferul de culoare
BYTE  cRedBits;         // numarul de biti pentru rosu
BYTE  cRedShift;        // marimea translatiei pentru rosu
BYTE  cGreenBits;       // numarul de biti pentru verde
BYTE  cGreenShift;      // marimea translatiei pentru verde
BYTE  cBlueBits;        // numarul de biti pentru albastru
BYTE  cBlueShift;       // marimea translatiei pentru albastru
BYTE  cAlphaBits;       // numarul de biti pentru destinatia alpha
BYTE   cAlphaShift;        // marimea translatei pentru destinatia
alpha
BYTE  cAccumBits;       // numarul de biti din accumulation buffer
BYTE   cAccumRedBits;      // numarul de biti pentru rosu pentru
accumulation buffer
BYTE   cAccumGreenBits; // numarul de biti pentru verde pentru
accumulation buffer
BYTE  cAccumBlueBits;  // numarul de biti pentru albastru pentru
accumulation buffer
BYTE    cAccumAlphaBits; // numarul de biti pentru alpha din
accumulation buffer
BYTE  cDepthBits;       // numarul de biti pentru depth buffer
BYTE  cStencilBits;     // numarul de biti pentru stencil buffer
BYTE  cAuxBuffers;      // numarul de buffere auxiliare
BYTE  iLayerType;       // ignorat
BYTE  bReserved;        // numarul de plane overlay si underlay
DWORD dwLayerMask;      // ignorat
DWORD  dwVisibleMask;      // culoarea de transparenta a planului
underlay
DWORD dwDamageMask;     // ignorat
```

```
} PIXELFORMATDESCRIPTOR;
```

Fanioanele corespunzătoare proprietăților *buffer*-ului de desenare sunt cele sintetizate in tabelul urmator:

PFD_DRAW_TO_WINDOW	Desenarea se face în fereastră
PFD_DRAW_TO_BITMAP	Desenarea se face într-un bitmap
PFD_SUPPORT_GDI	Desenarea se face cu GDI
PFD_SUPPORT_OPENGL	Desenarea suportă OpenGL
PFD_GENERIC_ACCELERATED	Desenarea este accelerată cu ajutorul unui driver MCD
PFD_GENERIC_FORMAT	Desenarea este realizată de o implementare software.
PFD_NEED_PALETTE	Desenarea se face cu paletă de culori limitată.
PFD_NEED_SYSTEM_PALETTE	Indică faptul că OpenGL suportă desenare în modul cu 256 de culori.
PFD_DOUBLEBUFFER	Desenarea se realizează cu două *buffer*-e.
PFD_STEREO	Desenarea este stereoscopică
PFD_DEPTH_DONTCARE	Acest fanion este utilizat doar la interogarea formatului de pixel. Pentru optimizare, implementarea nu alocă *buffer*-ul de desenare.
PFD_DOUBLE_BUFFER_DONTCARE	Acest fanion este utilizat doar la interogarea formatului de pixel. Indică faptul că nu se va utiliza *double-buffer*.

Enumerarea formatelor de pixel

Formatul de pixel pentru o fereastră este identificat printr-un index. O implementare exportă un număr de formate de pixel care pot fi utilizate [Seg2004]. Pentru a alege un format de pixel în cadrul unei ferestre, se procedează la selectarea unui format dintre cele exportate de către driver. Pentru determinarea caracteristicilor unui anumit format se utilizează funcția **DescribePixelFormat**:

```
PIXELFORMATDESCRIPTOR pfd;          // descriptor de format de pixel
int nFormatCount;                    // numarul de formate de pixel
exportate
. . .

// obtinerea numarului de formate de pixel
// necesita un device context
pfd.nSize = sizeof(PIXELFORMATDESCRIPTOR);
nFormatCount = DescribePixelFormat(hDC, 1, 0, NULL);

// obtinerea fiecarui format de pixel
for(int i = 1; i <= nFormatCount; i++)
    {
    // obtinerea descriereii fiecarui format de pixel
    DescribePixelFormat(hDC, i, pfd.nSize, &pfd);

. . .
. . .
}
```

Selectarea şi setarea unui format de pixel

Enumerarea tuturor formatelor de pixel precum şi alegerea celui potrivit cerințelor este o operație greoaie. Windows oferă o funcție, **ChoosePixelFormat**, menită să simplifice acest proces prin specificarea caracteristicilor necesare pentru formatul de pixel [Mcr2005], care va determina cea mai potrivită configurație şi va indica indexul corespunzător pentru formatul de pixel. Acest index se specifică mai departe prin intermediul funcției **SetPixelFormat**:

```
int nPixelFormat;
. . .
static PIXELFORMATDESCRIPTOR pfd = {
  sizeof(PIXELFORMATDESCRIPTOR), // dimensiunea acestei structuri
  1,                             // versiunea structurii
  PFD_DRAW_TO_WINDOW |           // desenare in fereastra
  PFD_SUPPORT_OPENGL |           // support pentru apeluri OpenGL
  PFD_DOUBLEBUFFER,              // mod Double buffer
  PFD_TYPE_RGBA,                 // mod de culoare RGBA
  32,                            // colori pe 32 de biti
```

```
  0,0,0,0,0,0,                    // neutilizat la selectia unui mod
  0,0,                            // neutilizat la selectia unui mod
  0,0,0,0,0,                      // neutilizat la selectia unui mod
  16,                             // dimensiunea depth buffer
  0,                              // fara stencil
  0,                              // fara buffere auxiliare
  0,                              // rezervat
  0,                              // fara plane overlay sau underlay
  0,                              // rezervat
  0,                          // fara culoare transparenta
  0};                             // neutilizat

// alegerea unui format de pixel care se potriveste cel mai bine cu
cel descries in pfd
// pentru contextual dat
nPixelFormat = ChoosePixelFormat(hDC, &pfd);

// Setarea formatului de pixel in device-context
SetPixelFormat(hDC, nPixelFormat, &pfd);
```

O aplicație obișnuită de Windows poate consta din mai multe ferestre. Pentru a identifica fereastra unde se vor aplica comenzile OpenGL aceasta trebuie să fie asociată unui RC (RenderingContext).

Un RC se creează prin apelul funcției **wglCreateContext** care primește un singur parametru, DC-ul asociat unei ferestre cu un format de pixel valid. Tipul unui RC este **HGLRC**:

```
HGLRC hRC;      // rendering context-ul OpenGL
HDC hDC;     // device-context-ul ferestrei
. . .
// selectarea si setarea formatului de pixel
. . .
hRC = wglCreateContext(hDC);
```

Un RC este generat pentru a fi compatibil cu fereastra pentru care a fost creat. Se pot utiliza mai multe RC-uri în cadrul aceleiași aplicații. Doar un singur RC poate fi activ la un moment dat în cadrul unui *thread*. Când un RC este activat, el devine curent și are asociat un DC. Pentru asocierea unui RC unei ferestre se utilizează funcția **wglMakeCurrent**:

```
void wglMakeCurrent(HDC hDC, HGLRC hRC);
```

Model de utilizare

Orice aplicație Windows pornește cu funcția **WinMain**. In cadrul acestei funcții se înregistrează tipul ferestrei, se creează fereastra și se procesează mesajele.

```
// Punctul de intrare al unei aplicatii Windows
int APIENTRY WinMain(    HINSTANCE      hInstance,
                         HINSTANCE      hPrevInstance,
                         LPSTR          lpCmdLine,
                         int            nCmdShow)
    {
    MSG         msg;         // structura de mesaje
    WNDCLASS    wc;          // structura de clasa de fereastra
    HWND        hWnd;        // handle pentru fereastra

    // inregistrarea stilului ferestrei
    wc.style         = CS_HREDRAW | CS_VREDRAW | CS_OWNDC;
    wc.lpfnWndProc   = (WNDPROC) WndProc;
    wc.cbClsExtra    = 0;
    wc.cbWndExtra    = 0;
    wc.hInstance     = hInstance;
    wc.hIcon         = NULL;
    wc.hCursor       = LoadCursor(NULL, IDC_ARROW);

    // nu este necesar un brush pentru fundal la OpenGL
    wc.hbrBackground  = NULL;

    wc.lpszMenuName   = NULL;
    wc.lpszClassName  = lpszAppName;

    // inregistrarea clasei de fereastra
    if(RegisterClass(&wc) == 0)
        return FALSE;


    // crearea ferestrei principale a aplicatiei
    hWnd = CreateWindow(
                lpszAppName,
                lpszAppName,

                //    OpenGL    requires    WS_CLIPCHILDREN    and
WS_CLIPSIBLINGS
                WS_OVERLAPPEDWINDOW     |     WS_CLIPCHILDREN     |
S_CLIPSIBLINGS,

                // pozitia si dimensiunea ferestrei
                100, 100,
                250, 250,
                NULL,
                NULL,
                hInstance,
                NULL);

    // daca fereastra nu a fost create, aplicatia se paraseste
    if(hWnd == NULL)
        return FALSE;


    // afisarea ferestrei
    ShowWindow(hWnd,SW_SHOW);
```

```
    UpdateWindow(hWnd);

    // procesarea mesajelor aplicatiei pana la inchiderea acesteia
    while( GetMessage(&msg, NULL, 0, 0))
        {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
        }

    return msg.wParam;
    }
```

- Pentru a preveni desenarea OpenGL în afara ferestrei, aceasta trebuie creată cu stilurile **WS_CLIPCHILDREN** și **WS_CLIPSIBLINGS**. De asemenea, clasa ferestrei trebuie să fie de tip **CS_OWNDC** pentru a se păstra setările de context.

Trebuie remarcat faptul că, spre deosebire de desenarea GDI unde funcțiile grafice modifică conținutul curent al ferestrei de vizualizare, în cazul desenării OpenGL conținutul ferestrei este actualizat doar în momentul apelului funcției **glRender()** printr-o redesenare completă. Prin urmare, în vizualizarea OpenGL actualizarea parțială a conținutului ferestrei este imposibilă. O soluție originală pentru rezolvarea acestei probleme este prezentată în cadrul aplicației din capitolul 5.

4.4.2. Selecția şi feedback

Selecția interactivă a obiectelor, inclusiv, reactia *(feedback)*, reprezintă o parte importantă a aplicațiilor de modelare. OpenGL oferă o serie de mecanisme ce pot fi utilizate la selecția și scoaterea în evidență *(highlighting)* a obiectelor.

Selecția

OpenGL oferă un mecanism de selecție a obiectelor în care geometria lor este transformată şi comparată cu o sub-regiune de selecție a *viewport* [Shr2004]. Mecanismul utilizează rețeaua de transformări pentru a compara vertecşii obiectului cu volumul vizibil. Pentru a reduce volumul vizibil la o sub-regiune a spațiului ecranului (în coordonate de fereastră) din *viewport*, coordonatele proiectate ale obiectului sunt transformate prin scalare şi translatare rezultând matricea [Mcr2005]:

$$T = \begin{pmatrix} \frac{P_x}{d_x} & 0 & 0 & p_x - 2\frac{q_x - o}{d_x} \\ 0 & \frac{P_{yx}}{d_y} & 0 & p_y - 2\frac{q_y - o_y}{d_y} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Unde o_x, o_y reprezintă originea pe axele x şi y; p_x şi p_y reprezintă lățimea şi respectiv înălțimea *viewport*-ului, iar q_x, q_y, d_x şi d_y reprezintă originea, lățimea şi respectiv înălțimea regiunii de selecție.

Obiectele sunt identificate prin atribuirea de nume cu ajutorul funcției **`glLoadName`**. Fiecare obiect este trimis către OpenGL unde este testat față de regiunea de selecție. Dacă testul este reuşit se va crea un *hit record* pentru identificarea obiectului[Shr2003] [Shr2004].

Modul de selecție a OpenGL indică faptul că un obiect a fost atins atunci când el intersectează volumul vizibil.

In multe cazuri se utilizează mai multe instanțieri ale datelor geometrice pentru reducerea utilizării memoriei. Instanțierea permite unei aplicații să creeze o singură reprezentare a unor date geometrice pentru fiecare tip de obiect utilizat într-o scenă. Dacă geometriei modelului i se asociază un singur nume, aplicația nu poate determina care din instante a fost selectată [Ast2004] [Lam2003]. Soluția la această problemă o reprezintă utilizarea unei stive de nume. Acest lucru permite aplicației, care reprezintă ierarhic modelele, să asocieze un nume fiecărui nivel ierarhic. Pe măsură ce scena este desenată, la parcurgerea ierarhiei, numele sunt puse şi luate de pe stivă. La crearea unui *hit record* stiva conține toate numele ce se află în stiva de nume. Aplicația determină care instanță a unui obiect a fost selectată prin analiza conținutului stivei de nume şi compararea cu numele păstrate în reprezentarea ierarhică a modelului. Figura 4-18a prezintă un cadru de automobil cu cele 4 roți desenate prin instanțierea aceluiaşi model de roată iar figura 4-18b prezintă ierarhia parțială a modelului.

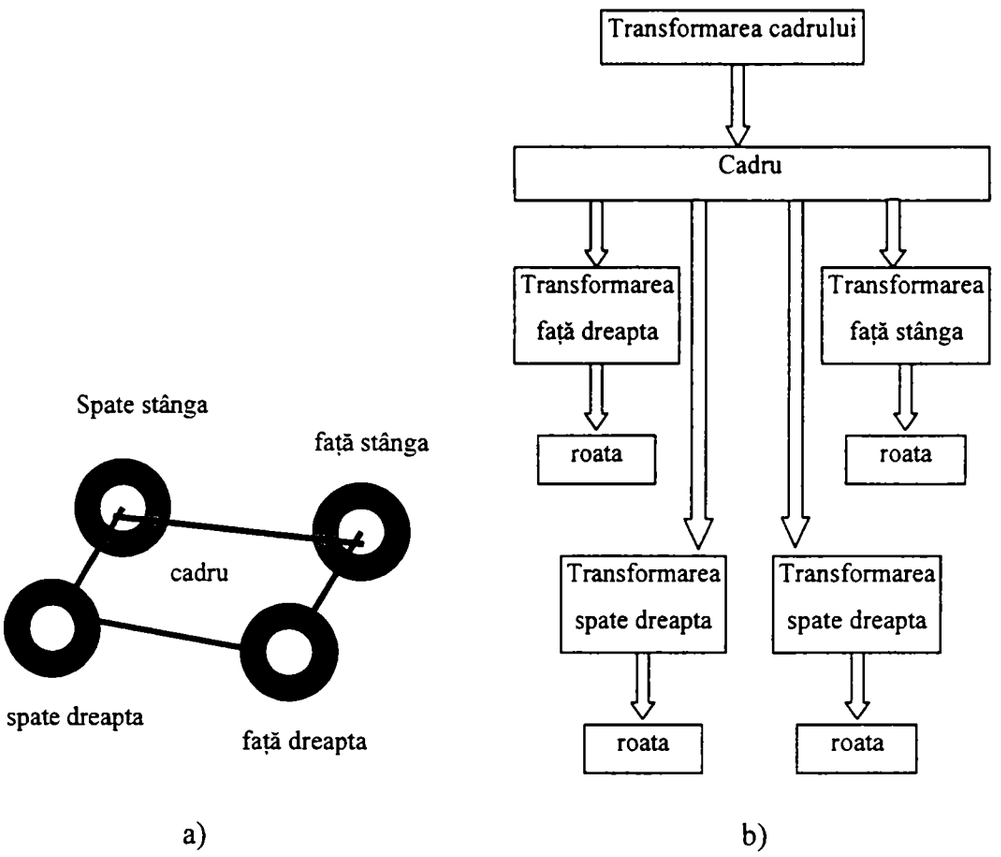


Figura 4-18. Cadru de automobil cu cele 4 roți desenate prin instanțierea aceluiaşi model de roată

Având în vedere că doar rezultatul transformărilor coordonatelor vertecşilor este important, nu este necesară transmiterea coordonatelor de textură, a normalelor, a culorilor sau a activării iluminării [Mcr2005] [Fin2004].

In cazul selecției se poate recurge la utilizarea unei forme simplificate a geometriei. Astfel, de exemplu, obiectele individuale pot fi înlocuite cu *bounding-box*-ul lor, mărind viteza în detrimentul preciziei. Pentru a mări precizia se poate adăuga încă un pas în care obiectele selectate prin geometria lor simplificată sunt reprocesate utilizând geometria adevărată [Han2005]. Acest algoritm în doi paşi îmbunătățeşte performanța dacă complexitatea combinată a celor doi paşi este mai mică decât complexitatea cazului în care se testează geometria reală în cadrul algoritmului cu un singur pas.

Există situații în care se doreşte identificarea unui punct $(x_o, y_o, z_o)^T$ din spațiul obiectului corespunzător unui punct $(x_w, y_w, z_w)^T$ din fereastră. In cazul proiecției ortogonale, coordonatele obiectului sunt calculate prin transformarea coordonatelor de fereastră, utilizând

inversa matricei *viewport*-ului V, a matricei transformării de proiecție V şi a matricei transformării de model M:

$$\begin{pmatrix} x_o \\ y_o \\ z_o \\ w_o \end{pmatrix} = M^{-1}P^{-1}V^{-1}\begin{pmatrix} x_w \\ y_w \\ z_w \\ w_w \end{pmatrix} = (PM)^{-1}V^{-1}\begin{pmatrix} x_w \\ y_w \\ z_w \\ 1 \end{pmatrix}$$

Există trei moduri de operare ale OpenGL: randare, selectare şi reactie [Seg2004]. In mod obişnuit la selecție de utilizează aceeaşi funcție de randare a scenei ca pentru desenare.

Buffer-ul de selecție este o matrice de întregi fără semn iar fiecare înregistrare are cel puțin 4 elemente. Primul index conține numărul de nume ce se aflau pe stiva de nume în momentul generării *hit record*-ului, următoarele două poziții conțin valorile minime şi maxime pentru coordonata z a tuturor vertecşilor (aceste valori sunt în domeniul [0,1] şi sunt scalate pe [0,MAX_UNSIGNED_INT]), ultima pozitie fiind baza stivei de nume. Dacă pe stivă apar mai multe nume (indicat de elementul de pe prima poziție), acestea urmează după elementul de pe poziția 4. Acest model este repetat pentru toate *hit-record*-urile generate.

In continuare se prezintă o diagramă de organizare a *buffer*-ului de selecție:

***Buffer* de selectie** **[0] - Numarul de nume pe stiva de nume in momentul aparitiei *hit*-ului = n_0**
[1] - valoarea minima pentru z
[2] - valoarea maxima pentru z
[n0+2] baza stivei de nume

...

Urmatorul *hit record* **[n_0+3] - numarul de nume pe stiva de nume pentru inregistrarea curenta = n_1**
[n_0+4] - valoarea minima pentru z

Formatul *buffer*-ului de selecție nu oferă nici un indiciu asupra numărului de *hit record*-uri. Bufferul de selecție este completat doar la trecerea OpenGL în modul **GL_RENDER**. Valoarea returnată de funcția **glRenderMode** reprezintă numărul de *hit record*-uri.

La selecția obiectelor cu ajutorul *mouse*-ului, se utilizează funcția ajutătoare **gluPickMatrix** ce creează matricea de descriere a noului volum de vedere:

```
void gluPickMatrix(GLdouble x, GLdouble y, GLdouble width, GLdouble
height, GLint viewport[4]);
```

x şi y reprezintă centrul volumului de vedere în coordonate de fereastră OpenGL. Acestea pot fi coordonatele cursorului de *mouse* pentru a centra volumul de vedere pe punctul de selecție. **Width** şi **height** reprezintă dimensiunile volumului de vedere în pixeli. Tabloul **viewport** conține coordonatele de fereastră ale *viewport*-ului curent. Acesta se poate obține prin apelul funcției **glGetIntegerv(GL_VIEWPORT, viewport)**. Trebuie remarcat faptul că pe verticală coordonatele ferestrei OpenGL sunt inversate.

Ca exemplu, funcția **gluPickMatrix** se apelează după cum urmează:

```
gluPickMatrix(mouse.xPos, viewport[3] - mouse.yPos, 2,2, viewport);
```

Reacția (Feedback)

Reacția (feedback-ul), ca şi selecția, reprezintă un mod de randare ce nu generează pixeli pe ecran, informația fiind scrisă într-un *feedback buffer* care indică modul în care scena este desenată pe ecran. Această informație include transformările vertecşilor în coordonate de fereastră, valorile pentru culoare rezultate din calculele de iluminare şi informații despre texturi [Shr2003] [Ast2004]. Practic informația obținută este suficientă pentru a rasteriza primitivele.

Rezultatul reactiei este utilizat pentru evidentierea obiectelor selectate.

Buffer-ul de feedback este un tablou de valori în virgulă mobilă specificat prin funcția **glFeedback**:

```
void glFeedbackBuffer(GLsizei size, GLenum type, GLfloat *buffer);
```

Această funcție primeşte dimensiunea *buffer*-ului de *feedback*, tipul şi cantitatea de informație dorită precum şi un *pointer* la *buffer*-ul propriu-zis.

Valorile valide pentru **type** sunt următoarele [Shr2004]:

Date de culoare	Date de textura	Nr. total de valori	tip	coordonate

GL_2D	x, y	-	-	2
GL_3D	x, y, z	-	-	3
GL_3D_COLOR	x, y, z	C	-	3 + C
GL_3D_COLOR_TEXTURE	x, y, z	C	4	7 + C
GL_4D_COLOR_TEXTURE	x, y, z, w	C	4	8 + C

Buffer-ul de *feedback* conține o listă de identificatori urmați de date de vertex și ntual date de culoare și textură. Acești identificatori trebuie analizați pentru a determina ul primitivelor desenate. Identificatorii pot fi următorii [Shr2004]:

lentificator	Primitivă
L_POINT_TOKEN	Puncte
GL_LINE_TOKEN	Linie
GL_LINE_RESET_TOKEN	Segment de linie cu resetarea stipple
GL_POLYGON_TOKEN	Poligon
GL_BITMAM_PTOKEN	Bitmap
GL_DRAW_PIXEL_TOKEN	Dreptunghi de pixel desenat
GL_COPY_PIXEL_TOKEN	Dreptunghi de pixel copiat
GL_PASS_THROUGH_TOKEN	Identificator definit de utilizator

Identificatorii **GL_PASS_THROUGH_TOKEN** se pot utiliza pentru a marca elemente e geometrie cu scopul de a fi identificate mai ușor în momentul analizei *feedback buffer*-lui.

Un exemplu de conținut de *feedback buffer* de tip **GL_3D** este prezentat în continuare:

```
eedback buffer      [0] - GL_POINT_TOKEN
                    [1] - coordonata x
                    [2] - coordonata y
                    [3] - coordonata z
                    [4] - GL_PASS_THROUGH
                    [5] - valoare definita de programator
                    [6] - GL_POLYGON_TOKEN
                    [7] - numar de vertecsi
                    [8] - coordonata x a primului vertex
                    [9] - coordonata y a primului vertex
```

```
[10] - coordonata z a primului vertex
[11] - coordonata x pt. al 2-lea vertex


[n] - coordonata z a ultimului vertex
```

4.4.3. Desenarea *wireframe* şi cu linii ascunse [Sav2005]

Una din modalitățile de reprezentare a modelelor este cea de desenare a muchiilor utilizând linii, ceea ce face posibila evidențierea detaliilor care altfel ar fi fost ascunse. Reprezentarea prin linii permite o evaluare vizuală asupra complexității geometriei modelului. Prin desenarea tuturor muchiilor unui obiect, utilizatorul poate evalua numărul de poligoane din care este compus modelul. Există mai multe variante de desenare a muchiilor printre care cele mai utilizate sunt reprezentarea *wireframe* şi cea cu linii ascunse [Ast2004] [Lam2003] [Han2005] [Lun2003].

Desenarea wireframe

In continuare se vor prezenta câțiva **algoritmi de desenare** *wireframe*:

1. Modelul se desenează prin poligoane în modul linie utilizând **`glBegin(GL_PLYGON)`** şi **`glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)`**.

   Acest algoritm este cel mai simplu atunci când aplicația poate desena modelul ca pe un solid.

   In comparație cu alti algoritmi, algoritmul poate fi mai lent pentru că desenarea de poligoane necesită o procesare mai laborioasă decât desenarea liniilor şi pentru că muchiile comune sunt desenate de două ori. Această dublare poate fi evitată prin gestionarea specială a fanionului de muchie vizibilă;
2. Desenarea poligoanelor se face prin utilizarea seriilor de linii, **`glBegin(GL_LINE_LOOP)`**.

   Aceast algoritm este aproape la fel de simplă ca şi cel anterior, necesitând o modificare doar înainte de apelul **`glBegin`**. Deşi se elimină astfel procesarea adițională necesară desenării de poligoane, toate dezavantajele metodei precedente se păstrează;

3. Sunt extrase muchiile modelului, iar apoi se desenează ca linii independente utilizând **`glBegin(GL_LINE)`**.

   Desi algoritmul presupune mai mult efort de programanre decât precedenții deoarece necesită identificarea muchiilor şi eliminarea duplicatelor, totuşi procesarea adițională se face o singură dată rezultând o desenare mai rapidă.

4. Muchiile modelului sunt extrase şi concatenate în serii de linii cât mai lungi şi apoi desenate cu **`glBegin(GL_LINE_STRIP)`**.

   Algoritmul are avantajul utilizării reduse a memoriei pentru păstrarea informației şi utilizează cea mai eficient algoritm de desenare a liniilor.

La alegerea unui anumit algoritm trebuie tinut cont de importanta alegerii primitivelor poligonale utilizate în model. Triunghiurile, *quad*-urile şi poligoanele independente se pot utiliza cu fanionul de muchie vizibilă, spre deosebire de seriile de triunghiuri, de evantaiele de triunghiuri şi de seriile de *quad*-uri

Deoarece primitivele conectate sunt în general mai eficiente decât cele independente, acestea sunt preferate de fiecare dată când este posibil.

Linii ascunse

In cadrul acestui algoritm obiectele sunt desenate în *wireframe,* în aşa fel încât liniile corespunzătoare muchiilor ascunse sunt omise sau se desenează într-un alt stil decât cele vizibile. Această tehnică poate clarifica desenele reprezentând obiecte complexe prin îmbunătățirea vizibilității lor [Mcr2005] [Her1995] [Att1989] .

Algoritmul propus in continuare considera că obiectul poate fi compus din poligoane. In primul pas se desenează obiectul prin poligoane iar în pasul doi se desenează laturile poligoanelor prin linii. La primul pas se actualizează doar *buffer*-ul de adâncime, iar la pasului doi, *buffer*-ul de adâncime permite numai desenarea liniilor vizibile.

Cei doi pasi conduc la urmatoarea detaliere de operatii:

Pasul I

1. Se dezactivează scrierea în *buffer*-ul de culoare cu **`glColorMask`**;
2. Se setează funcția de adâncime cu **`GL_EQUAL`**;
3. Se activează testul de adâncime cu **`glEnable(GL_DEPTH_TEST)`**;
4. Se desenează obiectele prin poligoane;

Pasul II

5. Se activează scrierea în *buffer*-ul de culoare;
6. Se desenează muchiile obiectului folosind una din metodele de desenare a muchiilor (*wireframe*).

Avand în vedere faptul că pixelii de pe muchiile primitivelor desenate prin poligoane şi pixelii de pe muchiile desenate prin linii au valori ale adâncimii numeric apropiate, pot apărea „artefacte" de desenare în urma erorilor de cuantificare [Mcr2005] [Seg2004]. Acest fenomen se manifestă prin omiterea anumitor pixeli de-a lungul liniilor, în locurile unde adâncimea unui pixel al liniei este mai mare decât pixelul de pe muchia poligonului. Utilizarea fanionului **`GL_EQUAL`** rezolvă parțial acest neajuns dar pentru a obține cele mai bune rezultate trebuie ca liniile să fie deplasate față de poligon utilizând **`glPoligonOffset`** sau **`glDeptRange`**.

In cazul în care se doreşte ca liniile ascunse să fie vizibile dar în acelaşi timp să fie desenate într-un stil diferit de cele vizibile, algoritmul se modifică în felul următor:
Pasul I.

1. Funcția de adâncime se alege **`GL_EQUAL`**;
2. Scrierea în *buffer*-ul de culoare nu este blocată;
3. Se setează culoarea sau modelul pentru liniile ascunse;
4. Se desenează obiectele ca şi muchii;

Pasul II.

5. Se dezactivează scrierea în *buffer*-ul de culoare;
6. Se desenează obiectele prin poligoane;

Pasul III.

7. Se setează culoarea şi/sau modelul pentru liniile vizibile;
8. Se desenează obiectele prin muchii utilizând una din metodele de desenare a muchiilor (*wireframe*).

In cadrul acestei tehnici, rezultatul dorit se obține în trei paşi. In primul pas se desenează toate muchiile cu linii punctate, în cel de-al doilea pas desenarea se face prin poligoane care actualizează *buffer*-ul de adâncime prevenind ştergerea liniilor ascunse la

desenarea din pasul anterior, ca în final să se deseneze pentru a doua oară muchiile obiectelor în linie continuă.

4.4.4. Descompunerea prin triunghiularizare

Triunghiularizarea reprezintă procesul de descompunere a unei suprafețe complexe în primitive mai simple cum sunt triunghiurile, fiind o operație necesară având în vedere că implementările OpenGL sunt optimizate pentru procesarea eficientă a triunghiurilor [Seg2004] [Shr2003] [Ast2004].

Unul din avantajele descompunerii în triunghiuri constă în faptul că această operație se efectuează la inițializarea aplicației. Un alt avantaj este dat de independența față de implementarea OpenGL. OpenGL nu specifică o anumită metodă de triunghiularizare astfel că fiecare implementare poate utiliza un algoritm propriu [Seg2004]. Majoritatea implementărilor oferă algoritmi simpli de triunghiularizare: poligoanele sunt transformate în evantaie de triunghiuri iar cuadrilaterele sunt împărțite în două triunghiuri.

Pentru ca desenarea OpenGL să fie cât mai rapidă, toate primitvele trebuie să fie convexe [Lam2003]. Prin urmare, poligoanele concave şi cele complexe (cu găuri) trebuie descompuse în triunghiuri. Biblioteca de utilități a OpenGL conține funcții de spargere a poligoanelor concave sau complexe în primitive mai mici şi mai simple.

Triunghiularizarea funcționează prin intermediul unui obiect de triunhgiularizare ce trebuie creat şi distrus de către programator [Shr2003]:

```
GLUtesselator *pTess;
pTess = gluNewTes();
. . .
// Efectuarea triunghiularizarii
. . .
gluDeleteDess(pTess);
```

Toate funcțiile de triunghiularizare utilizează un obiect de triunghiularizare ca şi prim parametru. Acest lucru permite efectuarea a mai multor operații de triunghiularizare în paralel. **Algoritmul** parcurs pentru acest caz este următorul:

1. Crearea obiectului de triunghiularizare;
2. Setarea stării şi a funcțiilor *callback* pentru obiectul de triunghiularizare
3. Inceperea unui poligon;
4. Inceperea unui contur;
5. Alimentarea obiectului de triunghiularizare cu vertecşii ce specifică conturul;

6. Terminarea conturului;
7. Reluare de la pasul 4 dacă există mai multe conturi;
8. Terminarea poligonului.

In timpul triunghiularizării se apelează o serie de funcții *callback* ce trebuie definite de dezvoltator. Aceste funcții se utilizează pentru a specifica informația de vertecşi şi pentru a începe şi termina primitivele. Pentru înregistrarea funcțiilor *callback* se utilizează funcția [Shr2003]:

```
void gluTessCallback(GLUTesselator *tobj, GLenum which, void
(*fn)());
```

Primul parametru este obiectul de triunghiularizare, al doilea specifică tipul funcției de *callback* ce va fi înregistrată respectiv al treilea parametru este un pointer la funcția de *callback* propriuzisă.

Ca exemplu se va prezenta inregistrarea unui set minim de funcții *callback*:

```
// Apelul glBegin la inceputul unui calup de triunghiuri
gluTessCallback(pTess, GLU_TESS_BEGIN, glBegin);

// Apelul glEnd la sfarsitul calupului de triunghiuri
gluTessCallback(pTess, GLU_TESS_END, glEnd);

// Apelul glVertex3dv pentru fiecare vertex
gluTessCallback(pTess, GLU_TESS_VERTEX, glVertex3dv);
```

Funcția *callback* GLU_TESS_BEGIN specifică funcția ce va fi apelată la începutul fiecărei noi primitive. Dezvoltatorul poate utiliza o funcție proprie ce poate efectua operații adiționale necesare la începerea unei noi primitive cum ar fi numărul de triunghiuri din cadrul poligonului triunghiularizat.

Funcția *callback* GLU_TESS_END, indică terminarea triunghiularizării unui poligon. Apelul GLU_TESS_VERTEX utilizează funcția glVertex3dv pentru a specifica vertecşii.

Procesul de triunghiularizare a unui poligon începe cu apelul funcției **gluTessBeginPolygon** (pasul 3) şi se încheie cu **gluTessEndPolygon** (pasul 8). Contururile poligoanelor se specifică (pasul 5) prin **gluTessVertex** între apelurile funcțiilor **gluTessBeginContour** (pasul 4) şi **gluTessEndContour** (pasul 6).

4.5. Criterii de evaluare a sistemelor CAD [Sav2002]

Având în vedere rata ridicată de învechire a tehnologiilor hardware şi software, devine necesară o creştere a complexității sistemelor de proiectare asistate de calculator. Dată fiind creşterea costurilor de producție, cerințele unui asemenea sistem depind de utilizabilitate, fiabilitate şi accesibilitate.

Un sistem interactiv de proiectare trebuie să fie orientat către utilizator. Acest lucru presupune generarea, calcularea şi afişarea informației sub o formă prietenoasă, uşor de înțeles de către utilizator.

Criteriile de bază utilizate în evaluarea unui sistem CAD trebuie alese din mai multe puncte de vedere, cum ar fi: cerințele utilizatorului, posibilitățile hardware/software ale dezvoltatorului etc.

Cele mai importante cerințe de sistem sunt acelea de a oferi funcțiile cerute de utilizator, de a fi fiabil şi de a acomoda utilizatorii din cadrul companiei. Sistemul trebuie să utilizeze tehnologii hardware si software relativ recente.

Modelatorul de produs din cadrul sistemului trebuie să fie capabil să modeleze tipurile de produse solicitate de utilizator. Acest lucru pare evident dar uneori pot exista conflicte de interese. Ca exemplu, pe de o parte, mulți oameni care foloseau hârtia ca mediu primar de definire a geometriei vor dori să utilizeze tehnici similare (adică un sistem 2D). Pe de altă parte, majoritatea produselor companiilor din domeniul ingineriei mecanice sunt tridimensionale, iar în viitor vor fi favorizate modelatoarele 3D pentru că acestea pot gestiona mai multe informații de produs.

Acest paragraf va prezenta un set de criterii de evaluare a unui sistem CAD, cu scopul de a stabili sistemul corespunzător scopului utilizatorului.

Au fost luate în considerare următoarele **criterii**:

- alegerea între proiectarea 2D şi 3D;
- viteza de învățare;
- viteza;
- informația grafică;
- schimbul de date;
- costurile de utilizare.

Aceste criterii nu sunt independente între ele. Gradul de importanță al fiecărui criteriu variază de la caz la caz. Ca aplicație a criteriilor menționate anterior se va considera cazul unui sistem CAD pentru construcții din lemn.

Cel mai important criteriu din cadrul dezvoltării unui sistem CAD pentru construcții din lemn este îndeplinirea cerințelor clientului [Phi2002]:

- alegerea unui sistem CAD 3D pentru o reprezentare mai bună şi mai precisă a elementelor proiectate;
- viteza de învățare: minimizarea vitezei de învățare datorită abilității restrânse a utilizatorilor de mânuire a calculatoarelor;
- viteza: reprezentarea 3D a datelor, generarea de planuri 2D, generarea de liste de materiale;
- informații grafice: aranjament optim a informațiilor afişate pe ecranul calculatorului, reprezentări fotorealiste a elementelor proiectate;
- schimbul de date: posibilitatea de import şi export din/către alte sisteme CAD, vizualizări în alte aplicații, generarea de reprezentări fotorealiste în aplicații specifice, sisteme de realitate virtuală;
- cost de utilizare: minimizarea achiziționării de aplicații adiționale prin includerea tuturor modulelor utilizate în cadrul vieții produsului.

Alegerea un sistem de proiectare 2D sau 3D

Datorită faptului că majoritatea desenelor aflate în circulație sunt bazate pe hârtie sau microfilm, tranziția de la hârtie la CAD 3D constă în cel puțin două faze care pot fi parcurse simultan:

1. trecerea de la hârtie la formate electronice;
2. conversia de la CAD 2D la 3D.

In cazul în care utilizatorul foloseşte un sistem CAD 2D, trecerea la un sistem CAD 3D poate fi justificat prin viteza de învățare, costurile adiționale de utilizare sau caracteristicile noului sistem. Cu toate acestea, au apărut şi concepții greşite în alegerea unui sistem 3D. Majoritatea utilizatorilor CAD 2D invocă costurile mari şi dificultatea de învățare.

Cu timpul, o cantitate mare de informații din cadrul proiectului se modifică sau va fi utilizată în cadrul altor produse. In absența unei bune gestiuni a datelor (cum este la sistemele 2D) devine tot mai dificilă gestionarea cunoştințelor legate de starea datelor şi a relațiilor.

Viteza de învățare

Inițial, în cadrul dezvoltării sistemului CAD se pune accent pe disponibilitatea funcțiilor individuale de definire a produselor. Abia ulterior, atenția se îndreaptă spre abilitatea utilizatorilor de a învăța sistemul şi de a-l utiliza zilnic.

Orice sistem CAD, 2D sau 3D, are o anumită viteză de învățare. Tranziția cea mai grea este aceea de la desenarea manuală la CAD. In general, trecerea direct la proiectarea 3D este mai rapidă decât trecerea prin proiectarea 2D. Unul din motive poate fi efortul mai ridicat depus în proiectarea interfețelor sistemelor 3D față de sistemele 2D. Singura caracteristică avansată din cadrul sistemelor 2D o reprezintă cursorul inteligent. Software-ul 3D modern oferă o serie de instrumente de uşurare a muncii: structura arborescentă de construcție, instrumente de ajutor animate, clicuri şi gesturi de maus în funcție de context etc.

Atunci când CAD-ul 2D a devenit disponibil pe PC-uri, software-ul 3D era considerat de domeniul sistemelor UNIX. Utilizatorii care făceau trecerea de la 2D la 3D ar fi trebuit să învețe în plus şi un nou sistem de operare. In prezent, chiar şi cele mai avansate aplicații 3D rulează pe sisteme Windows. De aceea, unu aspect legat de viteza de învățare sunt sistemele de operare [Mal2000].

La începutul anilor 1990, compania SolidWorks a pionierat o interfață mai intuitivă şi mai prietenoasă cu utilizatorul. Sub presiunea crescândă de a fi mai puternice şi mai uşor de utilizat, sistemele CAD 3D actuale prezintă interfețe pentru care viteza de învățare este considerabil redusă. Chiar dacă şi CAD-ul 2D a beneficiat de acest avantaj, acesta este un subset de facilități, rezultate din cerințele pieții sistemelor 3D.

Viteza de utilizare

Dacă se iau în considerare limbajele de programare utilizate în cadrul CAD-ului 2D, este foarte greu de apreciat viteza de desenare a unui produs în 3D şi 2D. Cu toate că, în cadrul CAD-ului 3D, cotele pot fi transmise către desenul final totuşi majoritatea cotelor unui element mai trebuie adăugate la desen în cadrul etapei finale.

In cazul în care există piese şi ansambluri similare, majoritatea software-ului 3D prezintă posibilitatea de a face legături cu baze de date interne sau externe. Prin modificarea unor valori într-o bază de date se vor genera automat componente şi planuri noi, rezultând o creştere semnificativă a vitezei.

Un alt avantaj important al CAD-ului 3D este dat de uşurința editării. Atunci când este necesară o modificare într-un desen 2D, trebuie modificate toate vederile şi cotele. In CAD-ul 3D la modificarea unui model, direct sau prin baza de date, desenul sau desenele sunt actualizate automat.

Informația grafică

Uneori pachetele CAD 2D sunt prea flexibile, ceea ce poate fi un dezavantaj. Ele au abilitatea de a ‚stiliza' desenele într-o asemenea măsură încât acestea pot fi interpretate greşit. Acest lucru poate fi un avantaj pentru companiile cu producție proprie şi un dezavantaj pentru companiile cu producție externă. Una dintre situațiile ce pot apare este aceea în care pe aceeaşi foaie de hârtie sunt incluse mai multe vederi 2D, rezultând suprapuneri ce pot deforma percepția informației reale. CAD-ul 3D obligă utilizatorul să fie mai realist oferind o caracteristică simplă, foarte apreciată de departamentele de producție şi furnizorii externi: o vedere 3D izometrică pentru o mai bună înțelegere a desenelor 2D.

Schimbul de date

Pentru a maximiza valoarea datelor în cadrul unei organizații, acestea trebuie să fie multifuncționale.

Datele generate de CAD-ul 2D sunt utile pentru:

- activități în cadrul sistemelor numerice de conducere;
- publicarea pe internet (cum este formatul de fişier Autodesk DWF);
- conversia la alte formate grafice (de ex. Adobe PDF).

Fişierele 3D CAD pot fi folosite în mai multe situații pentru evaluarea proiectului sau reducerea timpului de producție [Aca2002]:

- analiza cu elemente finite: descoperirea slăbiciunilor unui produs înaintea clientului;
- analiza dinamică: descoperirea interacțiunii din cadrul ansamblurilor aflate în mişcare virtuală;
- analiza mulajelor: proiectarea de mulaje cu mai puține puncte ‚reci' şi ‚goluri';
- prototipizarea rapidă: realizarea de părți din plastic, de exmplu pentru teste de ergonomie, direct din modele CAD în doar câteva ore;
- partajarea modelelor 3D cu alți utilizatori: partajarea pentru analize externe sau colaborări;

- vizualizarea produsului: realizarea de imagini şi animaţii fotorealiste cu luni de zile înainte ca modelul să devină realitate;
- producţie CAD/CAM: utilizarea modelelor 3D pentru generarea datelor necesare maşinilor de prelucrare.

Un câştig major în productivitate rezultă din reutilizarea datelor de produs în diferite domenii de aplicaţii. Anumite aplicaţii accesează datele din CAD sau alte baze de date, altele transferă date din/în baza de date. Este foarte importantă existenţa Application Programming Interface (API-urilor) în cadrul unui sistem CAD pentru a permite interfaţarea cu alte sisteme [Sta2000].

Costurile de utilizare

Costul software-ului 3D, totusi diferenta care la inceput era de 100:1, in prezent nu este prea mare. Acest lucru permite utilizatorilor achiziţionarea unui pachet complet de proiectare de solide/suprafeţe la un preţ rezonabil.

Utilizarea celui mai bun sistem CAD disponibil nu este o necesitate. O importanţă mai mare trebuie dată existenţei funcţionalităţilor necesare utilizatorului. Atunci când este posibil, companiile trebuie să evite dezvoltarea unui sistem CAD propriu. Majoritatea funcţiilor CAD necesare companiilor mici şi medii sunt deja disponibile în cadrul unor sisteme ce pot fi achiziţionate. Efortul necesar dezvoltării şi întreţinerii unui sistem CAD este enorm, iar companiile utilizatoare trebuie să-şi concentreze resursele asupra utilizării CAD pentru îmbunătăţirea produselor şi nu pe dezvoltarea de software CAD.

4.5.1. Studiu de caz: dezvoltarea unui sistem CAD de construcţii în lemn

Cel mai important criteriu în cadrul dezvoltării unui sistem CAD de construcţii din lemn este satisfacerea cerinţelor utilizatorului.

Pentru o reprezentare cât mai clară şi precisă a elementelor proiectate s-a ales un **sistem CAD 3D**. Figura 4-19 prezintă o reprezentare 3D a unui element de construcţie, a cărui formă utilizatorul o poate înţelege fără dificultate. In cadrul unui sistem 2D ar fi fost necesare mai multe vederi. De asemenea, utilizatorul mai trebuie să depună un efort mintal adiţional de combinare a vederilor 2D pentru a-şi imagina obiectul 3D.

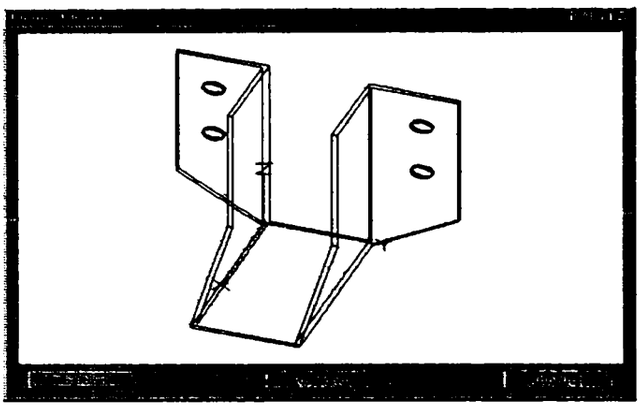


Figura 4-19. Reprezentare 3D a unui element de construcție

Viteza de învățare. Datorită cunoştinţelor reduse de utilizare a calculatoarelor s-a minimizat viteza de învățare. O atenție sporită s-a acordat dezvoltării unei interfețe grafice cu utilizatorul prietenoase, corespunzătoare condițiilor mediului de utilizare. Utilizatorii finali sunt de pregătire tâmplari, cu puține cunoştinţe în utilizarea calculatoarelor. Ca exemplu, fiecare comandă din cadrul meniului are atribuit un număr (Figura 4-20). Pentru a accesa rapid o comandă (Bearbeiten->Kopieren->Graphisch) fără navigarea meniului, utilizatorul trebuie să apese numerele corespunzătoare căii din meniu. Pentru a repeta rapid ultima comandă executată se utilizează tasta ‚+' din cadrul blocului numeric.

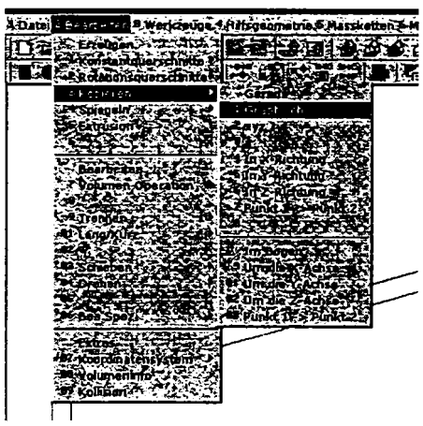


Figura 4-20. Apelul rapid al unui punct de meniu

Viteza. S-a crescut viteza de executare a comenzilor, de reprezentare 3D, de generare de planuri 2D, de generare de liste optimizate de materiale. Viteza reprezintă unul din cei mai importanți factori în generarea unei reprezentări 3D a elementelor de proiectare, în special rotirea sau deplasarea. De asemenea viteza este importantă în generarea listelor de materiale. De exemplu, în cazul barelor din acoperiş este efectuată o sortare în funcție de lungime, care poate reduce semnificativ costurile de construcție.

Informația grafică. S-a acordat atenție următoarelor aspecte: amplasarea optimă pe ecran a informației, informația ajutătoare, reprezentarea fotorealistă a elementelor proiectate. Aproape toate cutiile de dialog prezintă un desen explicativ pentru fiecare parametru modificabil de către utilizator, după cum este prezentat și în figura 4-21.

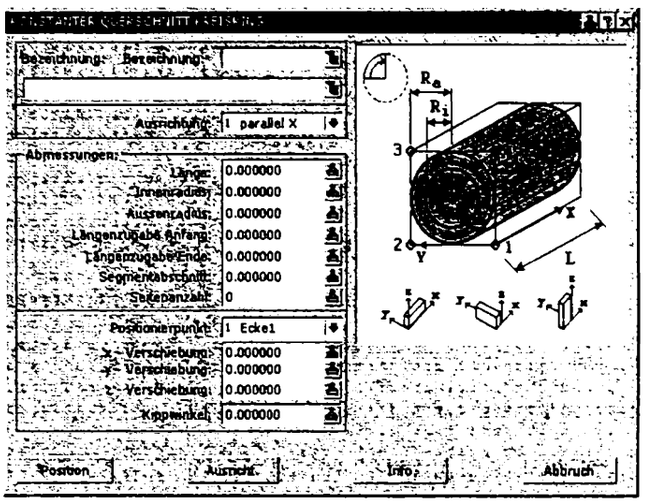


Figura 4-21. Cutie de dialog cu imagine explicativă

Două imagini fotorealiste reprezentând vederi din timpul unei rotiri a unui pod de lemn au fost salvate și sunt prezentate în figura 4-22.

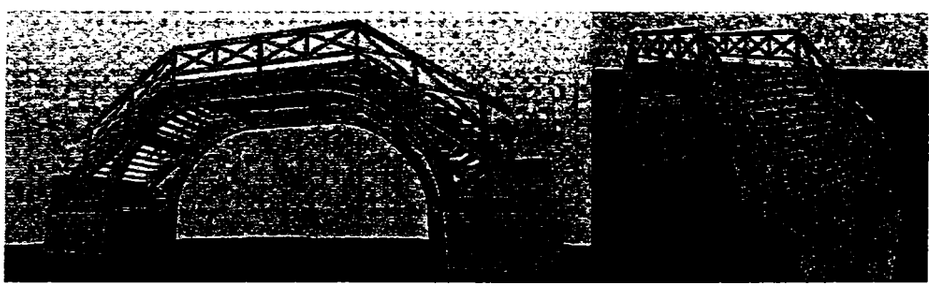


Figura 4-22. Reprezentări fotorealiste a unui pod de lemn

Schimbul de date: s-a avut în vedere posibilitatea de export/import de date către/din alte aplicații CAD și sisteme de realitate virtuală, publicarea pe internet etc.

Costul de utilizare: s-a avut în vedere minimizarea necesității achiziționării de software adițional prin includerea de module utilizate pe tot parcursul vieții produsului.

4.6. Concluzii şi contribuții

Capitolul tratează structurat problematica realizării interfețelor cu utilizatorul prin analiza domeniului a trei tehnologii software actuale, care se constituie în sursele cele mai cuprinzătoare privind soluțiile tehnologice destinate realizării de interfețe cu utilizatorul de înaltă performanță.

In prima parte, s-a abordat tehnologia programării orientate spre obiecte justificându-se alegerea acesteia prin analiza sintetică a elementelor sale esențiale şi sublinierea avantajelor. S-au formulat etapele de dezvoltare ale unui produs software într-o manieră obiectuală, fiind analizate în detaliu în urma evaluării analitice a elementelor necesare proiectării. S-au sintetizat aspecte teoretice şi practice legate de implementarea software realizând o analiză critică şi evidențiind aspectele practice legate de utilizarea template-urilor, de procesul de prototipizare şi de implementarea unei componente reutilizabile.

In a doua parte, s-a analizat proiectarea interfețelor cu utilizatorul în contextul proiectării sistemelor software având ca scop elaborarea unor reguli şi repere de proiectare a interfețelor performante cu utilizatorul. S-au structurat, formulat şi analizat câteva din principalele domenii care au un rol în stabilirea gradului de utilizabilitate al unui sistem, s-au enunțat o serie de criterii pentru realizarea interfețelor grafice şi s-au elaborat un set de reguli de proiectare a elementelor de interfață personalizate – elemente ce stau la baza dezvoltării aplicațiilor propuse în capitolul următor.

A treia parte a capitolului se concentrează asupra utilizării bibliotecii grafice OpenGL în cadrul dezvoltării unei vizualizări realiste a modelelor proiectate de către utilizator. S-a realizat un studiu şi o analiză ale elementelor esențiale din cadrul programării OpenGL, elaborându-se câte un algoritm de implementare pentru vizualizarea în ferestre multiple, utilizarea modului de selecție şi reacție (feedback) pentru selecția şi evidențierea pe ecran a unei entități grafice, desenarea în mod wireframe şi cu linii ascunse pentru o mai bună percepție a reprezentării tridimensionale a modelului 3D proiectat şi respectiv triunghiularizarea poligoanelor complexe pentru ca acestea să fie reprezentate corect pe ecran utilizând biblioteca utilitară a OpenGL.

In final, sunt determinate criterii eficiente de evaluare pentru ca un sistem CAD să satisfacă toate cerințele utilizatorului, alegerea corespunzătoare depinzând de mai mulți factori printre care se numără domeniul industrial, formatul datelor utilizat de client, furnizori și alte divizii din cadrul organizației, aplicații adiționale necesare.

Contribuțiile autorului se regăsesc în cadrul tuturor subiectelor abordate și sunt evidențiate în detaliu în cadrul ultimului capitol al tezei.

5. Interfața pentru un sistem CAD integrat cu aplicație în proiectarea construcțiilor din lemn

5.1. Introducere

Capitolul curent, în totalitate original, prezintă soluțiile de aplicare şi implementare a metodelor şi recomandărilor de dezvoltare software propuse în capitolul anterior, în cadrul interfeței sistemului ***CAD Dietrich's de construcții din lemn*** realizată integral de către autor. In concret, sunt prezentate detaliat soluțiile de implemetare legate de:

- *dezvoltarea unei interfețe cu utilizatorul pentru Manager de Proiecte CAD*;
- *dezvoltarea unor elemente personalizate de interfață;*
- *dezvoltarea unui proces standard de selecție de obiecte;*
- *dezvoltarea unei gestiuni speciale pentru un grup de cutii de dialog;*
- *dezvoltarea unui modul de vizualizare 3D a modelelor prin utilizarea bibliotecii grafice OpenGL.*

Soluțiile propuse şi utilizate în dezvoltările din cadrul acestui capitol au fost publicate, în parte, de către autor în cadrul următoarelor lucrări:

- **Evaluation Criteria for Computer Aided Design Systems,,** publicată în buletinul ştiințific al Universității Politehnica din Timişoara, România, *Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE*, Vol. 47 (61), 2002 [Sav2002];
- **Principles for Dialog Box Design and Operation Within a CAD System for Wood Construction**, publicată în buletinul ştiințific al Universității Politehnica din Timişoara, România, *Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE* Vol.47 (61), 2002 [Sav2002b];
- **User Interface Optimizations For Beam Processing Sets**. Publicată în cadrul *The 14th International Conference On Control Systems And Computer Science, July 2-5, Bucharest*, 2003 [Sav2003].
- **Special Management of Dialog Boxes within a Timber Construction CAD System,** publicată în buletinul ştiințific al Universității Politehnica din Timişoara, România, *Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE* Vol.49 (63), 2004 [Sav2004];

- **Standard Selection of Objects within a Wood Construction CAD System**, publicată în cadrul *The 1st Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, Timisoara, Romania, May 25-26, 2004 pp.125-131 [SAV2004b]
- **Visualization Techniques for Models Using OpenGL**, publicată în buletinul ştiinţific al Universităţii Politehnica din Timişoara, România, *Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE*, Vol. 50 (64), 2005 [Sav2005];
- **Advanced Developments in Standard Selection of Objects within a Wood Construction CAD System**, publicată în buletinul ştiinţific al Universităţii Politehnica din Timişoara, România, *Transactions on AUTOMATIC CONTROL and COMPUTER SCIENCE*, Vol. 50 (64), 2005 [Sav2005b];
-

Tehnologiile performante prezentate în cadrul acestui capitol reprezintă rezultatul unor studii şi cercetări întreprinse de autor pe parcursul mai multor ani privind realizarea şi îmbunătăţirea utilizabilităţii interfeţei sistemului CAD Dietrich's, contribuind în mare măsură la creşterea gradului de satisfacţie a utilizatorului.

Sistemul CAD *Dietrich's*. Interfața cu utilizatorul.

Sistemul CAD *Dietrich's* se adresează utilizatorilor ce doresc proiectarea şi realizarea de construcţii din lemn, acoperindu-se toată gama de servicii, de la arhitectură şi proiectare, până la realizarea clădirilor, producând desene, liste de materiale şi funcţii de control al facilităţilor de producţie.

Principalele facilităţi oferite de sistemul CAD includ:

- programe asistenţi extensivi pentru procesele de construcţie a zidurilor, tavane/podele;
- modulul de generare a acoperişului;
- realizarea grafică de acoperişuri cu sau fără profile cu posibilitatea modificării ulterioare a unghiurilor;
- bază de date extinsă de lucarne, alături de un procesor inteligent pentru construirea lor;
- calcule rapide de proprietăţi masice;

- funcții speciale de construcție a cadrelor de lemn şi a componentelor de construcție;
- biblioteci de construcții cu macro funcții;
- vizualizări foto-realistice a elementelor proiectate;
- generarea de desene, planuri;
- transfer integrat al tuturor proceselor către post-procesor;
- interfațare cu programe CAD arhitecturale;
- liste de materiale şi semifabricate pentru ziduri şi acoperişuri, țigle, lemne;

Funcțiile de bază pentru tâmplărie sunt extinse cu un set semnificativ de facilități ce vin în ajutorul proiectantului mai ales în cazul reutilizării informațiilor de proiectare:

- se pot crea, salva şi re-apela construcțiile proprii în/dintr-o bază de date, inclusiv prelucrările de maşini;
- se pot defini liber structuri organizatorice de model (MOS);
- se pot utiliza funcții extinse de administrare a structurii, de management şi de organizare.

Interfața cu utilizatorul realizată de autor şi integrată de către sistemul Dietrich's este bazată pe cea a sistemului de operare Windows, la care au fost realizate modificări cu caracter original.

S-a recurs la aceste modificări pentru a eficientiza lucrul cu programul prin reducerea timpului de învățare al sistemului, precum şi accelerarea activităților de rutină ce sunt efectuate în cadrul programului. Un alt obiectiv urmărit a fost acela de a accesibiliza aplicația prin dezvoltarea unei interfețe adecvate majorității utilizatorilor acestui program care, în general, nu posedă cunoştințe avansate de operare a calculatorului. Astfel, majoritatea operațiilor ce se pot realiza cu acest program pot fi efectuate prin utilizarea mouse-ului şi a blocului numeric al tastaturii.

Interfața cu utilizatorul propusă a fost astfel concepută încât aproape toate operațiile pe care utilizatorul le poate efectua la un moment dat - în funcție de starea programului şi de elementul curent selectat - sunt disponibile direct pe ecran (figura 5-1). De asemenea, la fiecare utilizare a unei opțiuni din cadrul unui sub-meniu, în partea dreaptă a ferestrei principale este prezentată o copie a sub-meniului respectiv. S-a ales acest comportament datorită faptului că un sub-meniu grupează de obicei opțiuni din acelaşi domeniu, astfel că

este foarte probabil ca utilizatorul să acceseze imediat o altă opțiune din cadrul aceluiași sub-meniu.

5.2. Managerul de proiecte

In cadrul sistemului, pentru ca utilizatorul să gestioneze cât mai uşor şi eficient proiectele realizate, este necesară implementarea unui manager de proiecte, care se ocupă de toate operațiunile ce implică activități organizatorice legate de proiecte şi de poziții din cadrul proiectelor.

Activitățile stabilite ca utile utilizatorului sunt cele de creare a unui proiect nou, copiere, mutare, ştergere, modificare a informațiilor globale de proiect. În plus față de operațiunile clasice, mai este disponibilă operația de împachetare printr-un algoritm eficient de comprimare a fişierelor ce aparțin unui proiect sau unei poziții. Astfel, transmiterea informațiilor de proiect devine mult mai simplă.

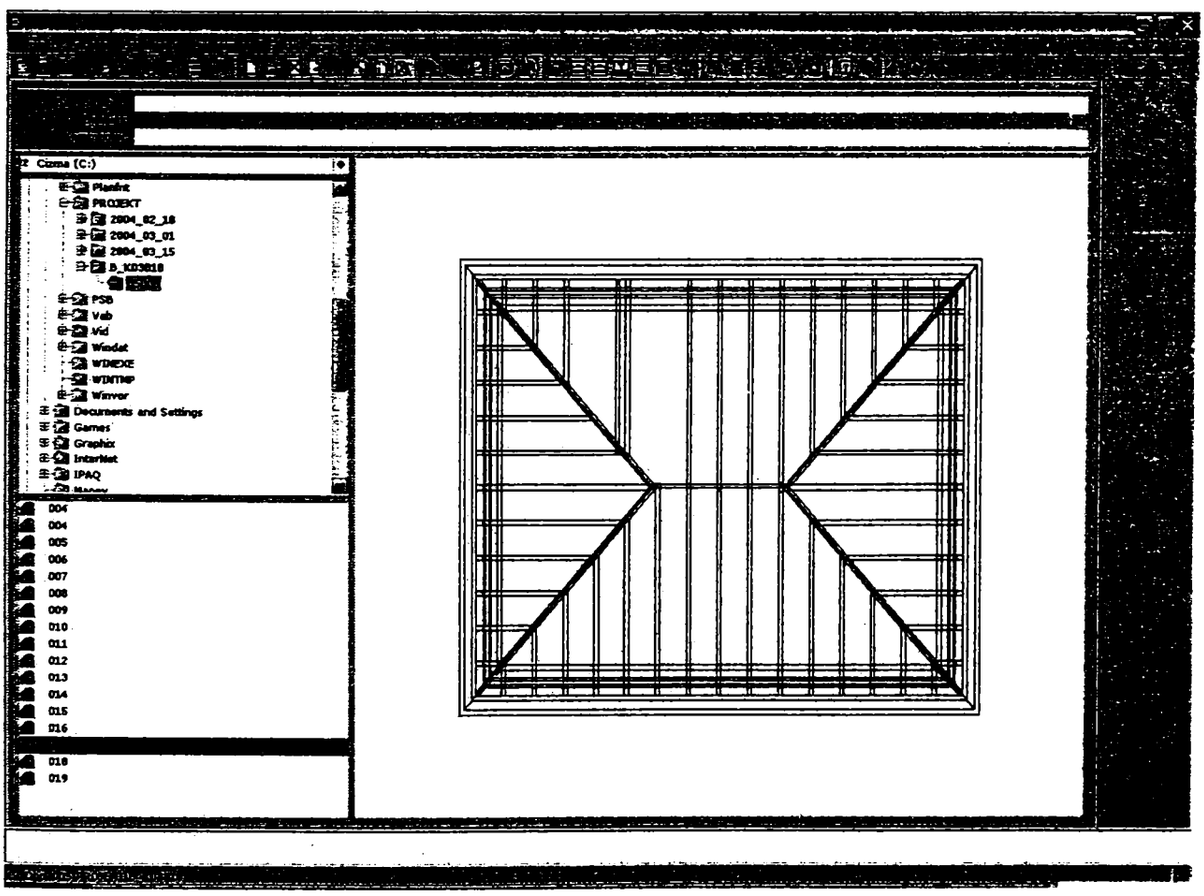


Figura 5-1. Interfața programului manager de proiecte

Informațiile de proiect cum ar fi numele proiectului, proprietarul şi eventual un scurt comentariu, se afişează în cadrul a trei linii de text imediat sub barele de instrumente. Aceste linii nu sunt editabile decât în cazul în care ele nu au fost încă completate.

Următoarele elemente din cadrul ferestrei s-au determinat ca fiind necesare:

- un combo-box pentru selectarea unității de disc;
- structură arborescentă pentru selectarea unui proiect. S-a ales acest tip de reprezentare a căii de acces la un proiect pentru ca utilizatorul să aibă o imagine clară asupra locației proiectului său;
- listă ce cuprinde toate tipurile de poziții existente în cadrul unui proiect;
- fereastră de previzualizare a informației conținute de poziția curentă selectată din listă.

Selecția proiectului. Datorită reprezentării vizuale intuitive a structurilor arborescente, selectarea unui proiect se face cu ajutorul unei astfel de structuri. Funcțiile de operare asupra proiectelor se integrează în cadrul meniului şi barelor de instrumente. În funcție de starea proiectului selectat este activ unul din butoanele de „împachetare" sau „despachetare".

Deoarece selecția simultană a proiectului şi a poziției active nu se poate realiza prin mecanismele proprii sistemului de operare, s-a realizat o rutină specială de tratare a acestei probleme. Totuşi, doar una din cele două elemente selectate reprezintă selecția principală şi reacționează la operarea tastaturii (ex. tastele de navigare).

Există trei icoane ce sunt utilizate şi la proiecte şi la poziții din cadrul proiectelor, astfel că ele se referă la selecția curentă:

- copierea în *clipboard*;
- inserarea din *clipboard*;
- ataşarea la un e-mail.

Toate pozițiile existente în cadrul proiectului curent selectat se organizează în cadrul **listei de poziții**. Pentru a face distincția vizuală între diferitele tipuri de poziții, fiecărui tip de poziție i s-a asociat o icoană specifică. Mai mult, alături de numele poziției apare şi un comentariu corespunzător.

Fereastra grafică de previzualizare a fost dezvoltată pentru previzualizarea informației conținută de poziția curentă (selectată), fiind activată sau dezactivată cu ajutorul unui buton special. Desenarea previzualizării unor poziții complexe durează mult timp, blocând procesarea mesajelor de sistem de către aplicație. Pentru a eficientiza procesul de selecție a pozițiilor, interfața a fost dezvoltată astfel încât desenarea să fie efectuată în paralel, într-un *thread* separat. În consecință, întreruperea desenării devine posibilă pentru a selecta o nouă poziție. Prin dezactivarea previzualizării, în această fereastră pot fi afișate alte imagini informative, în funcție de ora curentă.

5.2.1. Gestiunea unui proiect

In cadrul managerului de proiecte s-a creat sub-meniul *Projekt* (Figura 5-2) în care au fost grupate toate funcțiile de operare asupra unui proiect. Opțiunile determinate ca fiind necesare pentru gestionarea eficientă a proiectului sunt următoarele:

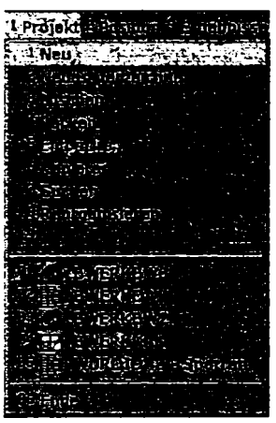


Figura 5-2. Submeniul de proiect

- **Proiect nou** (Figura 5-3) Pentru crearea unui nou proiect se deschide cutia de dialog corespunzătoare în care se completează numele proiectului din 5 caractere, proprietarul (Figura 5-4) şi un scurt comentariu sugestiv;
- **Director nou**;
- **Ştergerea** proiectului selectat;
- **Împachetarea** proiectului selectat;
- **Despachetarea** proiectului selectat;
- **Copierea** proiectului selectat;
- **Căutarea** proiectului selectat;

- **Reorganizarea** proiectului selectat, care se utilizează la repararea/reconstrucția unui proiect invalid.

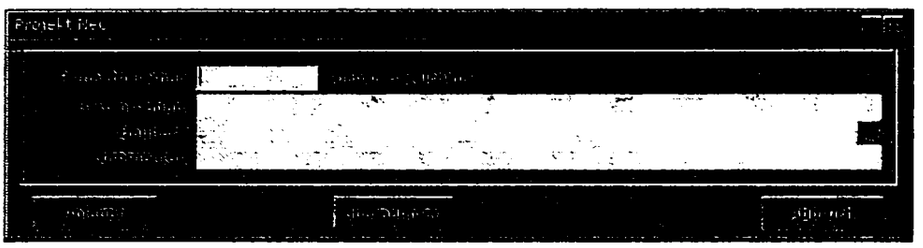


Figura 5-3. Crearea unui proiect nou

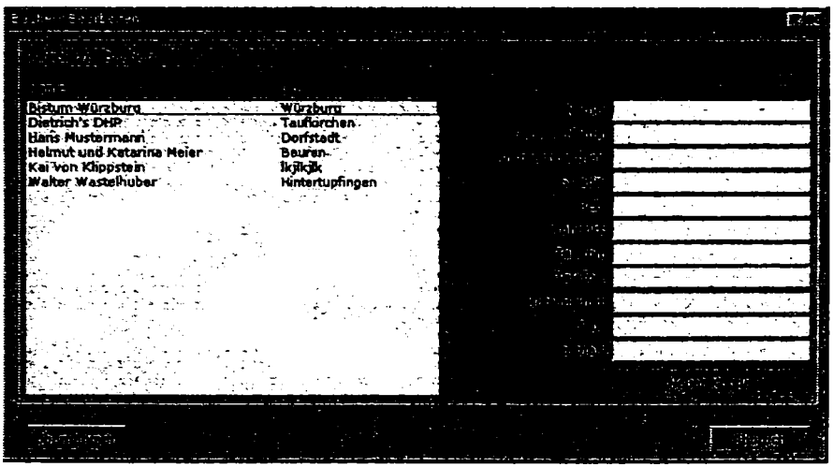


Figura 5-4. Completarea informațiilor despre proiectant

Datorită faptului că, de cele mai multe ori, utilizatorul dorește să-și continue activitatea în cadrul unui proiect, pentru suprimarea timpului de căutare a poziției dorite, înaintea punctului de meniu reprezentând opțiunea de părăsire a managerului de proiecte sunt listate ultimele cinci poziții apelate de către utilizator, figura 5-2.

5.2.2. Gestiunea unei poziții

Majoritatea opțiunilor determinate ca fiind necesare pentru o gestiune eficientă a pozițiilor din cadrul unui proiect sunt disponibile într-un meniu dedicat și au funcții asemănătoare cu cele corespondente din cadrul meniului de gestiune a proiectului:

- **Poziție nouă** (Figura 5-5). Pentru crearea unei poziții noi se deschide cutia de dialog corespunzătoare, în care utilizatorul indică tipul poziției, numărul (numele) și un scurt text informativ. În funcție de tipul de poziție selectat, conținutul unei liste se va modifica dinamic indicând pozițiile de același tip existente;

- **Prelucrare.** Acest punct de meniu se activează dacă este selectată o poziție validă și conduce la editarea ei (apelarea aplicației corespunzătoare);
- **Ștergere** - realizează ștergerea poziției curente;
- **Împachetare** – realizează un fișier comprimat pentru poziția curentă;
- **Despachetare** – despachetează poziția curentă dacă aceasta este împachetată;
- **Copiere** -- realizează copierea poziției curente;
- **Info** – deschide o cutie de dialog în care se poate edita o scurtă descriere semnificativă poziției curente.

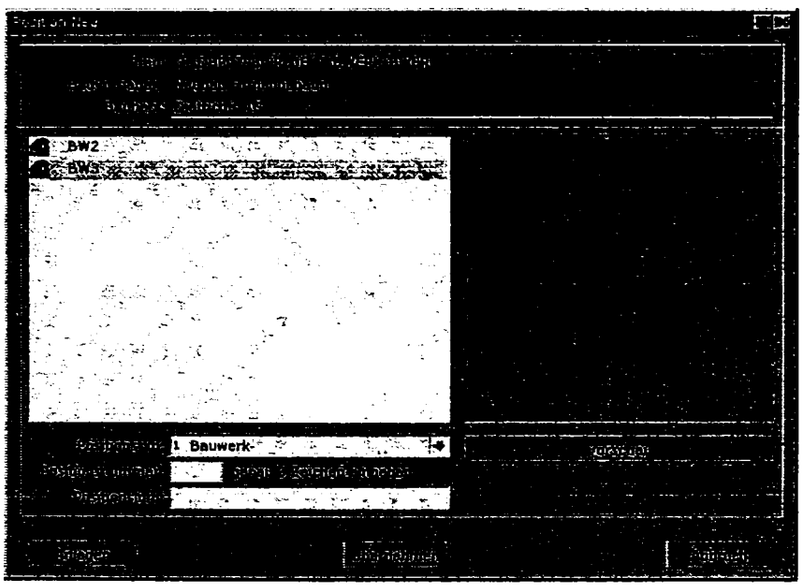


Figura 5-5. Crearea unei poziții noi

5.3. Elemente personalizate de interfață

Prin adaptarea elementelor standard de interfață (ale sistemului de operare Windows) în cadrul aplicației realizate s-au creat elementele personalizate de interfață care au rolul de a eficientiza activitatea de proiectare a utilizatorului, ele fiind parte integrantă a cutiilor de dialog.

5.3.1. Proiectarea cutiilor de dialog cu elemente speciale personalizate

Proiectarea cutiilor de dialog a implicat considerarea unui număr de factori precum *tipul informației, modurile de introducere a informației, elementele de interfață (standard*

sau personalizate) utilizate la afişarea informației, factori care facilitează introducerea sau afişarea informației.

S-a dorit o separare a procesului de proiectare a interfeței de procesul de integrare în cadrul aplicației astfel încât proiectantul cutiilor de dialog nu trebuie să posede cunoştințe avansate de programare. El trebuie să posede doar cunoştințe strict necesare construirii cutiilor de dialog în cadrul mediului de dezvoltare. Modelul cutiei de dialog precum şi specificațiile aferente elementelor utilizate stau la baza implementării secvenței de cod prin care se obține funcționalitatea dorită.

Prin această separare se urmăreşte ca proiectarea cutiilor de dialog să fie realizată de către persoane aflate în relații strânse cu actualii şi/sau potențialii utilizatori, pentru ca rezultatul proiectării să corespundă nivelului de pregătire al utilizatorilor, sau unui nivel uşor de atins cu o şcolarizare minimală. Elementele personalizate de interfață utilizate au fost create conform regulilor elaborate şi prezentate de către autor în cadrul paragrafului 4.3.3.

Modelul cutiei de dialog precum şi specificațiile aferente elementelor utilizate stau la baza implementării secvenței de cod prin care se va obține funcționalitatea dorită.

5.3.2. Principii de funcționare ale cutiilor de dialog

La realizarea interfeței cu utilizatorul concepute s-au luat în considerare potențiale portări ale sistemului pe alte sisteme de operare (X-Windows (Unix/Linux)).

Pentru a controla într-o măsură cât mai mare afişarea şi comportamentul fiecărui element de interfață utilizat, la realizarea interfeței s-a urmărit evitarea utilizării altor biblioteci software (ex. Microsft Foundation Classes - MFC, biblioteci dinamice) ceea ce a avut ca efect reducerea semnificativă a problemelor la utilizatori.

In continuare se prezintă principiile de funcționalitate a cutiilor de dialog şi a interfeței cu utilizatorul, aplicate în dezvoltarea interfeței performante a sistemului Dietrich's.

- ***Încheierea editării şi verificarea datelor introduse***

Sintaxa datelor introduse se verifică în momentul în care utilizatorul doreşte să părăsească un element de dialog. In cazul in care sintaxa nu este corectă, utilizatorului nu îi este permisă părăsirea elementului de dialog.

Validitatea valorilor (încadrarea în limite) se verifică doar la părăsirea întregii cutii de dialog. In cazul în care o valoare este greşită, părăsirea cutiei de dialog nu este permisă, iar focusul se poziționează pe valoarea invalidă.

- ***Părăsirea cutiei de dialog, OK – Abbruch***

Intr-un anumit context, pentru închiderea cutiei de dialog se oferă doar funcțiile care au sens şi sunt univoce:

- butonul <OK> părăseşte o cutie de dialog şi modificările făcute se păstrează;
- butonul <*Abbruch*> (abandonare) este utilizat numai când modificările operate pot fi anulate sau dacă o funcție poate fi părăsită;
- butonul <*Ende*> părăseşte o cutie de dialog în care nu se pot face modificări.

- ***Structura unei cutii de dialog***

Analiza tipurilor şi modurilor de transfer a informației trebuie să conducă la o structură şi un comportament optimizat al elementelor de interfață din cadrul unei cutii de dialog. Pentru aplicația considerată s-a obținut următoarea **structură de bază:**

- cutia de dialog (Figura 5-6) tipică sistemului Dietrich's este mărginită superior de o bară de titlu albastră, sub care se află domeniul de introducere a datelor;
- în cadrul cutiilor de dialog prevăzute cu imagini auxiliare sau previzualizare, domeniul de introducere a datelor este situat în partea stângă iar fereastra de vizualizare în dreapta;
- în partea de jos a cutiei de dialog se găsesc funcțiile pentru părăsirea dialogului sau pentru alte apeluri;
- cutiile de dialog pot fi controlate integral cu tastatura şi/sau cu mouse-ul. S-a pus un accent deosebit pe posibilitatea de control optim prin utilizarea exclusivă a tastaturii;
- procesul de introducere a datelor în cutiile de dialog se desfăşoară de sus în jos;
- butonul <OK> este poziționat la baza cutiei de dialog, spre deosebire de aplicațiile tipice Windows unde este poziționat în partea din dreapta sus.

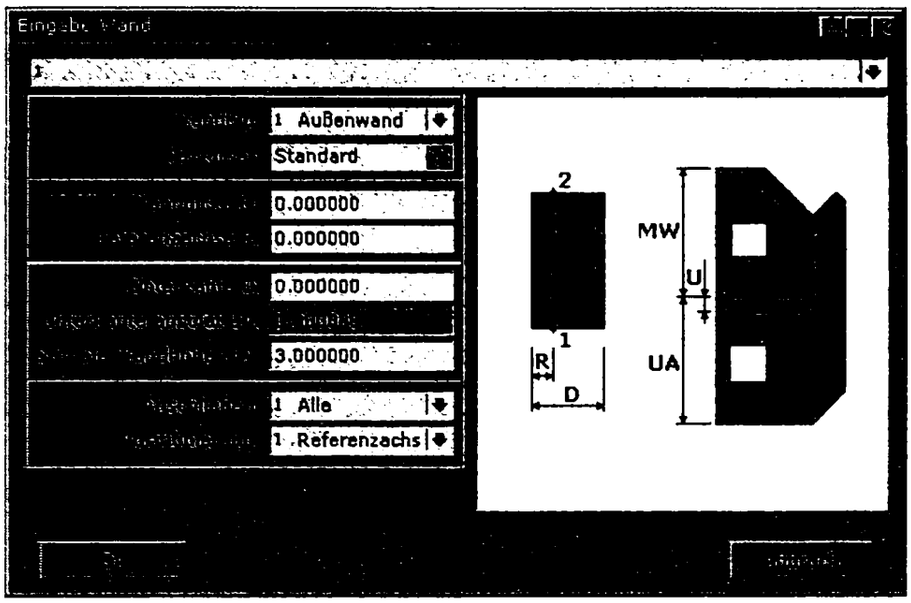


Figura 5-6. Structura tipică a unei cutii de dialog

Bara de titlu

Configuraţia barei de titlu este următoarea:

- în partea stângă a barei de titlu apare numele cutiei de dialog
- în partea dreaptă se află un buton cu o cruce corespunzător funcţiei "*Abbruch*" care atenţionează utilizatorul asupra datelor nesalvate.
- apelul funcţiilor temporare generale care nu se referă la un câmp special al cutiei de dialog se efectuează printr-un buton plasat la stânga butonului <*Abbruch*>;
- pentru funcţia de ajutor se include un buton cu semnul întrebării, conform stilului Windows.

Funcţii auxiliare specifice elementelor

Dacă unui element îi sunt asociate funcţii auxiliare speciale, la marginea sa dreaptă se dispune un buton cu un simbol special prin care se apelează funcţii temporare.

Caracteristicile grafice şi funcţionale sunt următoarele:

- butonul se desenează în interiorul elementului, aliniat la dreapta;
- la elementele de editare se scurtează câmpul de introducere propriu-zis iar în cazul *List-Box*-urilor prin translatarea spre stânga a săgeţii de deschidere a listei.

- ***Fontul şi dimensiunile***

Principalele caracteristici sunt următoarele:

- cutiile de dialog sunt adecvate constructiv pentru utilizarea unui font oarecare (inclusiv proporțional) de dimensiune oarecare (cel puțin două mărimi);
- poziția textelor şi a numerelor poate fi aliniată la stânga, la dreapta sau centrată;
- dimensiunile cutiei de dialog, dimensiunile şi pozițiile elementelor se adaptează la dimensiunea fontului.
- în funcție de formatul de imagine utilizat, dimensiunile imaginilor (în pixeli) pot rămâne constante (la *bitmap*) sau pot fi modificate (la cele vectoriale, ex. WMF);
- reprezentările grafice vectoriale cu scop ajutător se adaptează automat dimensiunii cutiei de dialog.

- ***Texte auxiliare***

Pentru fiecare element de introducere, ca şi pentru elementele meniului, la dezvoltarea cutiei de dialog s-a prevăzut posibilitatea afişării, în zona de jos a ecranului, a unui text descriptiv. Acesta apare automat când focusul este situat pe câmpul respectiv.

- ***Deplasarea în cadrul unei cutii de dialog***

Pentru o operare mai intuitivă şi mai eficientă în cadrul cutiilor de dialog s-au propus următoarele modificări față de comportamentul standard Windows:

- prin apăsarea tastei <Enter> nu se părăseşte cutia de dialog ci se încheie introducerea datelor în câmpul curent şi focusul trece la câmpul;
- *tastele săgeți* prezintă o funcționalitate diferită: deplasările ↑ şi ↓ se efectuează în cadrul cutiei de dialog, iar deplasările → şi ← se efectuează în cadrul elementului curent. Un *Edit Box* poate fi părăsit în orice moment cu săgețile ↑ şi ↓, pe când săgețile → şi ← au ca efect deplasarea în cadrul *Edit Box*-ului. Un *List Box* se părăseşte cu săgețile ↑ şi ↓ atâta timp cât nu este deschis. Cu săgețile → şi ← lista se deschide.
- dat fiind faptul că tasta <spațiu> este cea mai mare şi cel mai uşor de localizat, ea se utilizează ca şi comutator: la *Check Box*-uri se schimbă starea, la *List Box*-uri se deschide lista.

- ***Elemente ale cutiilor de dialog***

Pentru a asigura o utilizare cât mai simplă şi o perspectivă cât mai bună asupra modului de introducere a informației, în cadrul cutiilor de dialog realizate s-a apelat la un număr minim de elemente diferite de interfață. Figura 5-7 ilustrează o cutie tipică de dialog din cadrul interfeței proiectate.

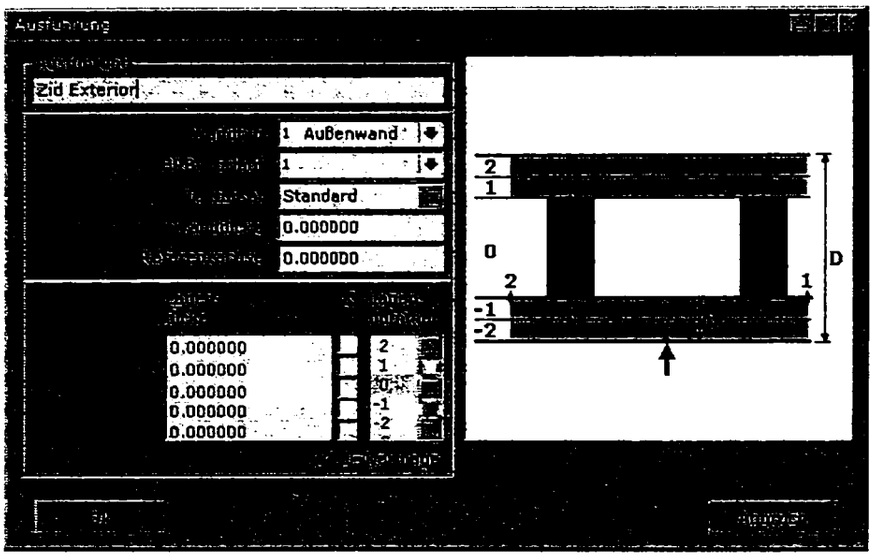


Figura 5-7. Elemente tipice conținute de o cutie de dialog

Principalele elemente de interfață utilizate sunt cele din tabelul 4-1.

Tabelul 4-1 Principalele elemente de interfață

Element de interfață	Funcție
Element Text	Text pentru afişare
Segment de cadru (*frame*)/linie	Linii pentru divizarea cutiei de dialog şi pentru cadru
Edit Box (text, număr)	Câmp de introducere pentru texte şi numere oarecare
List Box	Câmp de selecție cu un număr dat de înregistrări fixe
Check Box	Comutator
Buton	Apel de funcție, buton cu inscripție text
Buton cu imagine	Apel de funcție, buton cu inscripție imagine
List control	Liste în cadrul cărora elementele pot fi selectate pentru prelucrare, şterse sau se poate schimba poziția lor în cadrul listei.

Principalul motiv pentru care s-au personalizat elementele de interfața a fost acela ca fiecare versiune a sistemului de operare Windows introducea modificări vizuale si chiar funcționale fata de versiunea precedenta ceea ce putea crea probleme utilizatorului.

Principalele caracteristici vizuale şi de comportament pentru elementele interfeței realizate şi care rămân invariante față de versiunea sistemului de operare Windows sunt următoarele:

Elementul Text

- afişează text;
- de aliniere la stânga, la dreapta sau centrat;
- culoarea textului este în principiu albastră, dar poate fi setată pentru fiecare text în parte.

Elementul segment de cadru (*frame*):

- configurează şi divizează cutia de dialog;
- segmentele de cadru au o lățime de doi pixeli;
- segmentele orizontale au sus un rând de pixeli gri închis iar jos un rând de pixeli albi. Segmentele verticale au la stânga o coloană de pixeli gri închis iar la dreapta o coloană de pixeli albi.

Nu există un element special pentru cadre (*Group Box*), acestea se compun din segmente de cadre. Dacă este necesară introducerea unui titlu, atunci se utilizează un element de text normal.

Elementul *Edit Box* (Figura 5-8)**:**

- mulțimea datelor care poate fi introdusă este limitată de natura textului (de exemplu nume de fişier) şi în mod special în cazul introducerii numerelor;
- câmpul are un cadru normal, fondul este alb, culoarea textului este neagră. Dacă două *Edit Box*-uri verticale se ating, atunci dispar părțile de cadru dintre câmpurile individuale.

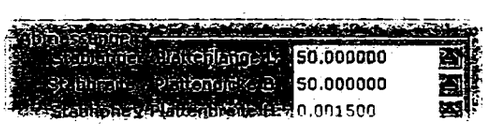


Figura 5-8. Elementul *Edit Box*

- pentru cazul în care o anumită mărime din cadrul cutiei de dialog se obține ca rezultat al unui calcul din alte mărimi (mărimi corelate), cutia de dialog este prevăzută cu o rutină de calcul a mărimii respective care se afișează pe un fond gri închis - caracteristic pentru indicarea valorilor rezultate din calcule. Dacă una dintre mărimi este modificată de către utilizator, rutina recalculează automat noua valoare a mărimii care se afișează tot pe fond gri închis;
- tastele săgeți ↑ și ↓ încheie editarea în orice moment și se trece la câmpul precedent, respectiv următor. Același lucru este valabil pentru <Tab> respectiv <Shift>+<Tab>;
- tasta <Enter> finalizează editarea și face trecerea la următorul element.
- *edit box*-ul devine câmp de afișare când modificarea valorii afișate este interzisă. În acest caz fondul textului se modifică în culoarea gri închis iar la deplasarea prin cutia de dialog acest câmp se ignoră.

Elementul *List Box* (Figura 5-9)**:**

- aliniat la stânga apare un număr albastru cu caractere mici, care permite selecția rapidă prin tastatură a opțiunii corespunzătoare, iar lângă acesta, cu negru, apare opțiunea propriu-zisă;
- la capătul din dreapta al câmpului, după o linie de demarcație gri închis, urmează o săgeată albastră orientată în jos ce deschide lista.

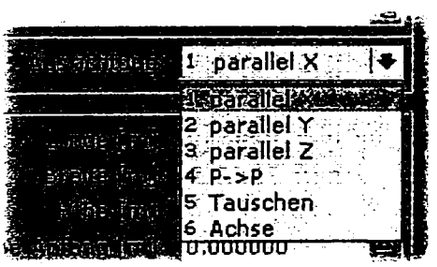


Figura 5-9. Elementul *List Box*

- pentru a afișa complet opțiunile lungi, lățimea listei se ajustează automat având ca reper cel mai lung șir din listă;
- la marginea din dreapta a listei apare un cursor de navigare (*scroll-bar*) vertical dacă este necesar.

- la primirea focusului, lista propriu-zisă se deschide cu tastele <Enter>, <spațiu>, ← sau → cu opțiunea actuală selectată;
- selecția se încheie cu <Enter>, <Tab> sau <Shift>+<Tab> astfel că focusul se deplasează pe elementul de dialog următor sau precedent;
- dacă imediat ce *List Box*-ul a primit focusul, se tastează ↑, ↓ sau <Tab> respectiv <Shift>+<Tab>, focusul se deplasează pe un alt element al cutiei de dialog, iar selecția actuală din listă se păstrează.

O metodă eficientă propusă de selectare în *List Box*-uri este reprezentată de introducerea unor numere de selecție rapidă (acceleratori), aceeaşi idee fiind aplicată şi în cazul accesului rapid la punctele de meniu. Înregistrările din liste sunt numerotate consecutiv. Dacă focusul este pe un *List Box* şi se introduce un asemenea număr, atunci se selectează înregistrarea corespunzătoare, iar focusul se mută pe elementul următor.

Din cele cunoscute de autor, această soluție este **cea mai rapidă formă de comandă** al unui *List Box*.

Combo Box-urile existente în Windows nu se utilizează deoarece combinația de câmpuri modificabile şi câmpuri fixe este relativ rară. Atunci când este necesar, în listă se va utiliza o opțiune speciala, la selecția căreia deschizându-se un *Edit Box* pentru a fi introdus un text liber.

Elementul *Check Box* (Figura 5-10)

- se utilizează atât individual cât şi în combinație cu liste (v. Figura 5-11);
- simbolul utilizat este *check-mark*-ul;
- pentru ca suprafața sensibilă la clicul de mouse să fie mai mare, *Check Box*-ul a fost proiectat dimensional superior comparativ corespondentului său din Windows;
- dacă focusul este pe un *Check Box*, starea sa poate fi modificată cu tastele <spațiu>, ← şi →, iar focusul rămâne pe *Check Box*;
- cu <Enter>, <Tab>, <Shift>+<Tab>, ↑ şi ↓ starea comutatorului rămâne nemodificată, iar focusul părăseşte controlul;
- diagrama tranzițiilor de stare care modelează comportamentul unui *check-box* precum şi modelul de interactor echivalent *check-box*-ului sunt analizate în paragraful 3.5.1.4.

Figura 5-10. Elementul Check Box

Elementul Buton (Figura 5-20)

Figura 5-11. Elementul Buton

- dacă focusul este pe un buton, acesta va fi declanşat prin tasta <Enter>;
- focusul este mutat cu tastele <Tab>, <Shift>+<Tab>, ↑ şi ↓. În această situație, tastelor ← şi → nu le-a fost asociată nici o funcție;
- diagrama tranzițiilor de stare care modelează comportamentul unui buton precum şi modelul de interactor echivalent butonului sunt analizate în paragraful 3.5.1.5

Elementul Buton cu imagine

- reprezintă un mod sugestiv de prezentare a funcției pe care o are un buton, în cadrul aplicației fiind create şi utilizate butoane cu imagine;
- butonul cu imagine prezintă o funcționalitate similară elementului buton normal, însă pe buton se află o imagine raster sau vectorială. În mod frecvent s-a optat pentru utilizarea imaginilor de tip vectorial, acestea prezentând un avantaj în cazul în care dimensiunile butonului se modifică (a suprafeței de comutare), dat fiind faptul că celelalte elemente se adaptează mărimii fontului.

Elementul *List Control* (Figura 5-12)

- sub *List Control* se află funcțiile cutiei de dialog pentru prelucrarea înregistrărilor marcate. Ordinea operațiilor este fixată: mai întâi trebuie efectuată o selecție în listă, apoi va fi selectată funcția;
- tastele <spațiu>, ← și → permit setarea și resetarea marcajului;

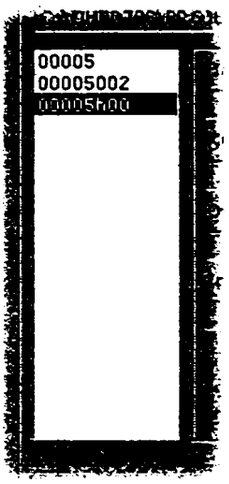


Figura 5-12. Elementul List Control

- ***Elemente combinate, tabele***

La proiectarea interfeței cu utilizatorul, pentru tabele nu s-a prevăzut nici un element în mod special. Un tabel se compune din celelalte elemente ale cutiei de dialog (Figura 5-13) iar funcționalitatea tabel se comandă prin intermediul cutiei de dialog.

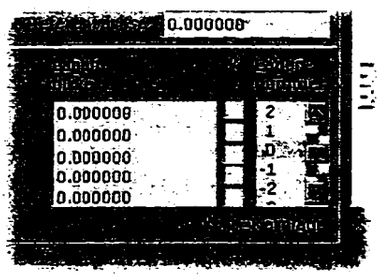


Figura 5-13. Exemplu de elemente combinate

Într-un tabel se plasează pe aceeași linie, succesiv, *Edit Box*-uri, *List Box*-uri și *Check Box*-uri. Prin intermediul *scroll-bar*-urilor se vor parcurge nu elementele propriu-zise, ci doar conținutul lor.

5.3.3. Introducerea de date

Utilizatorul, în cadrul procesului de proiectare, trebuie să specifice aplicației mai multe tipuri de valori. Pentru că acelaşi tip de valori trebuie introdus în diverse situații, unul din obiectivele proiectării interfeței constă în realizarea unor funcții şi cutii de dialog ce pot fi reutilizate. S-au identificat următoarele situații de introducere de date care trebuie luate în considerare:

- introducerea de coordonate libere 3D şi 2D;
- alegerea unui punct pe o muchie;
- introducerea distanței până la un plan de tăiere;
- introducerea unui unghi;
- introducerea de valori pentru copierea sau multiplicarea în jurul sau de-a lungul unei axe.

Pentru ca utilizatorului să i se ofere posibilitatea de ajustare a coordonatelor, în proiectarea interfeței s-a optat pentru soluția afişării unei cutii de dialog pentru fiecare punct ales în spațiul 3D (Figura 5-14).

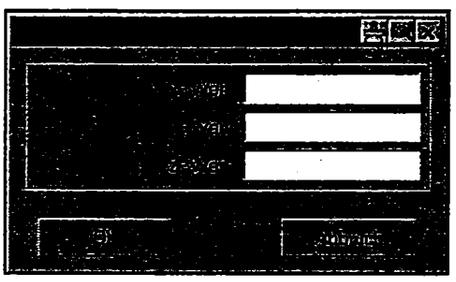


Figura 5-14. Introducerea de coordonate libere

La alegerea unui punct pe o muchie, utilizatorului i se prezintă o cutie de dialog (Figura 5-15) ce conține coordonatele punctului ales pe linie şi distanța față de capătul ales.

La modificarea unei coordonate, celelalte două se recalculează automat astfel încât punctul rezultat se află de asemenea pe linia selectată.

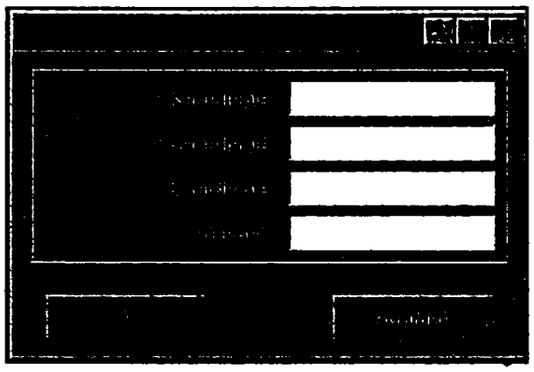


Figura 5-15. Alegerea unui punct pe o muchie

Pentru selectarea unui plan, în cazul unei operații de tăiere a unui element (o bară), se indică distanța până la planul de tăiere de-a lungul barei (Figura 5-16).

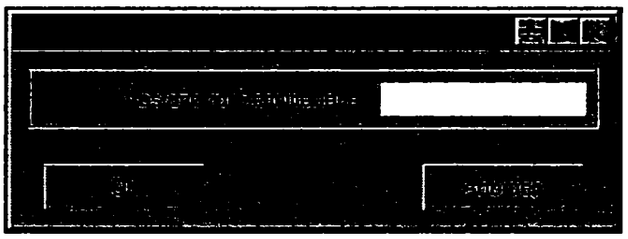


Figura 5-16. Introducerea distanței până la planul de tăiere

Rotirea unui obiect în jurul unei axe necesită selectarea prealabilă a unui obiect şi a unui/unor punct/puncte, iar în etapa a doua introducerea valorii unghiului de rotație (Figura 5-17).

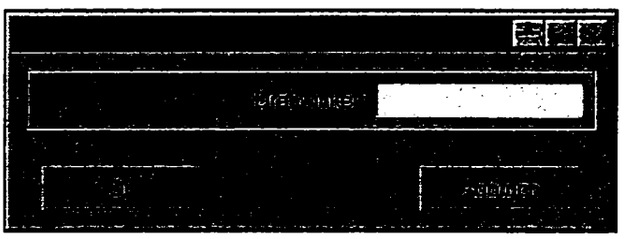


Figura 5-17. Introducerea unghiului de rotire

Acelaşi principiu a fost respectat şi în cazul deplasării unui element într-o anumită direcție.

Pentru copierea sau multiplicarea unui obiect în jurul sau de-a lungul unei axe s-au proiectat si implementat doua metode originale. Pentru introducerea datelor necesare acestei

operații utilizatorului i se prezintă o cutie de dialog în care trebuie introduse valori pentru numărul copiilor, unghiul sau distanța verticală dintre două copii (Figura 5-18).

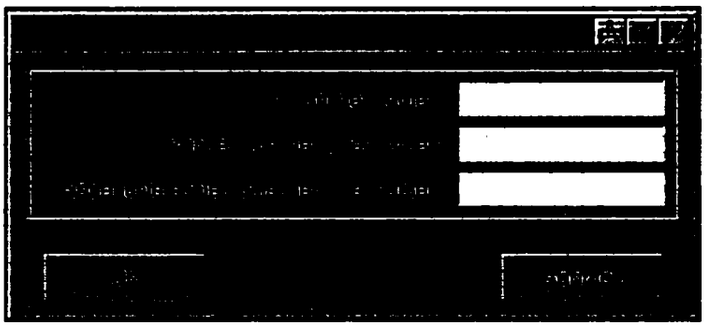


Figura 5-18. Parametri de copiere/multiplicare în jurul unei axe

In cazul celei de-a doua metode, utilizatorul poate să obțină copii ale unui element de-a lungul unei axe prin introducerea a două valori dintre cele trei enumerate: distanța dintre prima şi ultima copie, distanța dintre două copii succesive sau numărul de copii (Figura 5-19). Introducerea valorilor a doi dintre parametri implică (re)calcularea automată a celui de-al treilea.

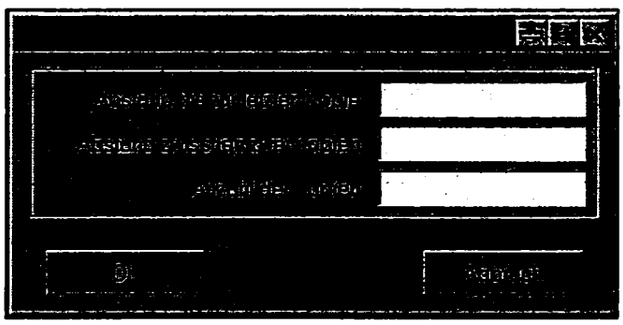


Figura 5-19. Parametrii de copiere/multiplicare de-a lungul unei axe

Sistemul permite şi generarea de corpuri 3D de rotație. În acest sens, pentru introducerea caracteristicilor generale ale corpului ce va fi generat, utilizatorul trebuie să completeze în cutia de dialog un set de date (Figura 5-20):

- indentificator
- descriere
- dimensiunile x şi y
- numărul de fețe poligonale.

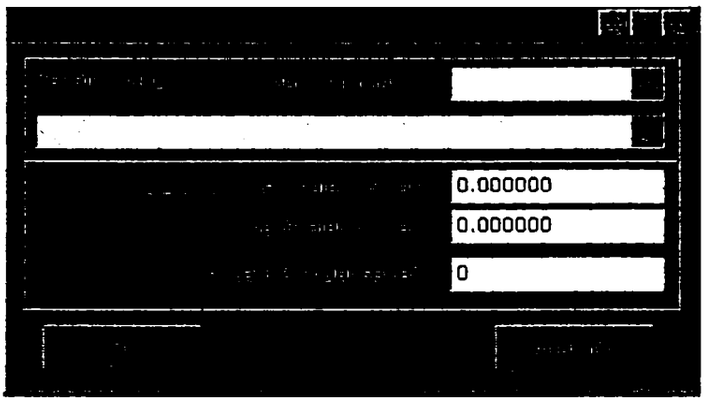


Figura 5-20. Generarea de corpuri de rotație

In continuare se indică punctele care formează conturul. Fiecare punct introdus se confirmă printr-o cutie de dialog (Figura 5-21), unde opțional se pot ajusta coordonatele.

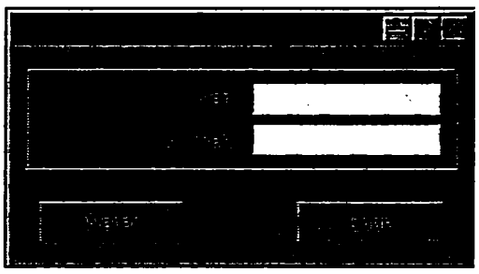


Figura 5-21. Introducerea punctelor conturului exterior

5.3.4. Parametri globali ai interfeței cu utilizatorul

Pentru obținerea unei flexibilități şi eficiențe sporite s-a pus problema ca utilizatorul să-şi poată adapta într-o cât mai mare măsură configurația interfeței aplicației avută în vedere. In acest sens, în cadrul acestui paragraf, se determină un set de parametri reconfigurabili, fiind posibilă modificarea lor de către utilizator.

Cutia de dialog (Figura 5-22) conține parametrii globali ai interfeței (de elemente, de reprezentare şi generali).

În cadrul secțiunii pentru elemente se pot activa/dezactiva oricare din elementele de interfață care sunt afişate:

- sistemul de coordonate al modelului;
- fondul de culoare neagră (alb în mod normal);
- meniul permanent din dreapta ecranului;

- fereastra pentru mesaje text suplimentare;
- mesaje de ajutor;
- linia de stare;
- reprezentarea monocoloră;
- anunțarea preliminară a selecției de volume.

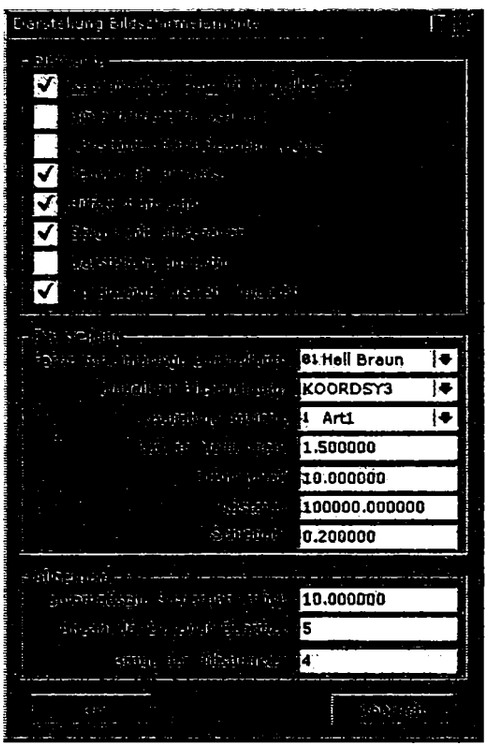


Figura 5-22. Parametrii globali ai interfeței cu utilizatorul

In cadrul secțiunii pentru reprezentarea informației sunt cuprinse următoarele opțiuni:

- culoarea utilizată în reprezentarea monocoloră;
- modelul utilizat pentru reprezentarea direcției de privire;
- durata afișării mesajelor;
- incrementul unghiului de rotație al modelului la rotirea modelului în cadrul vederii;
- distanța până la model;
- incrementul pasului de deplasare la translatarea modelului în cadrul vederii.

Parametrii generali stabilesc următorii parametri:

- intervalul de timp necesar până la declanşarea salvării automate;

- numărul paşilor de anulare/repetare a unei operaţii;
- dimensiunea punctului ajutător.

5.3.5. Bare de instrumente (Toolbars)

Deşi numărul funcțiilor din cadrul programului este relativ mare, utilizatorul lucrează curent doar cu un subset dintre acestea. Pentru a înlesni accesul la subsetul respectiv, utilizatorului i s-au pus la dispoziție o colecție de bare de instrumente care îi oferă acces direct la funcții.

Datorită faptului că barele de instrumente ocupă spaţiu important în cadrul ferestrei de lucru, s-a optat pentru o soluție care permite utilizatorului să selecteze numai barele de instrumente şi butoanele pe care le doreşte afişate pe ecran.

In acest scop, în cadrul proiectării au fost realizate cutiile de dialog de configurare a barelor de instrumente (Figura 5-23).

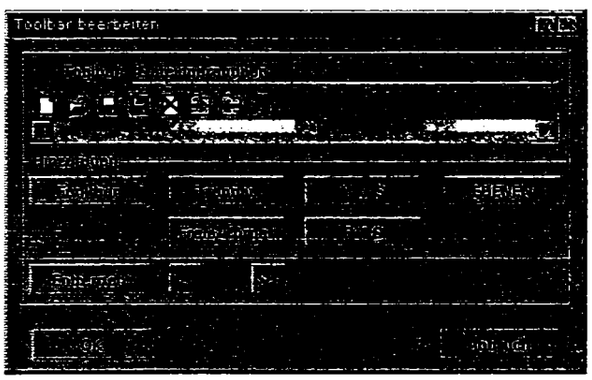


Figura 5-23. Configurarea barelor de instrumente

Cutia de dialog pentru setările generale cuprinde opțiuni pentru

- dimensiunea icoanelor;
- dimensiunea literelor;
- afişarea, configurarea şi reinițializarea barelor de instrumente.

Butoanele de creare şi configurare a barelor de instrumente apelează o nouă cutie de dialog în cadrul căreia se pot configura butoanele prin adăugare, poziționare sau ştergere. Cutia de dialog cuprinde numele barei şi o reprezentare/simulare a acesteia, inclusiv toate butoanele aparținătoare. Pentru adăugarea unui buton nou se selectează în prealabil poziția acestuia în cadrul barei de instrumente şi automat butonul de selecție a funcției ce va fi asociată se activează. Pentru selecția funcției, utilizatorul parcurge meniurile programului şi

selectează funcția dorită, preluându-se automat şi icoana corespunzătoare. Pe lângă butoane se pot introduce şi *list-box*-uri, prin butonul corespunzător.

5.3.6. Operații cu fişiere

Pentru operarea cu fişiere, cutiile de dialog corespunzătoare proiectate au la bază acelaşi model (Figura 5-24). Acestea conțin:

- numele proiectului;
- numele poziției;
- informații sau comentarii pentru poziția curentă;
- calea completă pe disc a proiectului;
- lista tuturor pozițiilor din cadrul proiectului;
- buton de activare a afişării de informații adiționale/previzualizare.

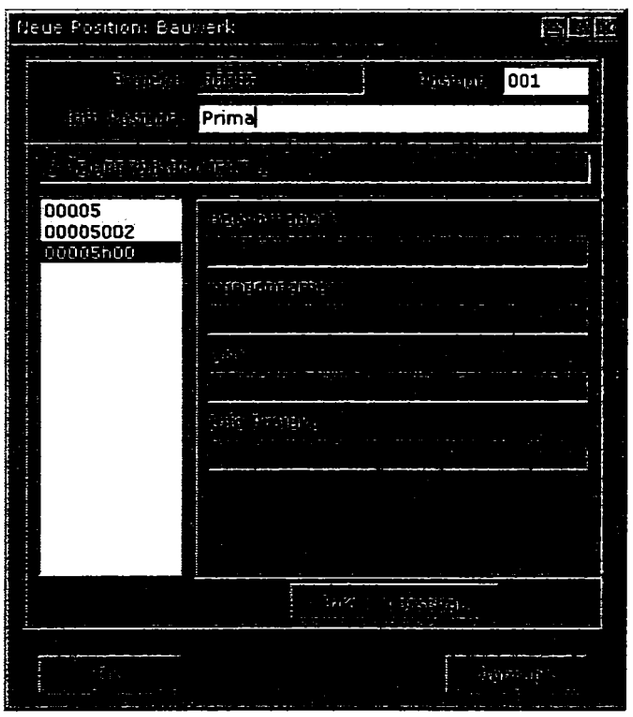


Figura 5-24. Model de cutie de dialog de operare cu fişiere

În funcție de starea butonului <informații>previzualizare> se afişează în acelaşi loc informații adiționale legate de proiectant şi de proiect sau se poate revizualiza modelul corespunzător poziției selectate.

5.3.7. Biblioteci de modele de proiectare

În partea superioară a cutiei de dialog proiectate pentru încărcarea unui model dintr-o bibliotecă, utilizatorului îi este indicată calea către biblioteca selectată. Alegerea unei biblioteci se realizează pe o structură arborescentă corespunzătoare ierarhiei directoarelor (Figura 5-25). Ultimul element al acestei structuri este un element individual din cadrul bibliotecii.

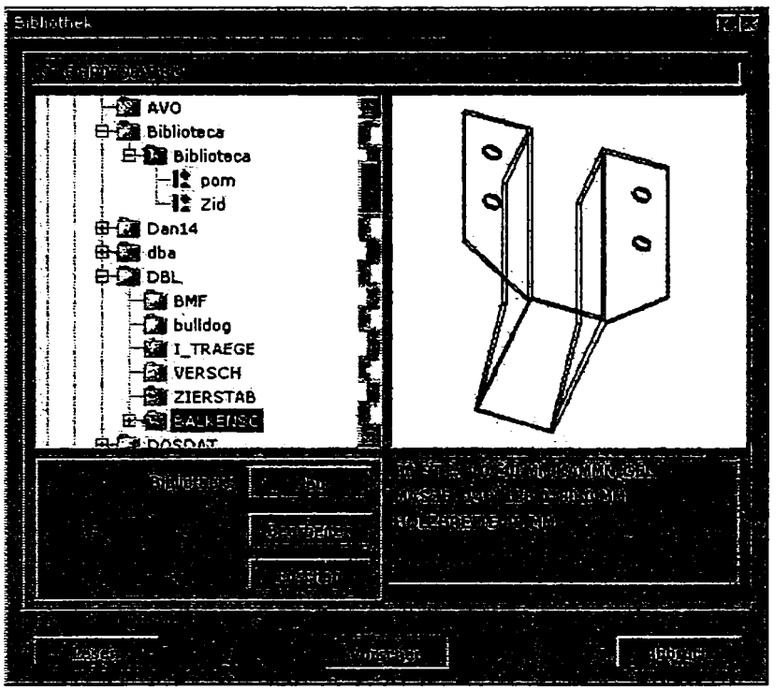


Figura 5-25. Alegerea elementelor dintr-o bibliotecă

În partea dreaptă a structurii de biblioteci se află o fereastră în care, dacă previzualizarea este activată, se va desena elementul curent selectat. Dacă previzualizarea nu este activată sau nu este selectat nici un element de bibliotecă, fereastra de previzualizare va afişa informaţii textuale adiţionale legate de elementul selectat.

Butonul de preluare se activează numai în cazul în care este selectat un element din cadrul unei biblioteci.

Pentru crearea unei biblioteci sau a unui element de bibliotecă, în cutia de dialog corespunzătoare (Figura 5-26) se introduc informaţii despre directorul, numele bibliotecii sau al elementului şi o descriere în funcţie de poziţia selectată din cadrul arborelui de biblioteci.

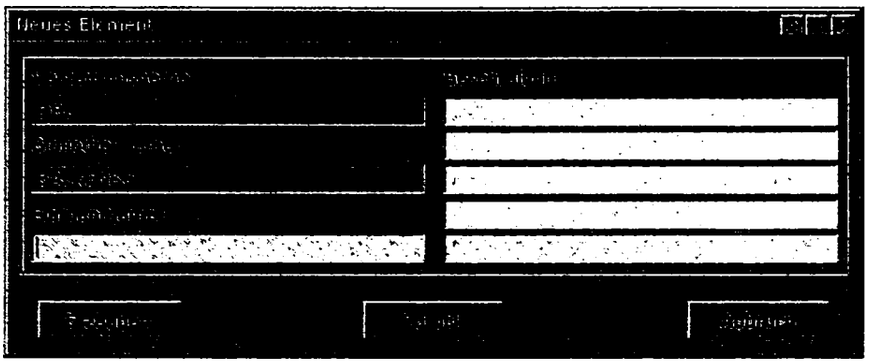


Figura 5-26. Adăugarea unui element nou

In cazul preluării unui obiect din bibliotecă, acesta este prezentat într-o fereastră specială (Figura 5-27) unde se va indica modul său de plasare în spațiul modelului.

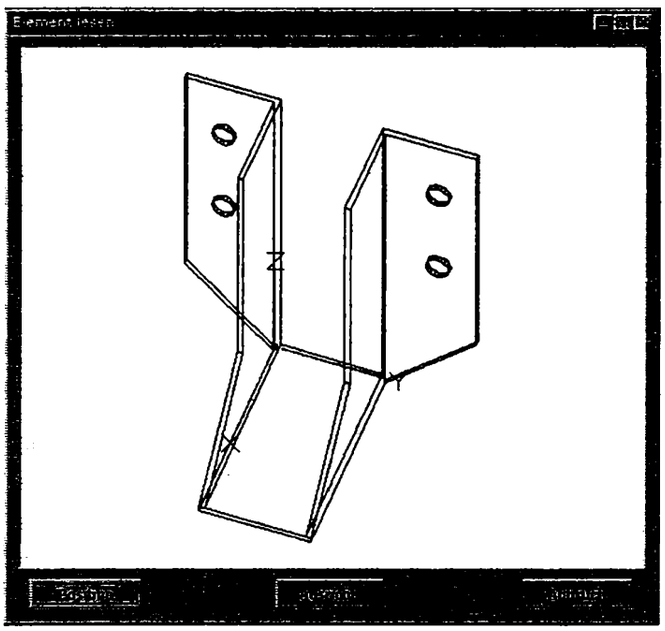


Figura 5-27. Citirea unui element de bibliotecă

Dimensional, elementul de bibliotecă reprezentat în fereastra de vizualizare este mult mai mare decât cel din fereastra de previzualizare pentru a reda utilizatorului o cât mai bună imagine asupra formei acestuia. La alegerea punctului de inserție se pot folosi funcții ajutătoare de aliniere la anumite puncte de reper.

5.3.8. Punct-simboluri (PSB)

5.3.8.1 Cerințe

Pentru eficientizarea procesului de aplicare a seturilor de prelucrări, evitând operații repetitive din partea utilizatorului, în cadrul sistemului Dietrich's s-a considerat necesar implementarea de punct-simboluri. Utilizarea punct-simbolurilor reprezintă o metodă rapidă şi simplă de plasare a unei singure prelucrări sau a unui grup, pe una sau mai multe bare. In continuare se prezintă funcționalitățile pentru care este necesară proiectarea şi implementarea unor elemente de interfață cu utilizatorul în cazul utilizării PSB-urilor.

Plasarea PSB-urilor

- *Poziționarea punct-simbolurilor pe bară prin specificarea unui punct de referință* utilizând opt moduri distincte:
  - *singulară-local:* poziționarea punct-simbolului se face de la capătul cu originea barei;
  - *singulară-global* poziția punct-simbolului este la acelaşi nivel dat de punctul de tăiere al nivelului cu fiecare bară;
  - *singulară-capăt de bară*: în acest caz se plasează câte două punct-simboluri, poziția este dată relativ la fiecare capăt al barei;
  - *serie-local*: se plasează în acelaşi loc unul sau mai multe punct-simboluri, poziția fiecărui punct-simbol se calculează față de capătul unde se află originea barei;
  - *serie-global*: se plasează în acelaşi loc unul sau mai multe punct-simboluri, poziția punct-simbolului este la acelaşi nivel dat de punctul de tăiere al nivelului cu fiecare bară;
  - *serie-capăt de bară*: se plasează la fiecare capăt al barei unul sau mai multe punct-simboluri, poziția este dată relativ la fiecare capăt al barei;
  - *de-a lungul unei linii*: pozițiile de plasare a punct-simbolurilor pe bare este dată de intersecția unei linii date de utilizator cu barele active;
  - *pe punct*: cu această funcție se plasează pe un punct ales pe una sau mai multe bare mai multe punct-simboluri.

- *Specificarea distanței față de un punct de referință* prin utilizarea a opt abordări:

- *măsurare în raport cu o direcție paralelă cu axa barei.* Pentru a determina o distanță utilizând această funcție se alege mai întâi un punct de referință. Distanța de la capătul cu originea barei până la acest punct reprezintă distanța căutată;
- *măsurare în raport cu o direcție paralelă cu o linie oarecare.* Inițial se selectează cele 2 capete ale unei linii. În continuare se selectează în spațiu un punct de referință. Distanța se măsoară de-a lungul liniei determinate. Distanța de la capătul de început al barei până la punctul selectat reprezintă poziția unde va fi plasat punct-simbolul;
- *măsurare în raport cu o direcție perpendiculară pe o linie oarecare*: după selectarea unei linii, se va selecta un punct de referință; distanța căutată reprezintă lungimea perpendicularei din punctul de referință pe linia selectată.
- *măsurare în raport cu o direcție perpendiculară pe un nivel global*: cu ajutorul celor trei funcții se calculează distanțele de la un punct la planele XoY, XoZ, YoZ.
- *măsurare în raport cu o direcție perpendiculară pe un nivel oarecare*: cu aceste două funcții se determină distanța de la un punct la un plan determinat de trei puncte sau două linii.

Utilizarea editorului de PSB

Generarea de punct-simboluri implica utilizarea unui editor special unde PSB-urile pot fi create, modificate şi/sau salvate în biblioteci. Editorul trebuie sa ofere utilizatorului următoarele facilități:

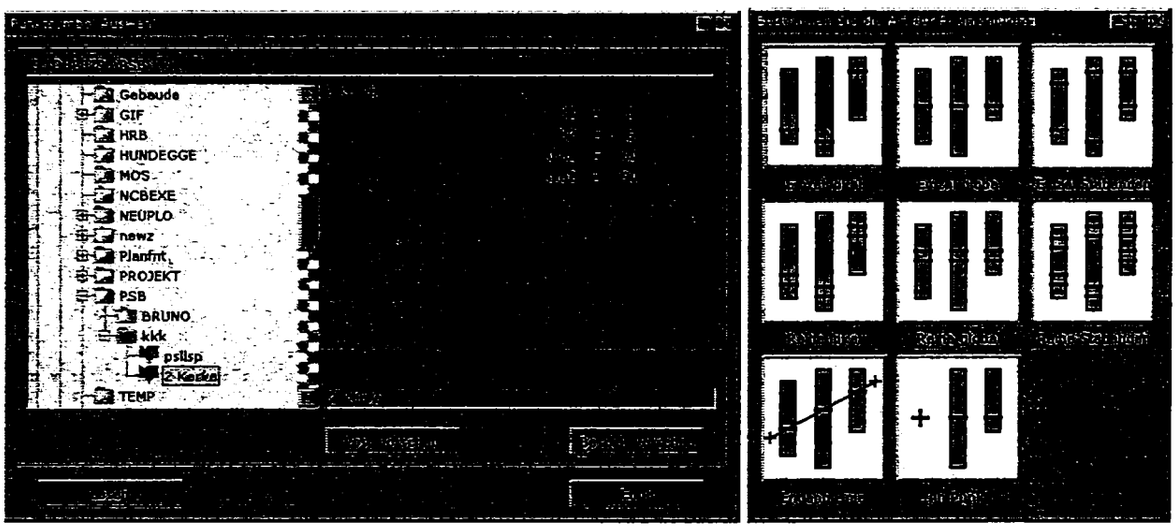
- posibilitatea utilizării de variabile a căror valori sunt completate de către utilizator în momentul plasării pe o bară;
- funcțiile disponibile în cadrul procesului de editare a unui PSB sunt *Nou, Preluare, Prelucrare* şi *Ştergere*. Manifestarea acestor funcții depinde de tipul elementului selectat. Este necesară implementarea unei previzualizări şi a afişării listei variabilelor utilizate;
- modificarea unui punct-simbol implică deschiderea cutiei de definire a variabilelor inițializată cu parametrii PSB-ului;
- definirea şi editarea variabilelor prin nume, valoare implicită şi unitate de măsură;

- definirea şi editarea PSB-ului prin specificarea prelucrărilor din componența sa;
- utilizarea unui editor de formule pentru calculul mărimilor utilizate ca parametri pentru prelucrări.

5.3.8.2 Soluție de implementare a PSB

In continuare se prezintă soluția de proiectare şi implementare a elementelor de interfață cu utilizatorul pentru utilizarea punct-simbolurilor, conform cerințelor formulate în paragraful 5.3.8.1.

La selectarea unui punct-simbol se deschide o cutie de dialog, prezentată în figura 5-28a.



a) b)

Figura 5-28. Selecția unui punct-simbol şi a modului de poziționare

- S-au implementat modurile de plasare enunțate în paragraful 5.3.8.1 rezultând reprezentările simbolice din figura 5-28b.

- Specificarea distantei față de un punct de referință. După indicarea barei de referință se afişează cutia de dialog prezentată în figura 5-29, care conține reprezentările simbolice corespunzătoare abordărilor de măsurare a distanței față de un punct de referință considerate in paragraful 5.3.8.1:

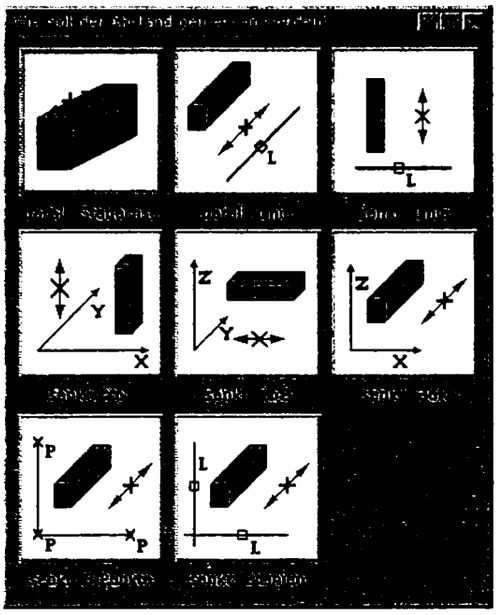


Figura 5-29. Modurile de măsurare a deplasamentului

Editorul PSB

Procesului de editare a unui punct-simbol demarează prin utilizarea cutiei de dialog din figura 5-28.

Calea către biblioteca curentă este specificată atât în *text-box*-ul de la începutul cutiei cât şi în lista arborescentă de selecție a bibliotecii. Ultimul nivel al listei arborescente reprezintă un punct-simbol. Fiecare element al listei arborescente are asociată o icoană sugestivă pentru a uşura determinarea tipului elementului. În funcție de elementul selectat, în jumătatea din dreapta a cutiei de dialog s-a proiectat o fereastră în care se afişează o previzualizare de identificare: un desen sau o listă cu numele variabilelor alături de valorile implicite. Dacă elementul selectat este un director, atunci se afişează o imagine dată de un fişier predefinit aflat în acest director.

În cazul apelării funcțiilor *Nou, Preluare* şi *Prelucrare,* dacă este selectat un director se deschide cutia de dialog de editare a informațiilor generale ale unui fişier de punct-simboluri (Figura 5-31).

Dacă selecția se află pe un punct-simbol, apelul aceloraşi funcții deschide cutia de editare a variabilelor sale (Figura 5-32).

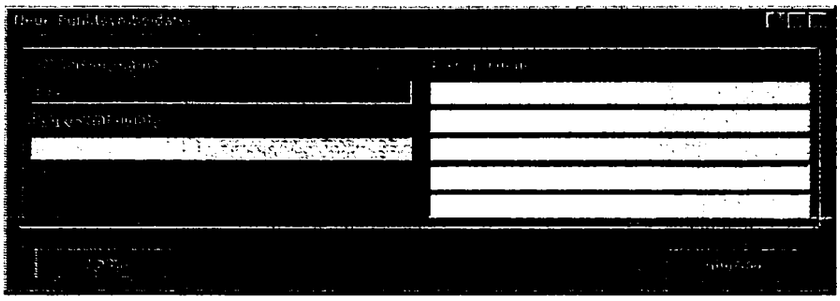


Figura 5-31. Crearea unui nou fişier de punct-simbol

Coloana de *check-box*-uri din cadrul cutiei de dialog indică faptul că variabila corespunzătoare are o valoare fixă sau o valoare ce poate fi modificată. Coloana a doua conține numele variabilelor, numărul acestora fiind virtual nelimitat. Următoarele două coloane conțin valorile implicite şi unitățile de măsură.

Figura 5-32. Cutia de editare a variabilelor punct-simbolului

Datorită localizării utilizatorilor în țări cu unități de măsură diferite, acestea pot fi exprimate prin număr scalar/metri/centimetri/milimetri/țoli (*inch*)/picioare-țoli. Dacă o variabilă primeşte una din unitățile de măsură enumerate mai sus exceptând „număr scalar" înseamnă că reprezintă o măsură a lungimii. Deoarece, intern, calculele se efectuează exclusiv în metri, acestea sunt transformate automat. In cazul valorilor exprimate în picioare-țoli acestea trebuie să conțină trei separatori zecimali.

Se mai pot specifica următoarele tipuri de variabile:

- grade: o variabilă exprimată în grade reprezintă un unghi.
- A,B: reprezintă o referință de capăt: de început sau sfârşit;
- C-F: reprezintă o față de referință laterală;

- A-F: reprezintă o față de referință;
- W/M: o variabilă de tipul *Werkstadt/Montage* poate avea doar valorile 0 și 1.

Cutia de dialog pentru editarea punct-simbolului (Figura 5-33) realizează toate operațiile de generare/operare a unui punct-simbol.

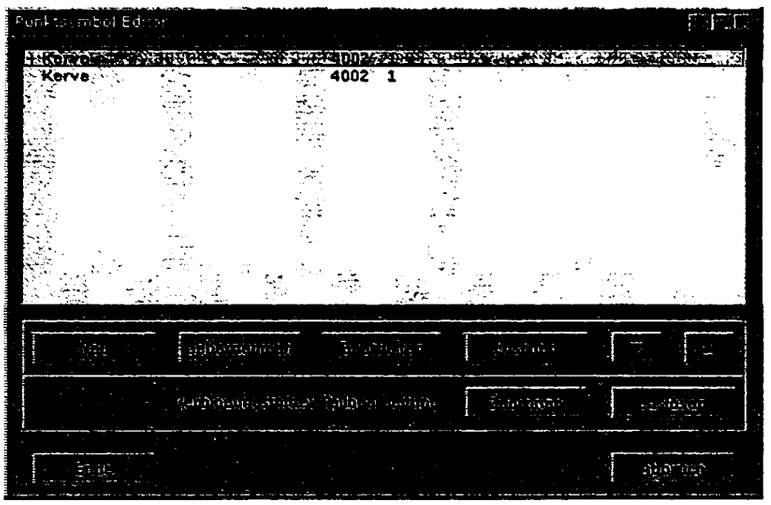


Figura 5-33. Lista de prelucrări din cadrul unui punct-simbol

La generarea unui PSB nou se deschide o listă ce cuprinde toate prelucrările posibile din care se alege cea dorită (Figura 5-34). După selectarea prelucrării se deschide cutia de precizare a parametrilor specifici acelei prelucrări (Figura 5-35).

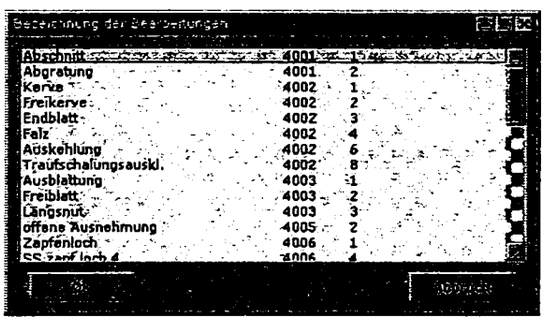


Figura 5-34. Selecția unui tip de prelucrare și specificarea parametrilor săi

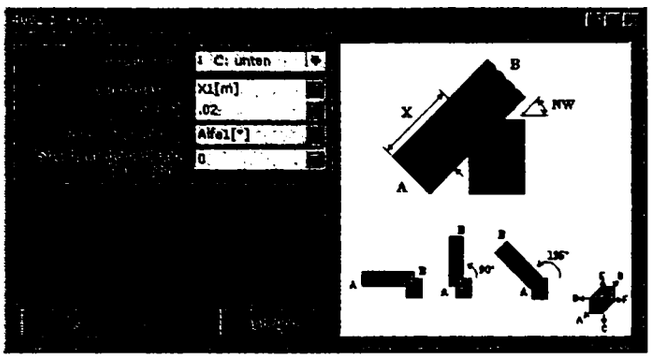


Figura 5-35. Specificarea parametrilor unei prelucrări

Butonul de preluare (*Uebernehmen*) permite copierea unei prelucrări deja existente. Pentru o flexibilitate sporită în editare, adițional s-au implementat funcții de editare, ştergere şi ordonare a prelucrărilor precum şi o funcție de abandonare a procedurii curente, caz în care programul revine la momentul selecției unui punct-simbol.

În funcție de caracteristici, la definirea unei prelucrări poate apare necesitatea introducerii mai multor tipuri de valori, pentru cazurile în care valorile individuale sunt cunoscute, cum sunt: fețe de referință, capăt de referință, sau variabilă, se utilizează un *drop-list* din care utilizatorul va alege.

Câmpurile de introducere se completează cu valori concrete, cu un nume de variabilă sau cu o formulă. Ca alternativă la introducerea directă a unei formule se apelează editorul de formule prin utilizarea butonului din dreapta câmpului de editare, rezultatul obținut fiind transferat în câmpul respectiv.

Editorul de formule. Cutia de dialog proiectată şi implementată pentru *editorul de formule* este prezentată în figura 5-36.

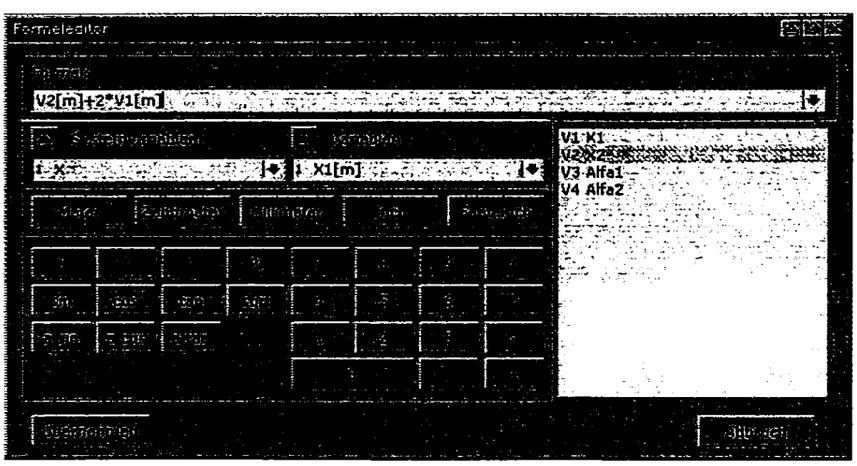


Figura 5-36. Editorul de formule

Primul câmp de editare conține formula care poate fi editată, fără limită relativ la numărul de caractere. Formulele deja introduse sunt disponibile într-o listă, păstrată până la părăsirea programului.

Pe lângă variabilele definite de utilizator, se folosesc şi variabile dintr-un set definit de sistem. Acestea sunt de tip X, lungime, lățime, înălțime, secțiune paralelă cu perpendiculara la o față de referință şi valoare de secțiune a feței de referință. Pe lângă acestea mai există două variabile ‚fața de sus' şi ‚fața de jos', dar acestea pot fi folosite doar pentru un parametru de tipul „față de referință".

În câmpul de tip *drop-list* se află variabilele definite de către utilizator. La selecția unei variabile, prin apăsarea butonului de preluare a informației, variabila însoțită de unitatea de măsură va fi inserată la poziția din formulă a cursorului.

Ca şi alternativă la introducerea de la tastatură, cifrele şi operatorii pot fi introduşi cu ajutorul butonului corespunzător din cadrul editorului de formule. În cazul acestui mod de introducere a datelor nu se efectuează nici o analiză de sintaxă.

Formula introdusă poate fi preluată în cutia de dialog de editare a prelucrării sau poate fi abandonată.

5.3.9. Liste de componente şi de dimensiuni generate în sistemul Dietrich's

In cadrul proiectării unei construcții, generarea de liste de componente şi dimensiuni reprezintă un proces important. Modul în care aceste liste sunt generate şi prezentate utilizatorului influențează într-o foarte mare măsură felul în care acesta le înțelege şi le interpretează. Alcătuirea listelor trebuie să fie pe înțelesul utilizatorului (şi nu al programatorului), iar datele pe care aceste liste le conțin trebuie să furnizeze informații complete şi fără redundanțe (v. paragraful 4.3.2.17).

Trebuie acoperite de fapt două etape: sortarea elementelor şi generarea propriu-zisă a listelor de materiale.

5.3.9.1 Sortarea elementelor

În scopul obținerii listelor de materiale, sortarea reprezintă operația principală şi are ca obiectiv ordonarea barelor după mai multe criterii (tip de bară, material). Reperele procesului de sortare pot fi sintetizate prin:

Ordinea. La sortare se iau în considerare, în principiu, elemente care au ca tip de bară (*Bauteilart*) "*Stab*" şi ca material lemnul ("*Holz*").

Gruparea şi prelucrarea elementelor rămase se face în următoarea ordine:

- tip de bară "*Platte*" şi materialul "*Holz*"
- tip de bară "*Platte*", material oarecare
- tip de bară "*Stab*", material "*Metall*"
- tip de bară "*Stab*", material oarecare
- tip de bară "*Stahlteil*", material "*Metall*"
- tip de bară "*Beschlag*", material oarecare

Un alt criteriu de diferențiere al unei categorii de elemente obținută prin sortarea anterioară este denumirea lor (*Bezeichnung*).

În final se sortează toate elementele rămase, independent de informație.

Dimensiunile. La diferențierea elementelor pe baza dimensiunilor lor se admite o toleranță de ± 0,5 mm, astfel că elementele care nu diferă în lungime, lățime sau înălțime cu mai mult de 1 mm primesc acelaşi număr de identificare. Din acest motiv, pentru sortare se formează clase cu aceleaşi dimensiuni. Acest lucru poate fi efectuat astfel încât lungimea unei bare să formeze o clasă.

Informațiile de elemente. La sortare se pot lua în considerare şi informațiile de element ținându-se cont de conținutul rândurilor de descriere. Elementele cu aceleaşi dimensiuni dar cu conținut diferit în unul sau mai multe câmpuri ale structurii de informații de element vor primi numere diferite de identificare.

Culoarea de afişare, prețul, numărul curent sau numărul din cadrul listei de lemne nu se pot constitui în criterii de sortare.

Numărul din lista de lemne. La generarea de numere în cadrul listei de lemne se iau în considerare şi primesc numere semnificative numai elementele care au ca tip de bară "*Stab*" şi "*Platte*", restul tipurilor de bare li se atribuie numărul zero. În structura de informații de element şi în programul de generare a planurilor acest număr se reprezintă ca liniuță ('-').

Numărul curent. Pe lângă reperele de sortare prezentate mai sus, în vederea sortării după numărul curent sunt luate în considerare şi prelucrările de bară efectuate dar şi potențiale (punct-simbolurile). La prelucrări se respectă aceleaşi limite de toleranță ca în cazul dimensiunilor elementului.

Toate elementele primesc un număr curent cu excepția celor care au ca tip de bară "element de legătură" sau "zid" care primesc numărul zero reprezentat ca liniuță ('-') în structura de informații de element şi în programul de generare a planurilor.

Ordinea numerelor. Ordinea în care sunt parcurse elementele trebuie să țină cont de numărul de tip de bară utilizat în structura informațiilor de element. Ordinea numerelor de tip de bară este prestabilită în cadrul unui fişier. Opțional, utilizatorul poate influența ordinea în care se parcurg pentru sortare elementele de tip "*Stab*" şi ca material "*Holz*", ordinea celorlalte tipuri de elemente nefiind influențabilă de utilizator.

La sortare, lista de numere se parcurge de sus în jos. Întâi vor fi numerotate acele elemente pentru care numărul din denumirea de element corespunde numărului din primul rând al fişierului. În continuare se sortează acele elemente pentru care numărul din denumirea de element corespunde numărului din al doilea rând al fişierului şi aşa mai departe. Elementele care nu au în denumire un număr de tip de bară se sortează ultimele.

Dacă fişierul ce conține numerele tipurilor de bară nu există, se va genera un fişier standard. Fişierul standard are următoarea structură:

- primul rând al fişierului conține un comentariu;
- cel de-al doilea rând conține numărul de versiune;
- următoarele rânduri încep cu un număr unic de tip de bară pe trei cifre urmat de semnul egal şi un comentariu.

5.3.9.2 Liste de materiale [Sav2001]

In cadrul paragrafului se prezintă structura cutiei de dialog ca soluție optimizată de specificare a configurației listelor de materiale, precum si procesul de generare a listelor de materiale.

Cutia de dialog

Generarea listelor implică apelarea inițială a cutiei de dialog (Figura 5-37) în care se selectează lista dorită şi se introduce numărul de poziție (Figura 5-38).

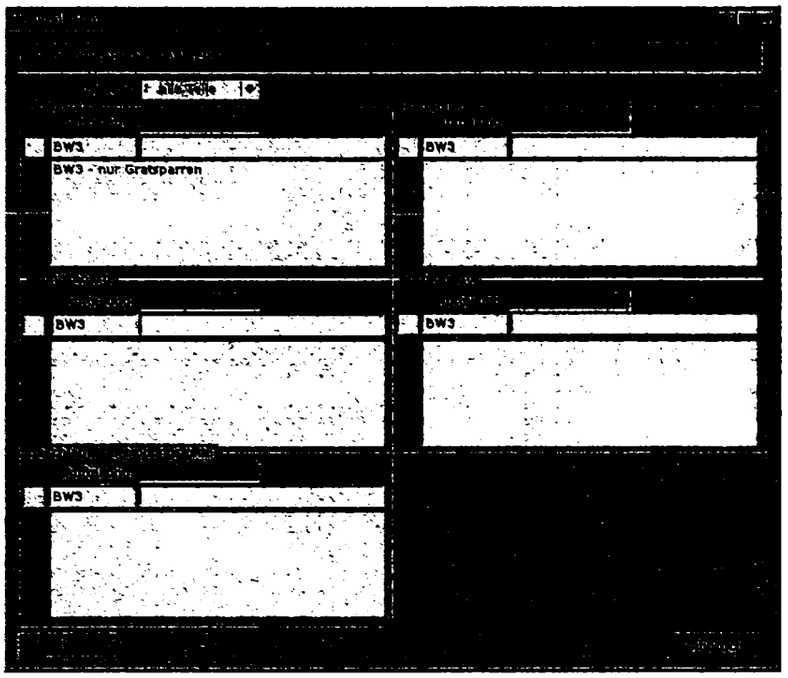


Figura 5-37. Selecția listelor de materiale

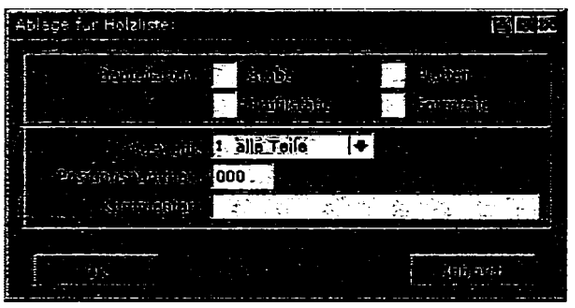


Figura 5-38. Selecția componentelor incluse în lista de materiale

Alegerea listelor generate se face printr-un buton *check-box*. La deschiderea cutiei de dialog câmpurile sunt inițializate cu numărul corespunzător poziției actuale. La introducerea primului număr de poziție, toate celelalte primesc automat acelaşi număr de poziție, existând posibilitatea modificării acestuia. In cazul în care toate numerele de poziție ale listelor selectate sunt aceleaşi, atunci se grupează în acelaşi fişier. In caz contrar, se vor genera fişiere pentru fiecare număr de poziție.

Listele de materiale

Listele de lemne

În cadrul listelor de lemne se scriu doar elementele care au ca tip de bară "*Stab*" şi ca material "*Holz*", celelalte elemente nu se iau în considerare.

Liste de bucăți (Stücklisten)

- listele încep cu un antet;
- prima linie a fiecărei pagini a listei conține textul ce apare în ultimul rând al fişierului model HOLZLIST.DAT. Antetul pentru prima pagină este exemplificat în Figura 5-39; în antetul următoarelor pagini apar, pe lângă rândul din modelul HOLZLIST.DAT, doar rândurile din Figura 5-40; în rândul "*Kommentar*" se completează textul care a fost introdus în cutia de dialog.
- în rândul "*Auftrag*" se completează numărul de proiect şi numărul de poziție al listei.
- în rândul "*Kunde*" se completează numele clientului aşa cum este el declarat pentru proiectul curent.
- în rândul "*Bauvorhaben*" se completează valoarea dată de către utilizator la crearea proiectului.

Înaintea fiecărei liste, între capul de listă şi titlul de coloană se scrie un rând cu tipul listei (Lista de plăci etc.)

Lista de plăci (*Plattenliste*)

Lista plăcilor conține toate elementele care au ca tip de bară "*Platte*". Coloanele listei

```
KOMMENTAR :
AUFTRAG         :
KUNDE           :                              SEITE:           1
BAUVORHABEN     :                              DATUM: 10.10.2005
```

Figura 5-39. Antetul primei pagini

```
AUFTRAG         :                               SEITE:           2
BAUVORHABEN     :                               DATUM: 10.10.98
```

Figura 5-40. Antetul următoarelor pagini

au structura din figura 5-41.

KOMMENTAR :

AUFTRAG :

KUNDE : SEITE: 1

BAUVORHABEN : DATUM: 10.10.2005

Figura 5-41. Antetul primei pagini

Semnificația fiecărei coloane este indicată în tabelul 4-2:

Tabelul 4-2 Semnificația coloanelor listei de plăci

Titlu Coloană	Semnificație
Nr.:	numărul curent al rândului (4).
Pos.:	numărul curent pe proiect (4).
Anz.:	numărul de plăci ale căror grosime, lățime şi lungime sunt egale (4).
Art.Nr.:	textul care apare în structura informațiilor de volum în câmpul cu numărul de articol (14).
Bezeichnung:	textul care apare în structura informațiilor de volum în câmpul descriere (18)
Dicke:	valoarea care apare în structura informațiilor de volum în câmpul corespunzător lățimii (5,1).
Breite:	valoarea care apare în structura informațiilor de volum în câmpul corespunzător înălțimii (5,3).
Länge:	valoarea care apare în structura informațiilor de volum în câmpul pentru lungime (5,3).
Fläche:	produsul dintre lățime, lungime şi număr (5,2).

Prima valoare din paranteze indică numărul de caractere. A doua valoare din paranteze, dacă există, indică numărul de zecimale.

Lista de garnituri (*Beschlagliste*)

Volumele care au ca tip de bară "*Beschlag*" se scriu într-o listă de garnituri.

Coloanele acestei liste au structura din Figura 5-42.

Conținutul coloanelor este descris în tabelul 4-3, valoarea din paranteză indicând numărul de caractere.

Nr.	Pos.	Anz.	Art.Nr	Bezeichnung	W/M

Figura 5-42. Coloanele listei de garnituri

Tabelul 4-3 Semnificația coloanelor listei de garnituri

Titlu Coloană	Semnificație
Nr.:	numărul curent al rândului (5).
Pos.:	numărul curent din DICAM (5).
Anz.:	numărul de Volume identice (5).
Art.Nr.:	textul care apare în structura de informații de volum în câmpul corespunzător numărului articolului (14).
Bezeichnung:	textul care apare în structura de informații de volum în câmpul de descriere (31).
W/M:	dacă într-un rând de descriere din cadrul structurii informațiilor de volum apare şirul "W/M=0", atunci se scrie "W", dacă apare şirul "W/M=1" se scrie "M"(5).

Lista de elemente de legătură

În această listă se trec rezultatele evaluării şirurilor de descriere a elementelor tip de legătură. Structura coloanelor listei este cea din Figura 5-43.

Nr.	Pos.	Anz.	Art.Nr	Bezeichnung	W/M	Durchm.	Länge
						in mm	in mm

Figura 5-43. Coloanele listei de legătură

Conținutul coloanelor este descris în tabelul 4-4. Prima valoare din paranteze reprezintă numărul de caractere. Cea de-a doua valoare (dacă există) reprezintă numărul de poziții zecimale).

Tabelul 4-4 Semnificația coloanelor listei de elemente de legătură

Titlu Coloană	Semnificație
Nr.:	numărul curent al rândului (4).
Pos.	această coloană rămâne goală (4).
Anz.:	numărul de volume identice (4).

Art.Nr.:	şirul care apare în parametrul corespunzător din informația de volum (14).
Bezeichnung:	şirul care apare în parametrul corespunzător din informația de volum (20).
W/M:	dacă într-un şir de descriere din structura informației de volum apare şirul "W/M=0", atunci se scrie "W", dacă apare şirul "W/M=1" se trece "M" (5).
Durchmesser:	şirul care apare în parametrul corespunzător în informația de volum (4,1).
Länge:	distanța dintre punctele P1 şi P2 (5,1).

Lista componentelor din oțel (*Stahlteilliste*)

În această listă se vor trece toate elementele pentru care tipul de bară este "*Stahlteil*". Structura coloanelor listei este prezentată în Figura 5-44.

Nr.	Pos	Anz.	Art.Nr	Bezeichnung	Güte/Schn.	W/M	L	B	H
							in mm	in mm	in mm

Figura 5-44. Coloanele listei componentelor de oțel

Conținutul coloanelor este descris în tabelul 4-5. Coloanele au următorul conținut: prima valoare din paranteză indică numărul de caractere, a doua valoare din paranteză (dacă există) indică numărul de cifre zecimale.

Tabelul 4-5 Semnificația coloanelor listei componentelor de oțel

Titlu Coloană	Semnificație
Nr.:	numărul curent al rândului (4).
Pos.:	numărul curent din DICAM (4).
Anz.:	numărul de volume identice (4).
Art.Nr.:	textul care apare în structura informației de volum în câmpul câmpul corespunzător (14).
Bezeichnung:	textul care apare în structura informației de volum în câmpul de descriere (18).
Güte/Schn.:	textul care apare în structura informației de volum în câmpul calitate/tip de tăiere (5).

W/M:	dacă într-un rând de descriere din structura informației de volum apare şirul "W/M=0", atunci se trece "W", dacă apare şirul "W/M=1" se trece "M" (5).
Länge:	valoarea care apare în structura informației de volum în câmpul corespunzător lungimii (7,1).
Breite:	valoarea care apare în structura informației de volum în câmpul corespunzător lățimii (5,1).
Höhe:	valoarea care apare în structura informației de volum în câmpul corespunzător înălțimii (5,1).

5.4. Selecție standard de obiecte [Sav2004b] [Sav2005b]

5.4.1. Definirea parametrilor funcției şi a procesului de selecție

Având în vedere faptul că, în cadrul sistemului Dietrich's, există diferite operații ce necesită selecție de obiecte se impune o uniformizare prin standardizare a procesului de selecție.

In continuare se prezintă o abordare orientată pe obiecte prin utilizarea *template*-urilor (v. paragraful 4.2.1.6.1) ca o soluție originală de dezvoltare a unei funcții de selecție de obiecte [Sav2004b] [Sav2005b].

Parametrii funcției sunt următorii:

- **`SelectionMode`**
- **`ObjectSelectionPoolGroup1`**
- **`ObjectSelectionPoolGroup2`**
- **`Repeat`**
- **`MessageSelectingGroup1`**
- **`MessageSelectingGroup2`**
- **`MessageConfirmSelectionGroup1`**
- **`MessageConfirmSelectionGroup2`**
- **`SelectedObjectsGroup1`**
- **`SelectedObjectsGroup2`**

Definirea parametrilor funcției

SelectionMode. După analiza tuturor comenzilor care necesită selecția, s-au conturat patru moduri standard de selecție:

1. **SingleObject** – poate fi selectat un singur obiect care va fi returnat în SelectedObjectsGroup1;
2. **OneGroup** – pot fi selectate mai multe obiecte care vor fi returnate în SelectedGroup1;
3. **OnePair** – se selectează separat două obiecte care vor fi returnate în SelectedObjectsGroup1 şi respectiv SelectedObjectsGroup2;
4. **TwoGroups** – se selectează două grupuri de obiecte care conțin unul sau mai multe obiecte care vor fi returnate în SelectedObjectsGroup1 şi respectiv în SelectedObjectsGroup2.

ObjectSelectionPoolGroup1 şi 2. In funcție de comanda executată, doar anumite tipuri de obiecte sunt valide şi pot fi utilizate ca şi operanzi. Cele două grupe de selecție conțin toate obiectele valide pentru fiecare din paşii de selecție. Obiectele valide sunt identificate de funcția apelantă şi transmise ca parametri funcției de selecție standard. La apăsarea butonului stânga a mouse-ului pe un obiect, doar obiectele ce se regăsesc în setul de obiecte candidate vor fi oferite la confirmare.

Restart. Procesul de selecție poate fi repornit după efectuarea operației asupra obiectelor selectate.

MessageSelectingGroup1 şi 2. Aceşti parametri reprezintă mesajele ce vor fi afişate utilizatorului în fereastra de stare şi informează utilizatorul asupra modului de selecție. Există câte un mesaj pentru fiecare grup de selecție.

MessageConfirmSelectionGroup1 şi 2. Confirmarea selecției de către utilizator este utilă mai ales când în apropierea punctului indicat se află mai multe obiecte. Dacă obiectul evidențiat nu este cel dorit de utilizator, acesta îl poate refuza. Dacă utilizatorul refuză toate obiectele oferite, procesul de selecție este abandonat.

SelectedObjectsGroup1 şi 2. Obiectele selectate se colectează în primul sau în ambele seturi de obiecte selectate după cum urmează:

Mod de selecție	SelectedObjectsGroup1	SelectedObjectsGroup2
SingleObject	Obiectul selectat	Gol

OneGroup	Obiectele selectate	Gol
OnePair	Obiectule selectat din primul grup	Obiectul selectat din grupul al 2-lea
TwoGroups	Obiectele selectate din primul grup	Obiectele selectate din grupul al 2-lea

Procesul de selecție

In cazul în care este necesară selecția unor obiecte se apelează funcția de selecție standard, fiecare parametru fiind ales corespunzător.

La implementarea unei comenzi din cadrul aplicației trebuie determinat tipul de selecție. In continuare se determină toate obiectele valide din cadrul proiectului curent şi se introduc în mod corespunzător într-una din cele două liste de obiecte de selecție. De asemenea, listele ce conțin obiectele selectate trebuie golite.

La apelul unei comenzi ce necesită selecția unui grup de obiecte, obiectele active sunt automat selectate.

Procesul de selecție a fost modelat sub forma unui sistem cu evenimente discrete în capitolul 3.5.3. Pornind de la diagrama sa de tranziții de stare (fig 3-20) se proiectează diagramele logice ce sunt prezentate în continuare. Figura 5-45 conține diagrama care descrie procesul de selecție şi confirmare a primului obiect. Figura 5-46 prezintă procesul de selecție a mai multor obiecte pentru primul grup. Figura 5-47 descrie procesul de selecție a celorlalte obiecte în cazul selecției unei perechi sau a două grupuri de obiecte. Figura 5-48 prezintă diagrama care descrie procesul de selecție pentru obiectele rămase ce vor face parte din setul al doilea de obiecte selectate. Prescurtările RMB şi LMB indică apăsarea butonului drept respectiv stâng al mouse-ului.

Figura 5-49 prezintă primul pas din cadrul procesului de selecție de către utilizator a unui obiect. Obiectul deasupra căruia se află cursorul de *mouse* este evidențiat printr-o culoare diferită.

5.4.2. Implementarea clasei

In continuare se prezintă problematica, alegerea şi implementarea soluției optime în cazul utilizării claselor template respectiv implementarea propriu-zisă a clasei generale de selecție [Sav2004b][Sav2005b].

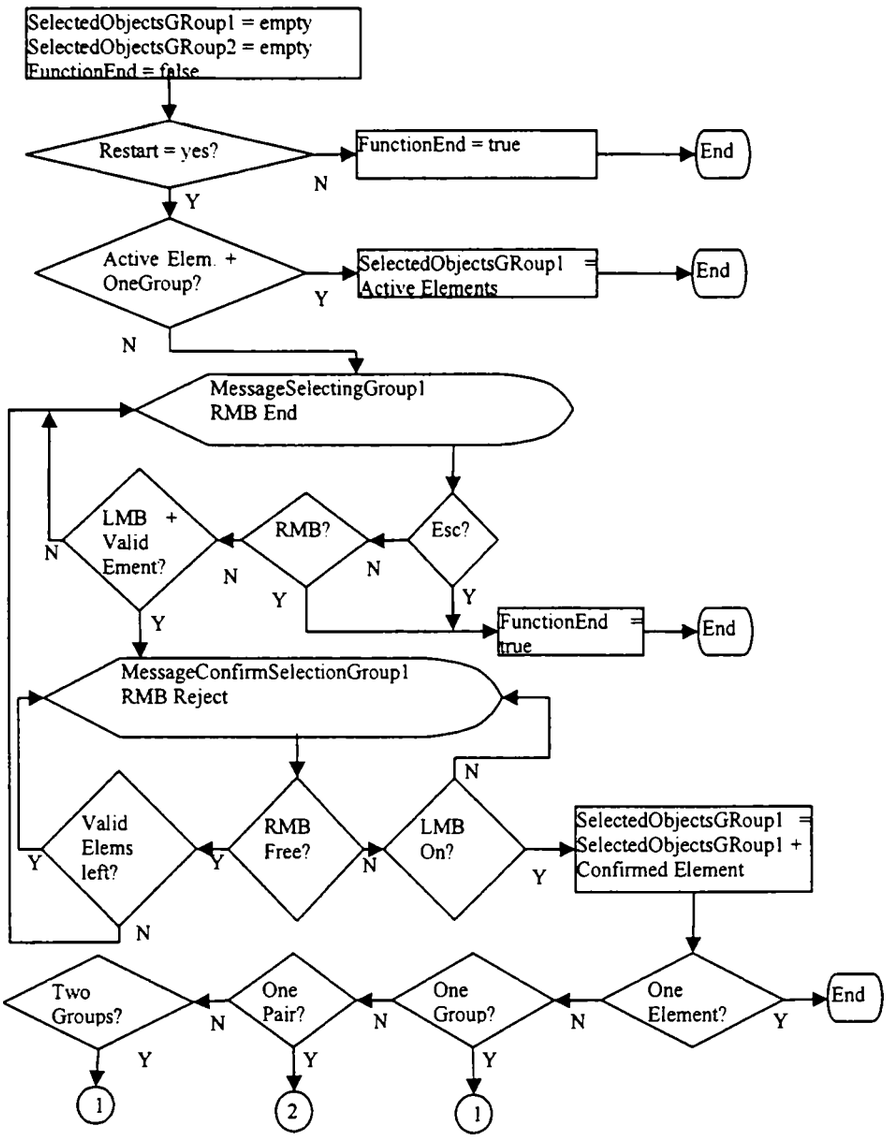


Figura 5-45. Diagrama care descrie procesul de selecție și confirmare a primului obiect

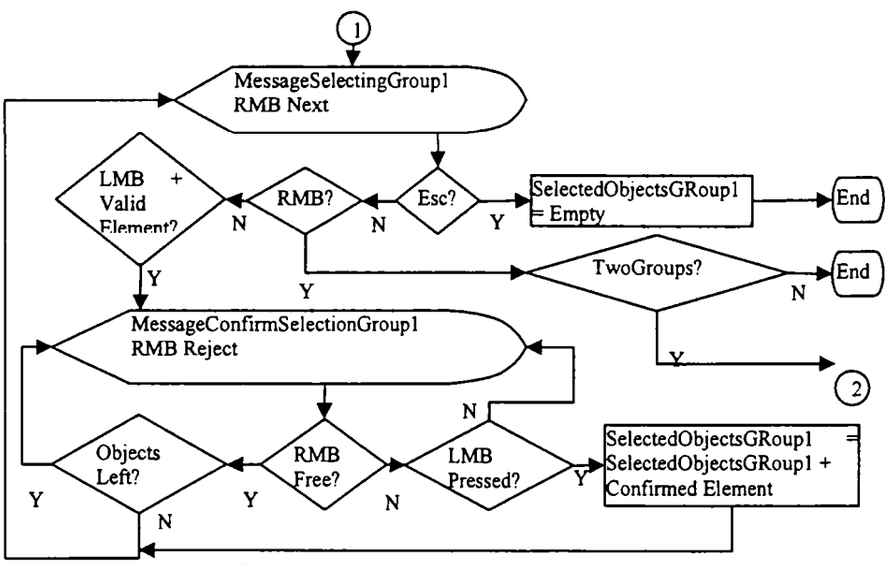


Figura 5-46. Procesul de selecție a mai multor obiecte pentru primul grup

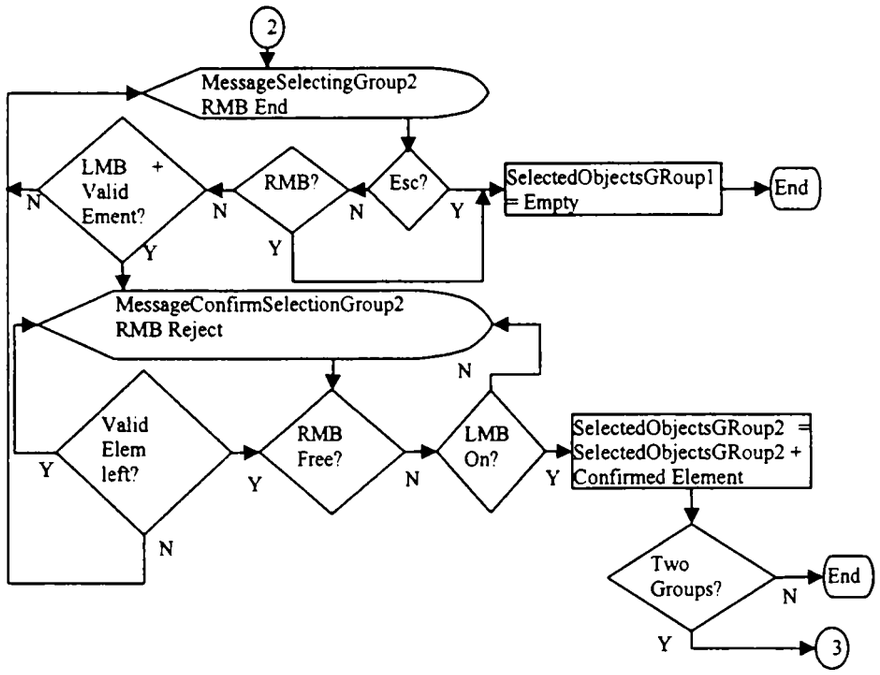


Figura 5-47. Procesul de selecție a celorlalte obiecte în cazul selecției a unei perechi sau a două grupuri de obiecte

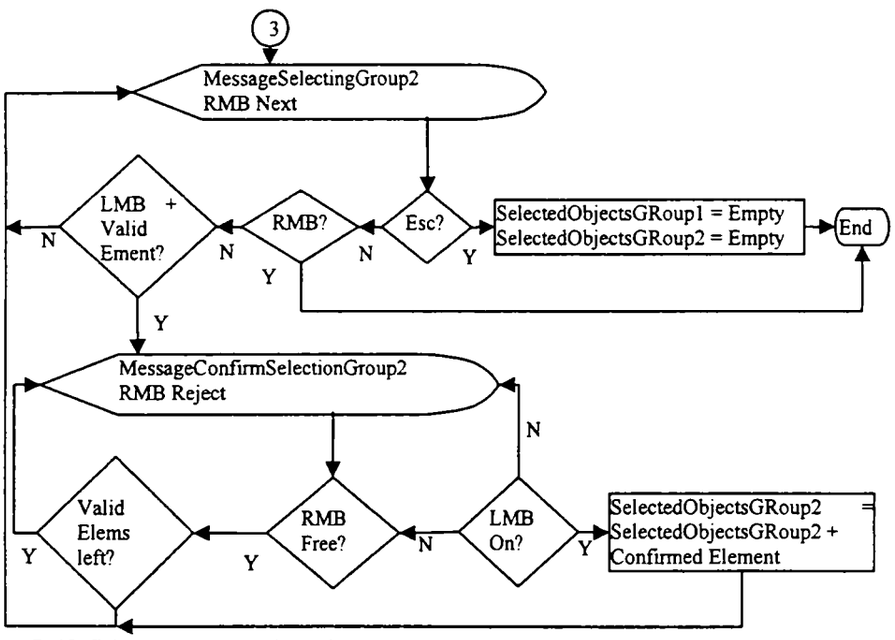


Figura 5-48. Diagrama care descrie procesul de selecție pentru obiectele rămase ce vor face parte din setul al doilea de obiecte selectate

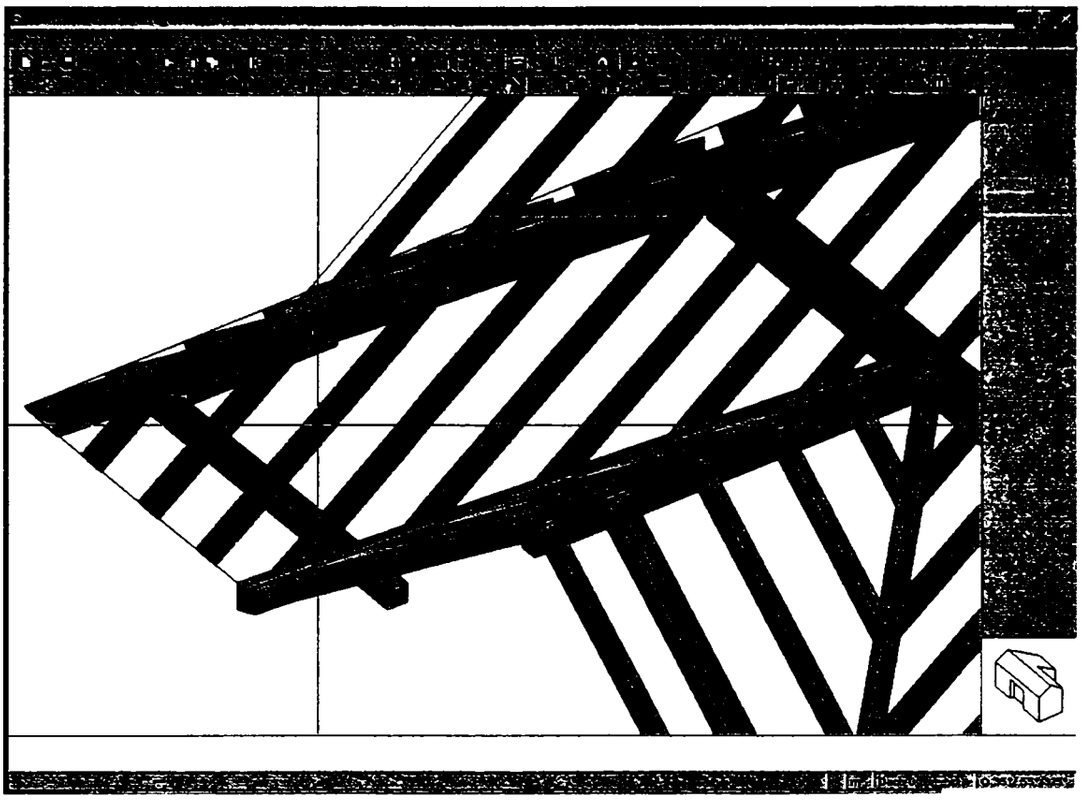


Figura 5-49 Selecția grafică a unui obiect

5.4.2.1 Clasa template CDhpInputDeviceSelectObjects

Mecanismul de selecție este implementat în C++ ca o clasă template. Ierarhia clasei template este prezentată în figura 5-50.

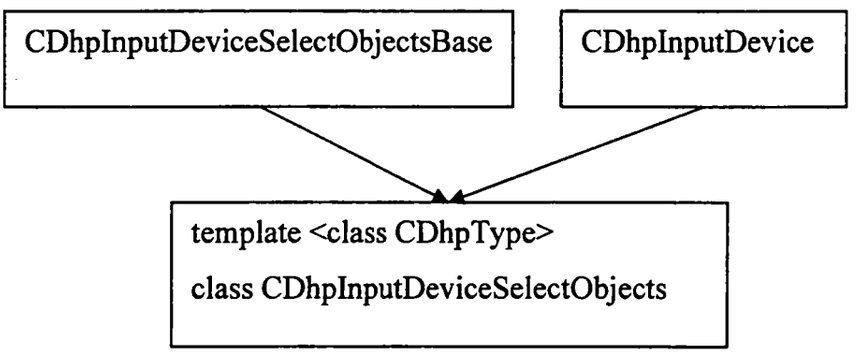


Figura 5-50. Ierarhia clasei template CDhpInputDeviceSelectObjects

Dezvoltările şi implementarea propusă a clasei de selecție în cadrul aplicației a fost realizată pe baza metodologiei sintetizate în paragrafele 4.2.1.5 şi 4.2.1.6.1.

Soluția propusă pentru instanțiere este instanțierea explicită a template-ului pentru fiecare tip necesar într-un fişier cpp separat:

```
template class CDhpInputDeviceSelectObjects<CDhpObject>;
template class CDhpInputDeviceSelectObjects<CDhpPoint>;
template class CDhpInputDeviceSelectObjects<CDhpLine>;
```

Clasa CDhpInputDevice tratează interacțiunile cu utilizatorul cum sunt mișcarea mouse-ului și clicurile de butoane.

Din motive de claritate, clasa de bază **CDhpInputDevice-SelectObjectsBase** implementează funcționalitățile ce nu utilizează **CDhpType**, cum sunt enumerațiile pentru modurile de selecție, enumerațiile pentru filtrele automate și structura de sortare a obiectelor.

```
class CDhpInputDeviceSelectObjectsBase
{
public:
  /// Cele 6 moduri de selectie
  enum
     {SELECT_OBJECT,SELECT_OBJECT1,SELECT_OBJECT2,SELECT_GROUP,SELE
CT_GROUP1,SELECT_GROUP2 };
  .
  .
  .
    /// obiectul si distanta de sortare
  struct DhpObjectAndSortDistance
    {CDhpObject *pDhpObject;
    double dSortDistance;};
  .
  .
  .
};
```

Declarația clasei template este:

template <class CDhpType> class CDhpInputDeviceSelectObjects : public CDhpInputDeviceSelectObjectsBase, public CDhpInputDevice

Constructorul clasei necesită specificarea unui set de parametri:

1. **CDhpWindow *pDhpWindow** – fereastra în care se desfășoară procesul de selecție;
2. **short *pnReticleSize** – dimensiunea zonei active din jurul cursorului de mouse, în care se iau in considerare obiectele intersectate;
3. **CDhpList <CDhpType> *pListDhpObjectsInput** – lista entităților grafice care pot fi selectate;

4. **CDhpList <CDhpType> *pListDhpObjectsOutput** – lista ce va conține toate elementele selectate;
5. **int nSelectionMode,char *lpszSelectionString** – mesajul care este afișat la începutul selecției;
6. **char *lpszConfirmationString** – mesajul care este afișat la confirmarea unui obiect pentru a fi selectat;
7. **bool bHover = false** – indică dacă se evidențiază obiectele atinse în timpul mișcării mouse-ului ;
8. **bool bToolTips = false** – activează afișarea de sugestii atunci când cursorul de mouse se oprește deasupra unei entități geometrice;
9. **int nScroll = SCROLL_NONE** – restricționează deplasarea cursorului de mouse la fereastra curentă;

Funcțiile necesare care trebuie implementate astfel încât clasa de selecție să funcționeze corect sunt următoarele:

1. **int DoModal(void)** – funcția ce gestionează interacțiunea cu utilizatorul;
2. **void BlinkSelectedObjects(void)** – clipește obiectele selectate pentru ca utilizatorul să aibă o confirmare vizuală a obiectelor indicate;
3. **void ShowObjectsToolTip(void)** – afișarea unei descrieri scurte a obiectului grafic aflat în poziția cursorului de mouse;
4. **void OnLButtonDown(UINT nFlags,POINT *pPoint)** – cadrul aplicației apelează această funcție atunci când este apăsat butonul stâng al mouse-ului;
5. **void OnRButtonDown(UINT nFlags,POINT *pPoint)** - cadrul aplicației apelează această funcție atunci când este apăsat butonul drept al mouse-ului|;
6. **void OnMouseHover(UINT nFlags,POINT *pPoint)** – cadrul aplicației apelează această funcție atunci când cursorul de mouse se oprește deasupra unui obiect;
7. **void OnMouseMove(UINT nFlags,POINT *pPoint)** – cadrul aplicației apelează această funcție la fiecare mișcare de mouse;
8. **void OnKeyDown(UINT nChar,UINT nRepCnt,UINT nFlags)** - cadrul aplicației apelează această funcție atunci când este apăsată o tastă;

9. **`void EndInputDevice(int nVirtualKey)`** – terminarea interacțiunii cu utilizatorul;
10. **`void Restart(void)`** – inițializează variabilele membre;
11. **`void OnMouseWheel(UINT nFlags, short nDelta, POINT point)`** - cadrul aplicației apelează această funcție la mișcarea rotiței mouse-ului;

Figura 5-51 prezintă colaborările dintre clase în cadrul **`CDhpInputDevice-SelectObjects`**. Se indică toate clasele utilizate ca tip pentru variabilele clasei template.

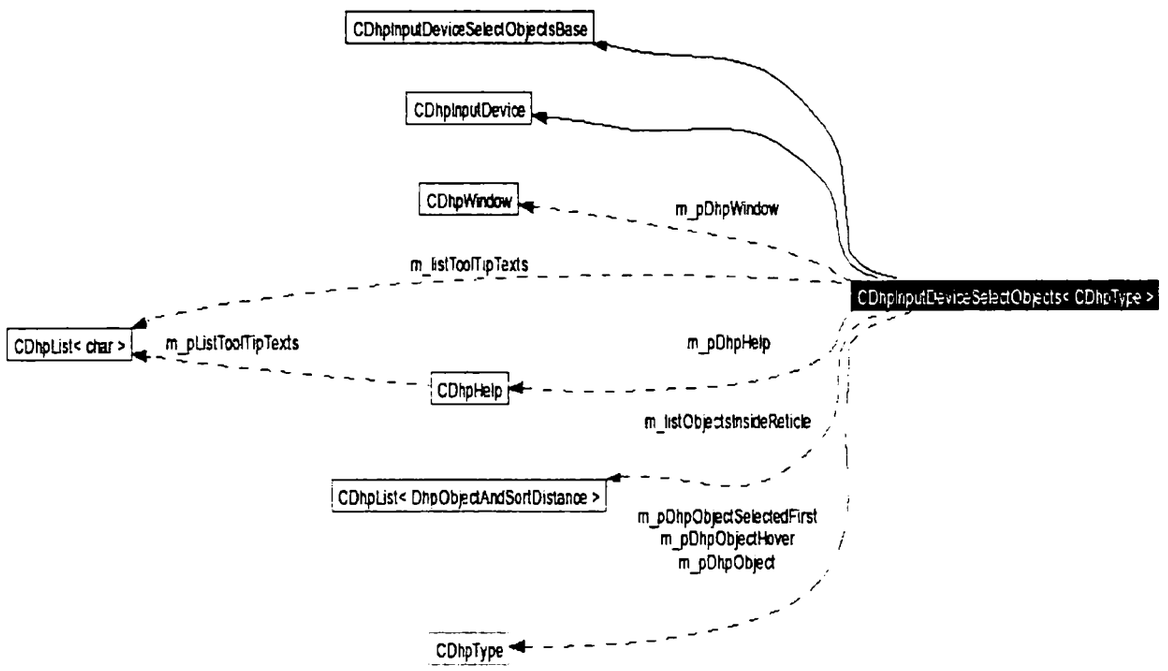


Figura 5-51. Graful colaborării între clase

5.5. Gestionare specială a cutiilor de dialog [Sav2004]

Problema care trebuie rezolvată în cadrul acestei aplicații constă în selectarea prelucrărilor de îmbinare dintr-un set de prelucrări disponibile predefinite, în cazul intersecțiilor a mai multor tipuri de bare în structura unui acoperiş.

Utilizatorul trebuie să specifice tipul prelucrării şi valorile implicite ale parametrilor lor [Phi2004].

Figura 5-52 prezintă un punct de intersecție a mai multor tipuri bare.

Pentru rezolvarea problemei au fost impuse îndeplinirea următoarelor condiții:

- abordarea orientată pe obiecte;
- reutilizarea codului

- toate cutiile de dialog trebuie să se conformeze cu specificațiile de proiectare a interfeței specifice sistemului Dietrich's
- fiecărui caz trebuie să îi corespundă o cutie de dialog în care utilizatorul să poată alege prelucrarea implicită.

Figura 5-53. Punct de intersecție a mai multor tipuri de bare.

Fiecare prelucrare are asociate o cutie de dialog şi o clasă C++ proiectate pentru a gestiona parametrii săi. Datorită faptului că parametrii prelucrării pot fi modificați în mai multe locuri, se doreşte reutilizarea resurselor existente.

La schimbarea prelucrării în cadrul cutiei corespunzătoare unui caz de intersecție, conținutul acesteia se modifică parțial pentru a afişa parametri corespunzători noii prelucrări selectate. Opțiunile de implementare sunt:

1. realizarea unei cutii de dialog noi pentru fiecare caz de intersecție;
2. utilizarea unei cutii de dialog pentru fiecare prelucrare în parte.

Cea de-a doua opțiune este cea dorită a fi implementată întrucât codul utilizat în cazul acesteia există deja şi, cu modificări minore, poate fi reutilizat accelerând procesul de implementare.

Pe lângă parametri de prelucrare, noile cutii de dialog mai trebuie să conțină:

1. un set de check-box-uri de selecție a prelucrării implicite;
2. un set de butoane prin care se selectează prelucrarea a cărei parametri se modifică.

In continuare se prezintă descrierea vizuală şi comportamentală a cutiilor de dialog precum şi modul de implementare a funcționalităților acestora.

Descrierea cutiilor de dialog

O cutie tipică de gestionare a unui caz de intersecție este prezentată în figura 5-53.

In partea stângă a cutiei se află o coloană de check-box-uri dintre care cel puțin unul trebuie selectat şi respectiv o coloană de butoane dintre care unul va trebui să fie apăsat. Check-box-ul selectat va indica prelucrarea implicită utilizată în cazul dat iar butonul apăsat va rămâne în aceasta stare pentru a indica tipul prelucrării căreia îi aparțin parametrii afişați.

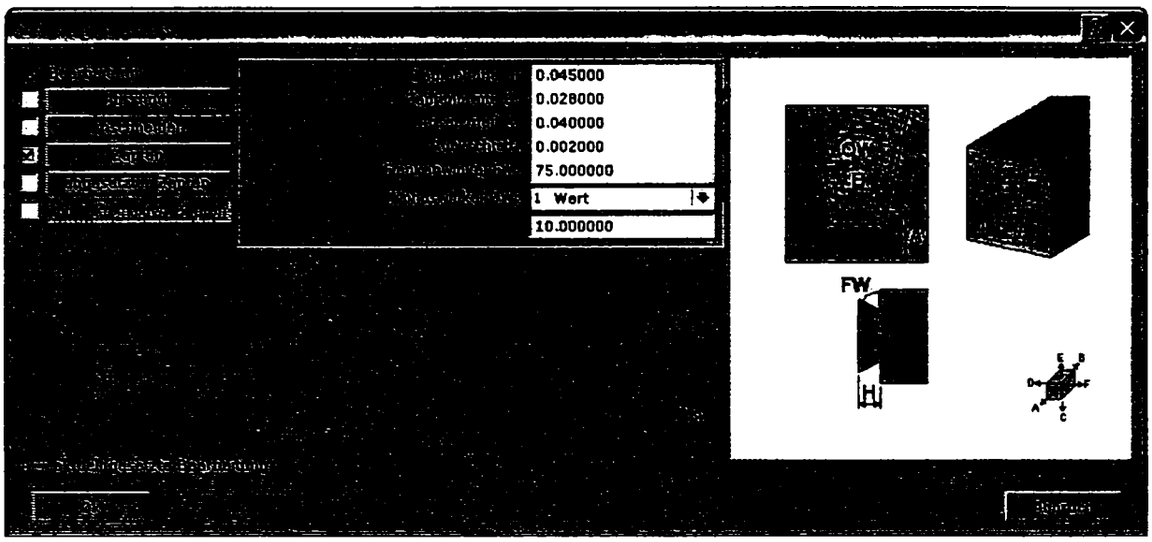


Figura 5-53. Cutie tipică de gestionare a unui caz de intersecție

Implementarea

Pornind de la diagrama tranzițiilor de stare a modelului funcției realizată conform 3.5, s-a proiectat o funcție principală cu menirea de a gestiona trecerea de la o cutie de dialog la alta, având următoarele obiective:

1. crearea listei de cutii de dialog corespunzătoare prelucrărilor posibile pentru cazul de intersecție curent;
2. identificarea cutiei de dialog corespunzătoare prelucrării implicite pentru a fi afişată prima;
3. utilizarea unei bucle condiționale, de interpretare a codului returnat la închiderea fiecărei cutii.

La apăsarea de către utilizator a butoanelor de prelucrare, a butonului OK şi al celui de Revocare, cutia se închide returnând identificatorul (ID-ul) corespunzător butonului apăsat. Dacă a fost apăsat unul din butoanele de selecție a unei prelucrări, funcția de gestiune deschide cutia de dialog corespunzătoare. Dacă s-a apăsat butonul de Revocare, funcția de gestiune va fi părăsită fără a salva modificările. La apăsarea butonului OK funcția de gestiune setează prelucrarea implicită pentru cazul curent precum şi parametri prelucrării.

Figura 5-54 prezintă diagrama de operare a funcției de gestiune a cutiilor de dialog de caz.

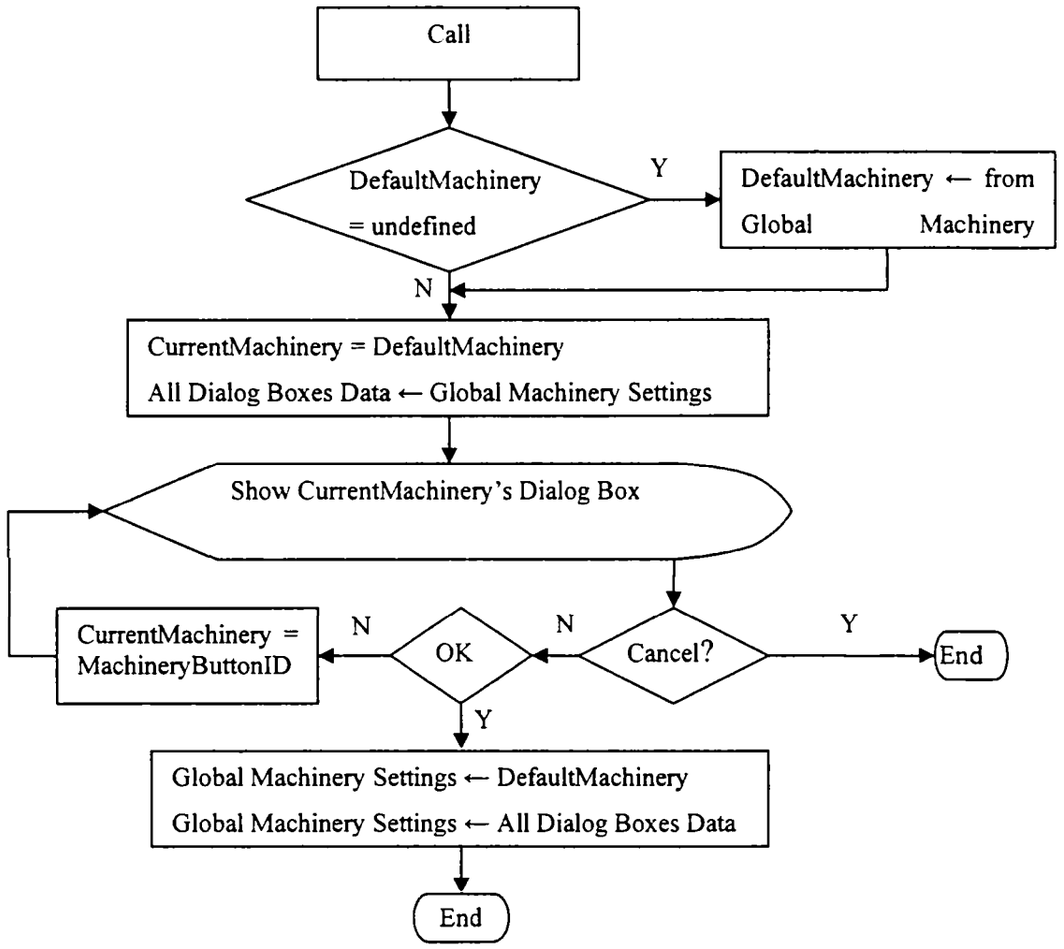


Figura 5-54. Diagrama de operare a funcției de gestiune a cutiilor de dialog de caz

DefautMachinery reprezintă prelucrarea al cărei check-box va fi selectat. CurrentMachinery reprezintă prelucrarea a cărei cutii de dialog este afișată. Crearea listei cutiilor de dialog pentru prelucrările de caz este realizată de către o funcție specială. In funcție de cazul curent, aceasta va completa o listă cu cutiile de dialog corespunzătoare. Structura utilizată conține următoarele informații:

3. ID-ul cutiei de dialog;
4. Titlul cutiei;
5. Parametri prelucrării.

Utilizarea unei copii a structurii de parametri ai unei prelucrări permite păstrarea valorilor originale până la apăsarea butonului OK.

Figura 5-55 prezintă structura de derivare a clasei cutiei de dialog de caz prin derivare multiplă.

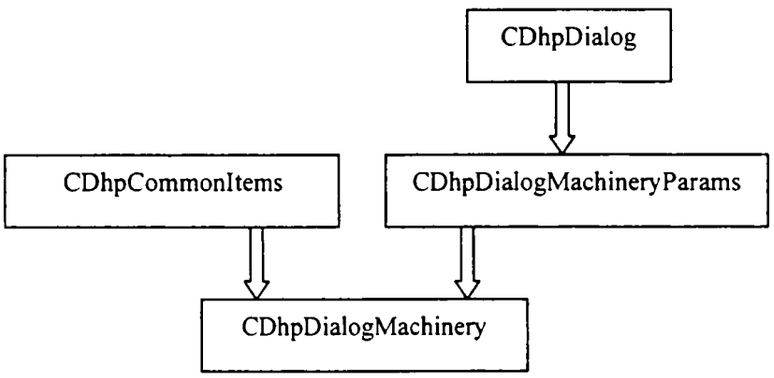


Figura 5-55. Structura de derivare a clasei cutiei de dialog de caz prin derivare multiplă

Datorită faptului că check-box-urile şi butoanele din partea stângă sunt aceleaşi pentru toate cutiile, s-a proiectat şi implementat o clasă unică de gestionare a acestora. Astfel, toate cutiile de dialog sunt multiplu derivate din fiecare clasă de gestiune a prelucrării precum şi din aceeaşi clasă de gestiune a selecției prelucrărilor.

Pentru o funcționare corectă a clasei cutiei de dialog, funcțiile membre ce gestionează mesajele standard de fereastră trebuie să apeleze la rândul lor funcția corespunzătoare din clasele de bază. De exemplu, **`CDhpDialogMachinery::OnInitDialog`** trebuie să apeleze funcțiile **`CDhpCommonItems::OnInitDialog`** şi **`CDhpDialogMachineryParameters::OnInitParameters`**. Asemănător se procedează în cazul următoarelor funcții:

1. **`DoDataExchange`** – care gestionează sincronizarea elementelor de interfață cu parametri asociați;
2. **`OnEN_CHANGE`** – care este apelată la modificarea unei selecții în cadrul unui droplist;
3. **`OnBN_CLICKED`** – care este apelată la apăsarea unui buton.

5.6. Vizualizarea 3D a modelelor utilizând OpenGL

Modelele geometrice 3D proiectate de utilizator în cadrul aplicației sunt afişate pe ecran utilizând un set de funcții de desenare speciale, proprietare, ce desenează muchiile modelelor sub formă de linii (*wireframe*). Singura facilitate pe care utilizatorul o are la dispoziție pentru a înțelege mai bine tridimensionalitatea modelelor este reprezentarea cu linii ascunse. Acest mod de vizualizare este util mai ales în cazul proiectării modelelor pentru că permite vizualizarea modelelor în întreaga lor complexitate. Pentru ca utilizatorul să poată

percepe vizual modelele proiectate, s-a apelat la biblioteca grafică OpenGL pentru vizualizarea foto-realistă a modelelor.

Obiectivul principal al implementării vizualizării OpenGL este să permită utilizatorului să lucreze în vizualizarea OpenGL în modul clasic obişnuit, în maniera specifică aplicației. In cadrul vederii OpenGL, utilizatorul poate vizualiza modelele geometrice şi efectua anumite operații asupra acestora similar modului nativ de vizualizare. Integrarea vizualizării OpenGL s-a realizat într-o asemenea manieră încât codul existent să sufere modificări minime pentru a funcționa corect. Implementarea vizualizării 3D a modelelor s-a realizat prin proiectarea unei clase, **`CDhpOpenGLWorkspace`**, care realizează la rândul ei toate operațiile necesare. De asemenea s-au proiectat o serie de structuri de date pentru păstrarea informației de model într-o formă optimizată pentru utilizarea în OpenGL.

In paragraful 5.6.1 sunt enunțate şi analizate principalele clase şi structuri de date create. Paragrafele 5.6.2 şi 5.6.3 formulează modurile de implementare a operațiilor de selecție de obiecte şi respectiv de evidențiere a obiectului selectat. Paragraful 5.6.4 abordează implementarea vizualizării modelelor cu linii ascunse. In final, descrierea implementării procesului de triangularizare este tratată în paragraful 5.6.5.

5.6.1. Structuri şi clase

Figura 5-56 prezintă diagrama de colaborări a clasei **`CDhpOpenGLWorkspace`**.

`CDhpOpenGLWorkspace` implementează următoarele facilități:

- deplasarea şi rotirea scenei;
- mărirea şi micşorarea desenului;
- multiple moduri de vizualizare a modelelor: cu linii ascunse, ca şi solide fără textură, cu textură şi cu textură şi muchii. In plus, toate corpurile pot fi desenate translucent;
- selecție de elemente grafice;
- evidențiere de elemente grafice;
- salvare a vederii sau a unei porțiuni ca fişier imagine in format PNG;
- tipărire a vederii cu rezoluție înaltă pentru o imprimare fidelă;
- configurarea paşilor de deplasare, de rotire şi a unghiului de vedere.

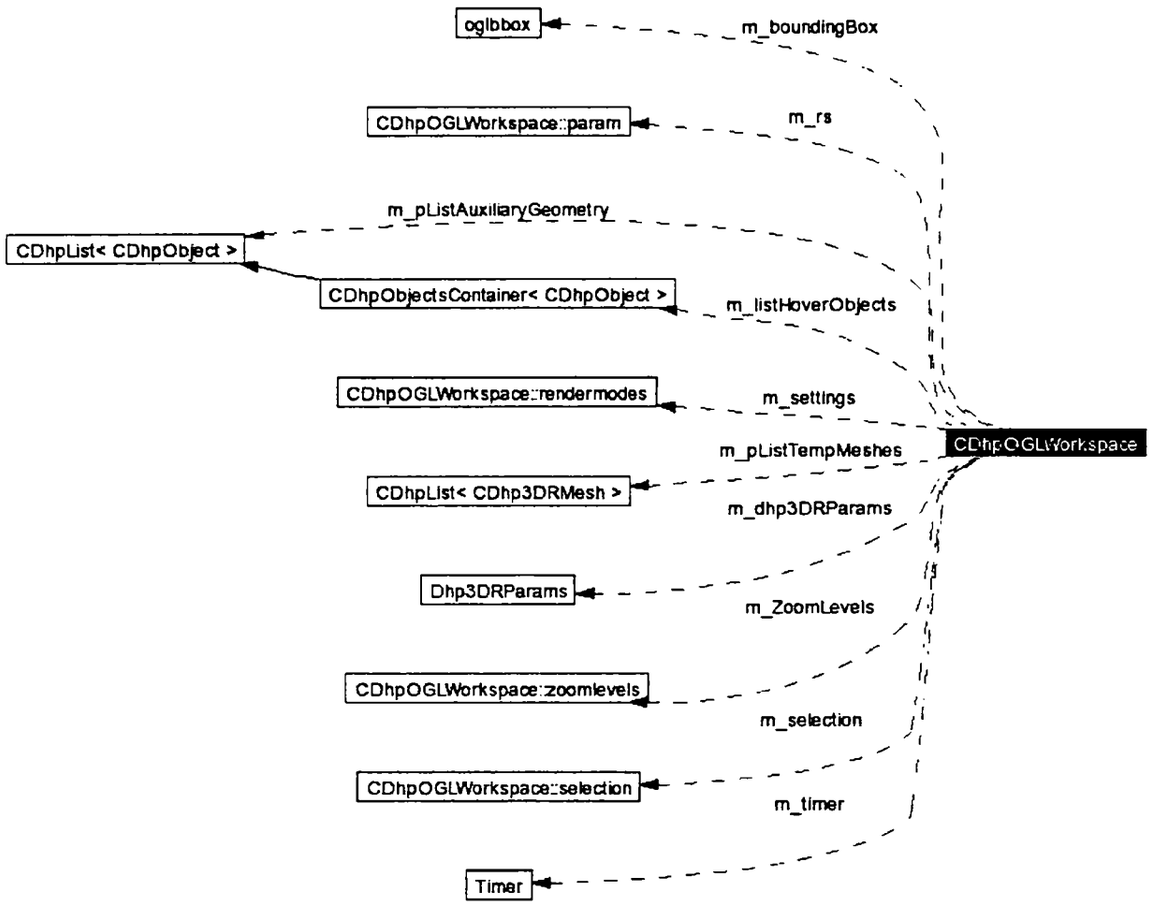


Figura 5-56. Diagrama de colaborări a clasei `CDhpOpenGLWorkspace`.

Informația geometrică optimizată pentru vizualizare în OpenGL se află în clasele **`CDhp3DRMesh`** şi **`CDhp3DRFace`**. Acestea sunt menite păstrării informațiilor despre geometria modelului. Figura 5-57 prezintă structura clasei **`CDhp3DRMesh`** iar figura 5-58 reprezintă diagrama de colaborări pentru aceeaşi clasă.

Clasa **`CDhp3DRMesh`** conține variabilele membre:

- **`m_nId`**, identificator de obiect utilizat în modul de selecție;
- **`m_listDhp3DRFaces`** de tip **`CDhp3DRFace`**, listă de fețe care reprezintă poligoanele de desenat;
- **`m_listLines`**, listă de linii utilizată la reprezentarea *wireframe* atunci când obiectul este unul de vizualizare;
- **`m_sTextureInfo`**, informații legate de texturi;
- **`m_sColor`**, culoare utilizată la vizualizarea fără texturi;
- **`m_nTranslucentFactor`**, gradul de translucență al obiectului atunci când modul de vizualizare corespunzător este activ.

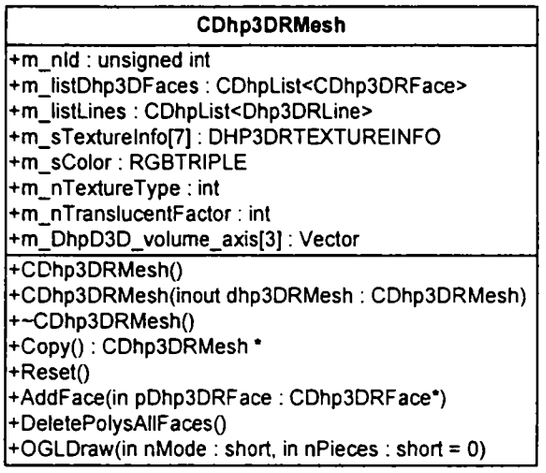


Figura 5-57. Structura clasei `CDhp3DRMesh`

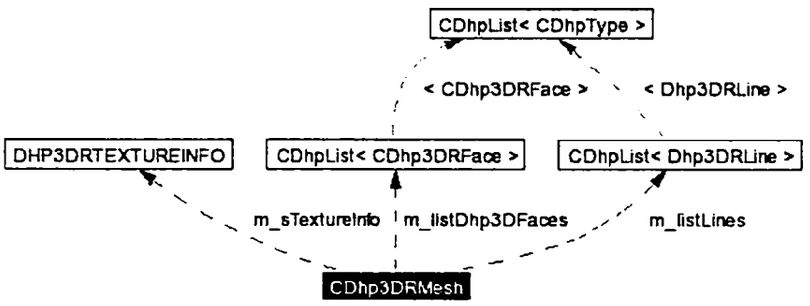


Figura 5-58. Diagrama de colaborări a clasei `CDhp3DRMesh`

Metodele clasei sunt următoarele:

- **`AddFace()`** adaugă o față;
- **`DeletePolysAllFaces()`** şterge poligoanele calculate pentru fiecare față;
- **`OGLDraw()`** emite comenzile OpenGL pentru desenarea modelului.

Figura 5-59 prezintă structura clasei **`CDhp3DRFace`** iar figura 5-60 reprezintă diagrama de colaborări a clasei **`CDhp3DRFace`**.

Variabilele membre sunt:

- **`normal`**, normala planului feței;
- **`vertices`**, punctele care formează poligoanele feței;
- **`polygeometry`**, de tip **`Dhp3DRGeometry`**, conține primitive grafice OpenGL;
- **`m_lpstrTextureFile`**, numele fişierului pentru textura feței;
- **`m_sRGBTriple`**, culoarea feței.

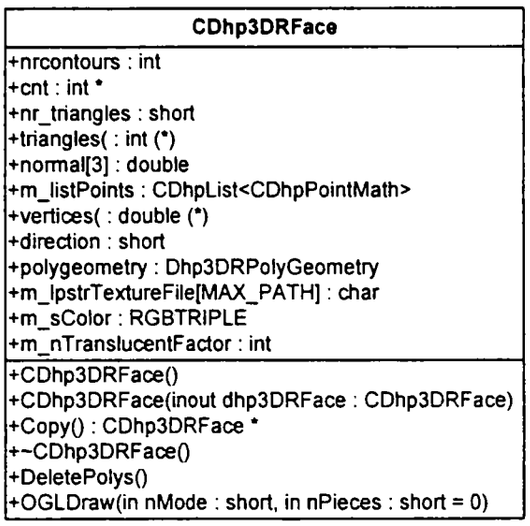


Figura 5-59. Structura clasei `CDhp3DRFace`

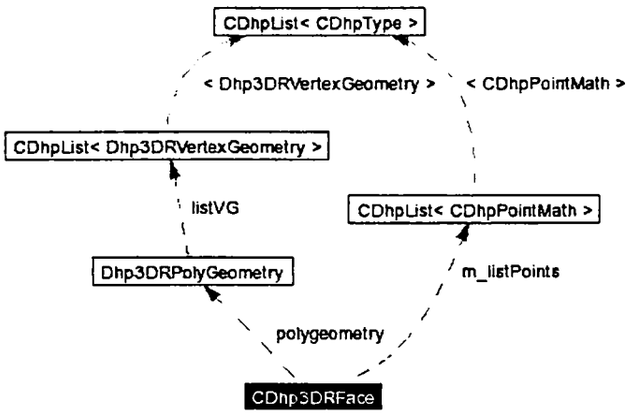


Figura 5-60. Diagrama de colaborări a clasei `CDhp3DRFace`

Metodele clasei sunt:

- **`DeletePolys()`** șterge poligoanele OpenGL;
- **`OGLDraw()`** emite comenzile OpenGL pentru desenarea feței.

Structura **`Dhp3DRGeometry`** conține primitivele grafice OpenGL obținute prin triunghiularizarea poligoanelor fețelor modelelor.

In cadrul aplicației, geometria modelelor este păstrată într-o structură nativă, optimizată pentru utilizarea în cadrul funcțiilor de modelare şi reprezentare. Clasa de reprezentare a modelului conține şi o variabilă membră de tip **`CDhp3DRMesh`** care păstrează informația optimizată pentru vizualizarea OpenGL. Modelele trebuie procesate înainte de a fi posibilă reprezentarea. Acest lucru se realizează prin apelul unei funcții de procesare,

MakeMesh(), care realizează procesarea în doi paşi. În primul pas se obțin toate fețele modelului sub formă de poligoanele cu găuri. Pentru acest pas s-au creat clasele de procesare **CDhp3DRMeshFactory** şi **CDhp3DRFaceFactory** ce realizează conversia. În cel de-al doilea pas se triunghiularizează fiecare poligon completându-se structura **polygeometry**.

In continuare se prezintă câteva probleme specifice ce au necesitat o atenție deosebită în implementare.

5.6.2. Selecția de obiecte

In cadrul procesului de selecție a modelelor descris în secțiunea 4.4, în funcție de modul de vizualizare s-au determinat operațiile ce trebuie tratate într-un mod specific OpenGL şi anume:

- procesul de indicare cu ajutorul mouse-ului a obiectelor pe ecran;
- evidențierea unui obiect prin clipire.

Inițial, clasa de selecție a fost implementată utilizând modul de vizualizare clasic. In urma analizei clasei de selecție s-au determinat situațiile care necesită modificări pentru a funcționa corect şi în cazul vizualizării OpenGL:

- în funcția **OnLButtonDown**, având coordonatele punctului de clic de mouse în fereastra curentă, se identifică obiectele aflate în zona activă a reticulului de mouse;
- în funcția **BlinkSelectedObjects**, evidențierea obiectelor selectate prin clipire trebuie realizată în OpenGL.

In cadrul funcției **OnLButtonDown**, dacă vizualizarea este OpenGL se apelează o funcție specifică de determinare a obiectelor din apropierea cursorului de mouse. Declarația funcției este următoarea:

```
bool GetObjectsInReticleFromOpenGLWorkspace(
  POINT ptMouse,
  CDhpList<CDhpObject> *pListDhpObjectsInput,
  CDhpList<CDhpObject> *pListObjectsInReticle,
  bool bGetChildren);
```

unde parametrii au următoarea semnificație:

- **ptMouse** conține coordonatele de fereastră ale punctului de clic de mouse;

- **pListDhpObjectsInput** conține lista obiectelor candidate la selecție;
- **pListObjectsInReticle** după apelul funcției va conține lista obiectelor aflate în vecinătatea punctului dat de ptMouse;
- **bGetChildren** este utilizat în situația în care se dorește selecția unui capăt de linie.

Paşii urmați in cadrul funcției specifice sunt:

1. Se selectează contextul de randare (RC) (v. paragraful 4.4.1.1):

```
ret = wglMakeCurrent(m_hDC, m_hGLRC);
```

2. Se înlocuiește lista globală de obiecte de desenat cu lista obiectelor candidate.

```
m_pListAuxiliaryGeometry = pListDhpObjectsInput;
if(!m_pListAuxiliaryGeometry)
  m_pListAuxiliaryGeometry = &m_listHoverObjects;
```

3. se alocă şi se selectează *buffer*-ul de selecție:

```
UINT *pSelectBuffer = new UINT[SIZE_FBUFFER];
glSelectBuffer(SIZE_FBUFFER, pSelectBuffer);
```

4. Se intră în modul de selecție:

```
glRenderMode(GL_SELECT);
```

5. Se setează vederea:

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPickMatrix((double)ptMouse.x,
              viewport[3]-(double)ptMouse.y,
              fangradius, fangradius, viewport);
GLdouble znear = 0.1F;
GLdouble zfar =  500.0F;
GLdouble gldAspect = (GLdouble) (viewport[2] - viewport[0]) /
```

```
                        (GLdouble) (viewport[3] - viewport[1]);
  //GLdouble gldAspect = (GLdouble) viewport[2] /
                         (GLdouble) viewport[3];
  if(m_settings.projectionMode == pmPerspective)
    gluPerspective(m_settings.dFOV / gldAspect,
                   gldAspect, znear, zfar);
  if(m_settings.projectionMode == pmOrtho)
  {
    GLdouble left, right, top, bottom;
    right = znear * tan(m_settings.dFOV/2.0F * PI_DIV_180);
    top = right / gldAspect;
    bottom = -top;
    left = -right;
    glOrtho (left*m_rs.ts.ZoomRatio, right*m_rs.ts.ZoomRatio,
             bottom*m_rs.ts.ZoomRatio,
             top*m_rs.ts.ZoomRatio, znear, zfar);
  }
```

6. Se inițializează stiva de nume.

7. Se apelează funcția de desenare (aceasta nu produce desenare pe ecran):

```
  DrawScene((short)bGetChildren);
```

Transmiterea parametrului **bGetChildren** permite determinarea capătului de linie cel mai apropiat de punctul de clic de mouse. Acest lucru se realizează prin divizarea liniei în două sub-segmente notate 1 şi 2. Prin urmare, înregistrarea de pe stiva de selecție va conține două nume, primul identifică linia selectată iar al doilea capătul cel mai apropiat.

8. Se revine în modul de randare obținându-se astfel numărul de selecții din *buffer*-ul de selecție:

```
  size = glRenderMode(GL_RENDER);
```

9. Se analizează *buffer*-ul de selecție construindu-se o listă ce conține fiecare element selectat. Elementele sunt ordonate în listă, în funcție de distanța față de ochiul utilizatorului, de la cel mai apropiat la cel mai îndepărtat;

10. Se eliberează RC-ul:

```
wglMakeCurrent(NULL, NULL) ;
```

11. Se parcurge lista de elemente selectate determinând tipul fiecărui element şi adăugându-l la lista **pListObjectsInReticle**. O tratare specială se efectuează dacă se selectează un capăt de linie (**bGetChildren == true**). In funcție de sub-segmentul selectat se determină capătul de linie corespunzător:

```
Vector p1, p2;
((CDhpLine*)vs->obj)->GetLine(p1,p2);
CDhpPoint *pt = new CDhpPoint;
if(vs->endp == 1)
  pt->SetPoint(p1[0], p1[1], p1[2]);
else
  pt->SetPoint(p2[0], p2[1], p2[2]);
pt->SetPersistent(false);
pDhpObj = pt;
```

In continuare se verifică dacă punctul determinat nu există deja în lista de elemente selectate. Dacă punctul de clic se află pe linie, dar nici unul din capete nu se află în reticul, atunci acesta nu este luat în considerare.

Pentru a obține coordonatele pixelului corespunzător unui punct 3D se poate utiliza funcția ajutătoare **gluProject**:

```
int gluProject(
  GLdouble objy,
  GLdouble objy,
  GLdouble objy,
  const GLdouble modelMatrix[16],
  const GLdouble projMatrix[16],
  const GLint viewport[4],
  GLdouble winx,
  GLdouble winy,
  GLdouble winz);
```

Funcția permite această determinare fără să fie nevoie de utilizarea modului de randare *feedback*.

12. Se dealocă *buffer*-ul de selecție şi se părăseşte funcția.

5.6.3. Evidențierea obiectelor

Pentru realizarea unui feedback vizual către utilizator cu obiectul pe care acesta l-a selectat, este necesară elaborarea unei metode de evidențiere a obiectelor similare cu cea din vizualizarea clasică.

In cazul vizualizării clasice, evidențierea unui obiect se realizează alternând desenarea obiectului între două culori. Principalul obiectiv al paragrafului curent este implementarea unei manifestări identice şi în vizualizarea OpenGL (figura 5-61). Obiectivul secundar îl reprezintă concepția şi implementarea unei soluții pentru problema prezentată în încheierea paragrafului 4.4.1.

Prin urmare, pentru vizualizarea OpenGL s-a proiectat funcția:

```
bool BlinkObjectsInOpenGLWorkspace(
  CDhpObject *pObj,
  COLORREF nColor);
```

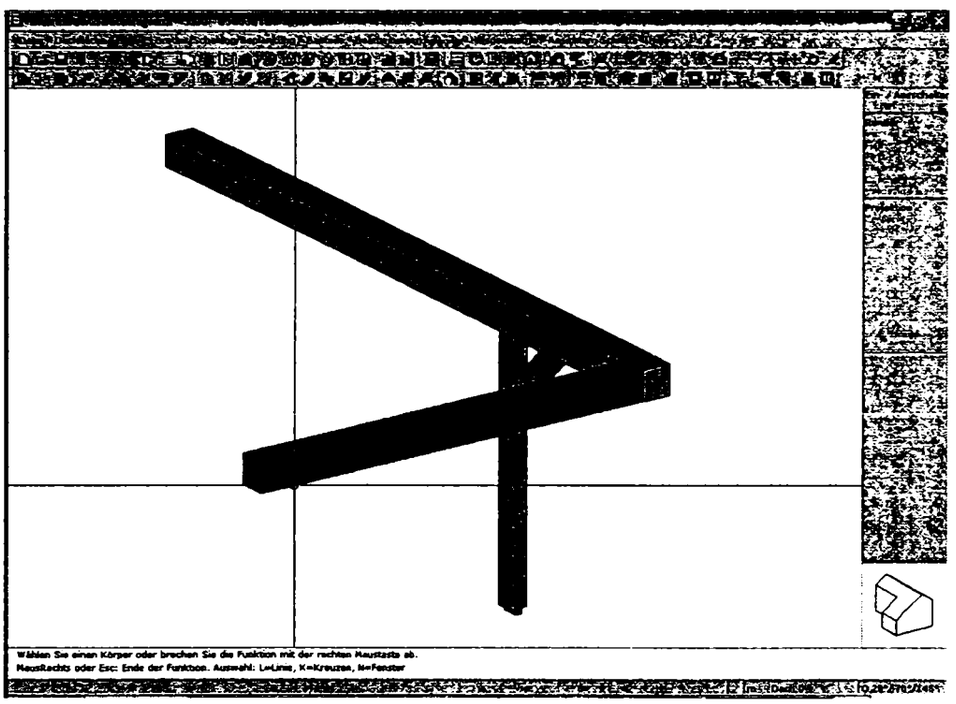


Figura 5-61 Evidențierea obiectului selectat

Practic, această funcție desenează obiectul transmis prin parametrul **`pObj`** cu culoarea **`nColor`**, alternarea culorilor fiind gestionată de către clasa de selecție.

Datorită modului specific OpenGL de desenare a unei vederi, evidențierea unui obiect se face prin marcarea obiectului în structurile interne şi desenarea întregii scene.

Dacă parametrul **`pObj`** este de tip **`CDhpVolume`** atunci desenarea este simplă. Adresa volumului respectiv împreună cu culoarea sunt atribuite variabilelor dedicate din clasa CDhpOpenGLWorkspace. In timpul desenării, dacă obiectul curent are aceeaşi adresă cu cea dată de variabila dedicată, acesta va fi desenat cu culoarea corespunzătoare.

Dacă obiectul evidențiat nu este de tipul **CDhpVolume**, desenarea se face apelându-se modul *feedback* al OpenGL (v. paragraful 4.4.2), obținându-se astfel primitivele grafice ce sunt desenate cu ajutorul funcțiilor grafice GDI prin suprapunerea peste conținutul curent al ferestrei de lucru.

Pașii parcurși sunt:

1. Se selectează RC-ul în fereastra curentă.
2. Se alocă și se inițializează *buffer*-ul de *feedback*:

```
GLfloat *fbuffer = new GLfloat[SIZE_FBUFFER];
glFeedbackBuffer(SIZE_FBUFFER, GL_3D, fbuffer);
```

3. Se inițializează vederea.
4. Se selectează modul *feedback*:

```
glRenderMode(GL_FEEDBACK);
```

5. Se desenează obiectul de evidențiat utilizând transformarea actuală:

```
SetProjectionMatrix();
glMatrixMode(GL_MODELVIEW);
glInitNames();
CDhpList<CDhpObject> *temp;
temp = new CDhpList<CDhpObject>;
temp->AddNode(pObj);
m_pListAuxiliaryGeometry = temp;
DrawScene(0);
temp = m_pListAuxiliaryGeometry;
m_pListAuxiliaryGeometry = NULL;
temp->DeleteList(DONT_DELETE_CONTENT);
delete temp;
temp = NULL;
glFlush();
```

6. Se revine în modul de randare și se determină numărul de valori din *buffer*-ul de *feedback*:

```
size = glRenderMode(GL_RENDER);
```

7. Se analizează *buffer*-ul de *feedback* și se desenează în fereastra curentă primitivele grafice extrase din *buffer*-ul de *feedback* utilizând culoarea **nColor**.

```
GLint count;
GLfloat token;
```

```
count = size;
while (count)
{
  token = fbuffer[size-count]; count--;
  if (token == GL_PASS_THROUGH_TOKEN)
  {
    printf (" %4.2f\n", fbuffer[size-count]);
    count--;
  }
  else if (token == GL_POINT_TOKEN)
  {
    int pt[3];
    pt[0] = (int)fbuffer[size-count]; count--;
    pt[1] = viewport[3] - (int)fbuffer[size-count];
    count--;
    pt[2] = (int)fbuffer[size-count]; count--;
    MoveToEx(m_hDC, pt[0] - (short)(pointDim / 2),
                    pt[1] + (short)(pointDim / 2), NULL);
    LineTo(m_hDC, pt[0] - (short)(pointDim / 2),
                  pt[1] - (short)(pointDim / 2));
    LineTo(m_hDC, pt[0] + (short)(pointDim / 2),
                  pt[1] - (short)(pointDim / 2));
    LineTo(m_hDC, pt[0] + (short)(pointDim / 2),
                  pt[1] + (short)(pointDim / 2));
    LineTo(m_hDC, pt[0] - (short)(pointDim / 2),
                  pt[1] + (short)(pointDim / 2));
  }
  else if (token == GL_LINE_TOKEN)
  {
    int pt1[3], pt2[3];
    pt1[0] = (int)fbuffer[size-count]; count--;
    pt1[1] = viewport[3] - (int)fbuffer[size-count];
    count--;
    pt1[2] = (int)fbuffer[size-count]; count--;
    pt2[0] = (int)fbuffer[size-count]; count--;
    pt2[1] = viewport[3] - (int)fbuffer[size-count];
    count--;
    pt2[2] = (int)fbuffer[size-count]; count--;
    MoveToEx(m_hDC, pt1[0], pt1[1], NULL);
    LineTo(m_hDC, pt2[0], pt2[1]);
  }
  else if (token == GL_LINE_RESET_TOKEN)
  {
    int pt1[3], pt2[3];
    pt1[0] = (int)fbuffer[size-count]; count--;
    pt1[1] = viewport[3] - (int)fbuffer[size-count];
    count--;
    pt1[2] = (int)fbuffer[size-count]; count--;
    pt2[0] = (int)fbuffer[size-count]; count--;
    pt2[1] = viewport[3] - (int)fbuffer[size-count];
    count--;
    pt2[2] = (int)fbuffer[size-count]; count--;
    MoveToEx(m_hDC, pt1[0], pt1[1], NULL);
    LineTo(m_hDC, pt2[0], pt2[1]);
  }
```

```
  else if (token == GL_POLYGON_TOKEN)
  {
    int f, vcount, pt[3];
    vcount = (int)fbuffer[size-count]; count--;
    for(f=0;f<vcount;f++)
    {
      pt[0] = (int)fbuffer[size-count]; count--;
      pt[1] = viewport[3] - (int)fbuffer[size-count];
      count--;
      pt[2] = (int)fbuffer[size-count]; count--;
      if(f)
        LineTo(m_hDC, pt[0], pt[1]);
      else
        MoveToEx(m_hDC, pt[0], pt[1], NULL);
    }
  }
}
```

8. Se dealocă *buffer*-ul de *feedback* şi se părăseşte funcția.

5.6.4. Vederea cu linii ascunse [Sav2005]

Pentru o percepție mai bună a reprezentării tridimensionale a modelului 3D proiectat, utilizatorului i s-au pus la dispoziție mai multe moduri de vizualizare 3D:

- cu linii ascunse (figura 5-62);
- cu linii ascunse desenate punctat (figura 4.63);
- corpuri solide (figura 4.64);
- corpuri solide cu texturi (figura 4.65);
- corpuri solide cu texturi şi muchii (figura 4.66).

O atenție deosebită este acordată modului de implementare a vizualizării cu linii ascunse datorită complexității metodei de obținere a reprezentării grafice. Adițional, se implementează posibilitatea de desenare a liniilor ascunse ca linii punctate. Ambele moduri de vizualizare au fost implementate conform algoritmilor de proiectare enunțați în paragraful 4.4.3.

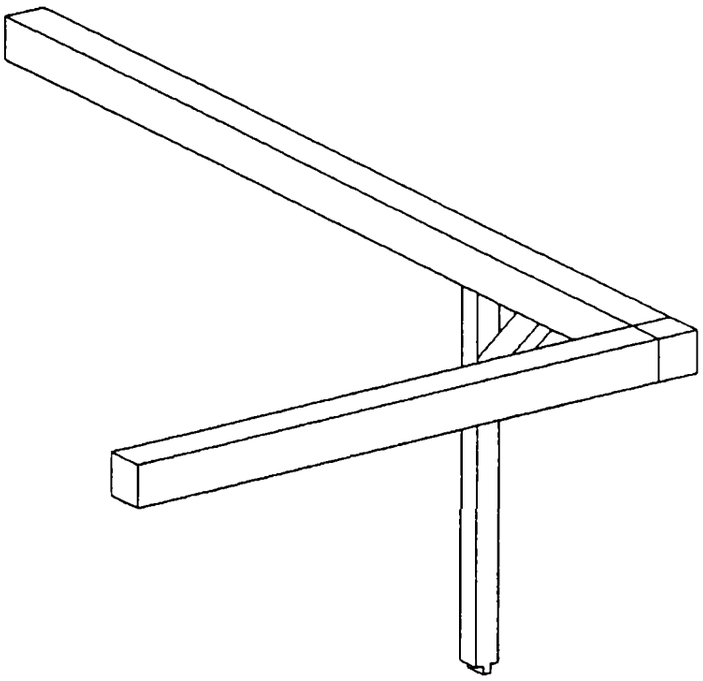


Figura 5-62. Reprezentarea cu linii ascunse

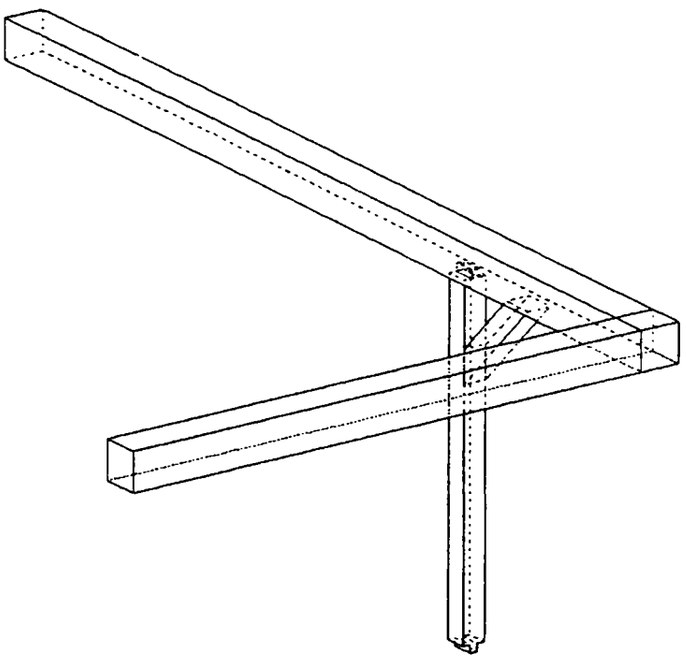


Figura 5-63. Reprezentarea cu linii ascunse desenate punctat

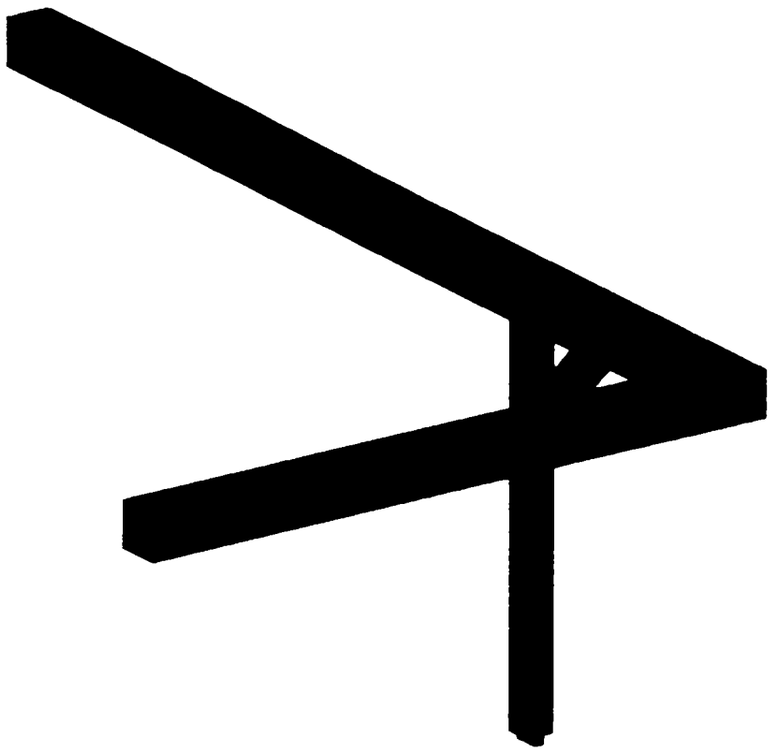


Figura 5-64. Reprezentarea ca și corpuri solide

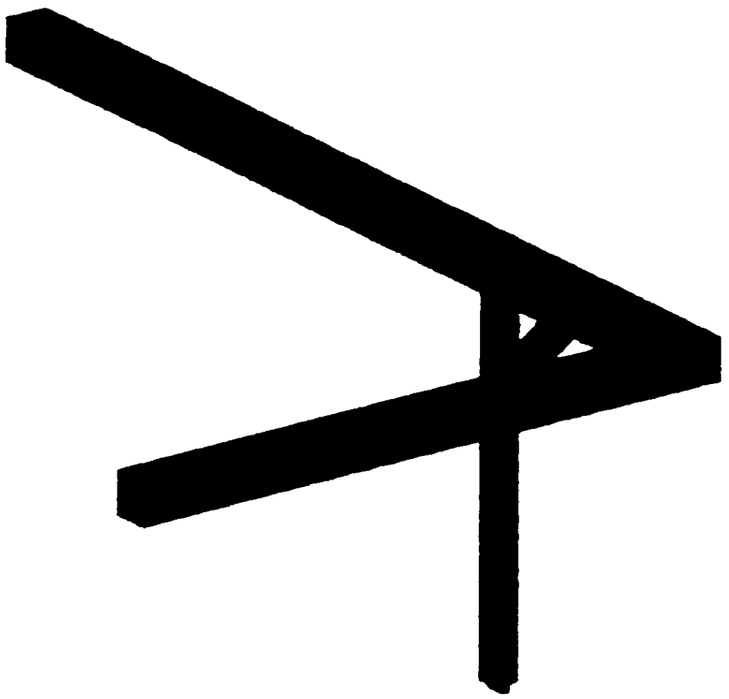


Figura 5-65. Reprezentarea ca și corpuri solide cu texturi

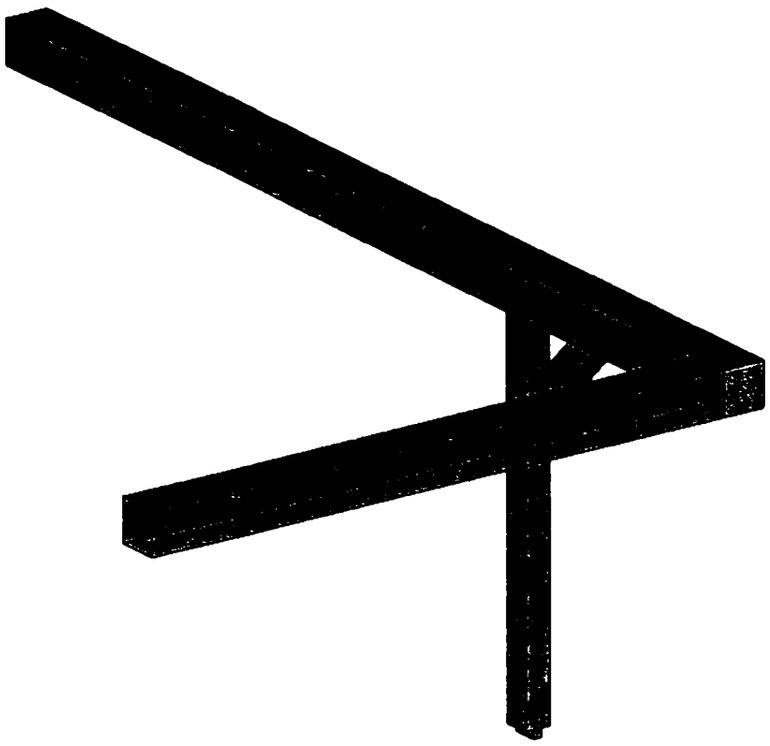


Figura 5-66 Reprezentarea ca şi corpuri solide cu texturi şi muchii

In cazul în care liniile ascunse nu se desenează, vederea se obține prin două desenări ale scenei. La prima desenare, obiectele 3D sunt desenate doar ca muchii (wireframe) iar la a 2-a desenare acestea sunt desenate prin poligoane. In cazul desenării liniilor ascunse ca şi linii punctate mai este necesară o a treia desenare a scenei care precede cele două desenări descrise anterior. In această situație, toate muchiile sunt desenate ca linii punctate. Desenarea scenei prin muchii sau ca şi corpuri solide se realizează prin atribuirea valorilor 0 respectiv 1 variabilei de stare **`m_settings.nPassNo`**. Efectul de linie punctată se obține prin apelul **`glLineStipple(1, 0xC0C0)`**.

Implementarea se realizează după cum urmează:

```
glDisable(GL_TEXTURE_2D);
glShadeModel(GL_FLAT);

if(m_settings.renderState == rsHiddenLines &&
   m_settings.bDottedHiddenLines)
{
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  glLineStipple(1, 0xC0C0);
  glEnable(GL_LINE_STIPPLE);
  m_settings.nPassNo = 1;
  DrawGeometry(nDrawSubEtities);
  glLineStipple(1, 0xFFFF);
```

```
        glDisable(GL_LINE_STIPPLE);
      }

      if(m_settings.renderState == rsHiddenLines &&
         m_settings.bDottedHiddenLines)
      {
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
        m_settings.nPassNo = 0;
        glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
      }
      DrawGeometry(nDrawSubEtities); // umplutura
      if(m_settings.renderState == rsHiddenLines &&
         m_settings.bDottedHiddenLines)
        glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);

      glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
      DrawGeometry(nDrawSubEtities);
      glEnable(GL_TEXTURE_2D);
```

5.6.5. Triangularizarea

După cum s-a arătat în paragraful 4.4.4, poligoanele complexe necesită descompunerea lor în primitive OpenGL, şi anume în triunghiuri.

După triunghiularizare, fețele obiectelor modelate se prezintă ca în figura 5-67. Pentru a obține o reprezentare reală a modelului, liniile adiționale rezultate în urma triunghiularizării trebuie ascunse. Acest lucru de poate realiza prin apelul funcției **glEdgeFlag(GLboolean flag)** cu parametrul **false**. Această funcție indică faptul că următorul vertex transmis către OpenGL reprezintă un început de linie vizibilă sau invizibilă.

In continuare se prezintă codul corespunzător desenării unei fețe:

```
  glNormal3dv(normal);
  glBegin(polygeometry.eType);
    DhpPosition pos = polygeometry.listVG.GetFirstPosition();
    while(pos)
    {
      Dhp3DRVertexGeometry *pVG = polygeometry.listVG.GetNext(pos);
      glEdgeFlag(pVG->bEdgeFlag);
      glTexCoord2dv(pVG->dTexCoord2d);
      glVertex3dv(pVG->dVertex3d);
    }
  glEnd();
```

Prin utilizarea triangularizării s-a urmărit obținerea unei reprezentări tridimensionale cât mai corecte. Prin efectuarea triangularizării o singură dată, doar după modificarea geometriei obiectului, şi păstrarea valorilor vertecşilor într-o structură dedicată s-a obținut o mărire considerabilă a vitezei de desenare.

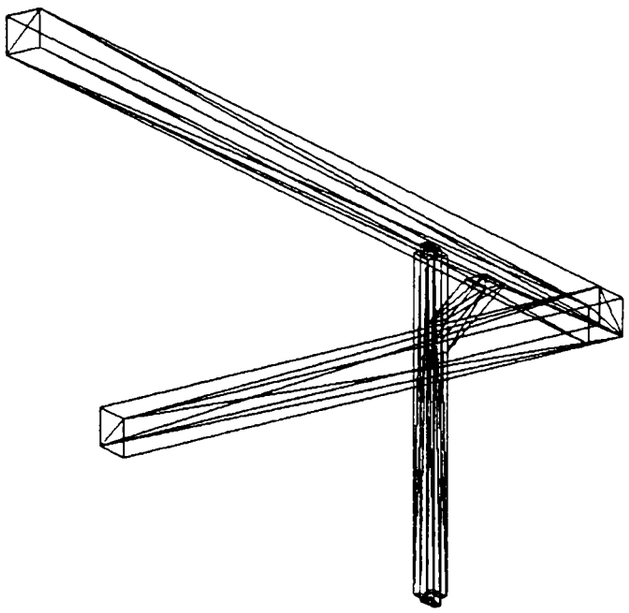


Figura 5-67 triunghiularizarea fețelor obiectelor modelate

5.7. Concluzii şi contribuții

Capitolul prezintă soluțiile de aplicare şi implementare a metodelor şi recomandărilor de dezvoltare software propuse în capitolul anterior, în cadrul interfeței sistemului Dietrich's de proiectare a construcțiilor din lemn, realizată de către autor.

Prezentarea soluțiilor de implementare urmăreşte în detaliu procesul de dezvoltare a următoarelor aplicații:

- *Interfața cu utilizatorul a managerului de proiecte* din cadrul sistemului Dietrich's permite o eficientizare a operațiilor de gestiune a proiectelor şi a pozițiilor de proiect, toate activitățile organizatorice legate de acestea fiind definite, grupate şi puse la dispoziția utilizatorului. Astfel, s-au identificat în cadrul managerului şi respectiv proiectat şi implementat în cadrul interfeței cu utilizatorul facilitățile necesare gestiunii eficiente a proiectelor;
- *Dezvoltarea de elemente personalizate de interfață* a implicat considerarea factorilor care facilitează introducerea şi afişarea informației pentru obținerea unei flexibilități şi eficiențe sporite a interfeței cu utilizatorul. S-au dezvoltat rutine speciale de implementare a funcționalităților ce nu se pot realiza prin mecanisme proprii sistemului de operare. Proiectarea şi optimizarea elementelor de interfață s-a realizat prin evitarea utilizării altor biblioteci de functii, obținându-se o relativă independență față de mediul de implementare precum şi o reducere semnificativă a problemelor la utilizatori. S-au stabilit *principii* pentru o serie de caracteristici ale interfeței cu utilizatorul, pentru a asigura o utilizare cât mai simplă şi o perspectivă cât mai bună asupra modului de introducere a informației;
- *Procesul standard de selecție de obiecte* a fost implementat ca o clasă *template*, cu utilizarea parametrilor determinați în urma condensării informației aferente, beneficiindu-se astfel de facilitățile intrinseci ale *template*-urilor;
- *Gestiunea specială pentru un grup de cutii de dialog* s-a realizat prin simularea unei cutii de dialog virtuale utilizând mai multe cutii de dialog individuale, principalele caracteristici ale abordării fiind programarea OO avansată, reutilizarea codului precum şi gradul ridicat de utilizabilitate;
- *Dezvoltarea vizualizării şi manipulării tridimensionale a modelelor prin utilizarea bibliotecii grafice OpenGL* a implicat integrarea vizualizării OpenGL într-o asemenea

manieră încât codul existent a suferit modificări minime pentru a funcționa corect. In acest sens s-a proiectat şi implementat clasa **`CDhpOpenGLWorkspace`**, care realizează toate operațiile necesare vizualizării OpenGL, precum şi structurile de date pentru păstrarea informației de model într-o formă optimizată pentru utilizarea în OpenGL. Funcțiile de selecție şi evidențiere ale unui obiect se grefează în clasa globală de selecție prin utilizarea funcțiilor **`GetObjectsInReticleFrom-OpenGLWorkspace`** si respectiv **`BlinkObjectsInOpenGLWorkspace`** pentru a-i asigura funcționarea corectă în orice mod de vizualizare. In urma identificării funcțiilor şi operațiilor OpenGL necesare, implementarea vizualizării cu linii ascunse punctate şi a triunghiularizării urmează algoritmii propuşi în cadrul capitolului precedent.

Contribuțiile autorului din cadrul capitolului, în întregime original, se regăsesc în cadrul tuturor subiectelor abordate şi sunt evidențiate în detaliu în cadrul ultimului capitol al tezei.

6. Concluzii finale şi contribuţii personale. Perspective

Proiectarea sistemelor pentru satisfacerea eficientă a utilizatorului în timpul luării deciziilor reprezintă scopul studiilor din cadrul IOC, având ca obiectiv principal înţelegerea interacţiunii dintre utilizator şi calculator. Domeniile psihologiei cognitive şi analizei sistemelor furnizează un cadru teoretic pentru formularea, validarea şi analiza modelelor de percepţie a utilizatorului, fiind esenţiale în luarea de decizii asupra organizării vizuale a interfeţei, precum şi pentru analiza comportării sistemului studiat. Abordarea modelării formale îmbunătăţeşte semnificativ procesul de dezvoltare precum şi calitatea interfeţei cu utilizatorul. Prin descrierea formală a cerinţelor de proiectare, dezvoltatorii sunt obligaţi să implementeze interfaţa mult mai atent, activitatea de specificare şi modelare ajutând la înţelegerea şi clarificarea tuturor aspectelor privind interfaţa cu utilizatorul. Modelul abstract al comportamentului interfeţei permite simularea interacţiunii dintre diferitele componente şi înţelegerea mai profundă a dinamicii ei.

Pentru a obţine cea mai bună soluţie de proiectare este necesar ca proiectantul interfeţei cu utilizatorul să posede cunoştinţe avansate de utilizabilitate, să funcţioneze ca o legătură între utilizatori şi programatori, în favoarea utilizatorilor, luând în considerare întregul lanţ al analizei, proiectării şi evaluării. Ergonomia software şi-a câştigat un rol bine definit în contextul interdisciplinar al proiectării interfeţelor cu utilizatorul, având o contribuţie importantă în procesul de acceptare a calculatoarelor.

Conceperea şi dezvoltarea interfeţelor cu utilizatorul necesită cunoştinţe şi experienţă complexă în domeniul tehnologiilor software actuale, cum sunt programarea orientată spre obiecte, interfeţele grafice cu utilizatorul şi vizualizarea realistă, în scopul de a identifica şi dezvolta acele elemente specifice care sunt cele mai adecvate realizării unei aplicaţii specifice.

Prin prisma necesităţilor aplicaţiilor dezvoltate în capitolul 5 s-au analizat sintetic, unde s-a considerat necesar până la nivel de detaliu, elementele esenţiale programării orientate spre obiecte subliniindu-se în acelaşi timp avantajele acesteia, prin realizarea unei analize paralele între POO şi modul de organizare a realităţii. Se formulează etapele esenţiale de dezvoltare a unui produs software într-o manieră obiectuală, fiind discutate în detaliu în urma evaluării analitice a elementelor necesare proiectării.

Datorită importanţei primordiale a utilizabilităţii şi a criteriilor de realizare a interfeţelor grafice cu utilizatorul din cadrul dezvoltării de software interactiv, se impune analiza principalelor domenii de interferenţă a proiectării sistemelor software cu proiectarea interfeţei cu utilizatorul şi respectiv identificarea elementelor de reper care influenţează decisiv calitatea şi performanţele aplicaţiei finale, elaborându-se reguli şi repere de proiectare a interfeţelor performante cu utilizatorul.

In cadrul aplicaţiilor CAD, vizualizarea realistă şi manipularea obiectelor tridimensionale cu ajutorul bibliotecii grafice OpenGL permite utilizatorului să lucreze într-un mediu performant de proiectare. Studiul implementării vizualizării utilizând biblioteca grafică OpenGL evidenţiază câteva probleme complexe în ceea ce priveşte vizualizarea în ferestre multiple, selecţia şi evidenţierea obiectelor, desenarea cu linii ascunse şi triunghiularizarea utilizând biblioteca grafică OpenGL, necesitând elaborarea unor soluţii avansate de implementare, sub forma unor algoritmi care se implementează în cadrul aplicaţiei.

Pentru a satisface toate cerinţele utilizatorului pentru un sistem CAD este necesară determinarea unor criterii de evaluare eficiente, alegerea corespunzătoare depinzând de mai mulţi factori printre care se numără domeniul industrial, formatul datelor utilizat de client, furnizori şi alte divizii din cadrul organizaţiei, aplicaţii adiţionale necesare.

Capitolul 5, în totalitate original, prezintă soluţiile de aplicare şi implementare a metodelor şi recomandărilor de dezvoltare software propuse în capitolul anterior, în cadrul interfeţei sistemului Dietrich's de proiectare a construcţiilor din lemn realizată de către autor.

Prezentarea soluţiilor de implementare urmăreşte în detaliu procesul de dezvoltare a următoarelor aplicaţii:

- *dezvoltarea unei interfeţe cu utilizatorul pentru un manager de proiecte CAD;*
- *dezvoltarea unor elemente personalizate de interfaţă;*
- *dezvoltarea unui proces standard de selecţie de obiecte;*
- *dezvoltarea unei gestiuni speciale pentru un grup de cutii de dialog;*
- *dezvoltarea unui modul de vizualizare şi manipulare 3D a modelelor prin utilizarea bibliotecii grafice OpenGL.*

Procesul de dezvoltare a aplicaţiilor menţionate a fost ghidat de recomandările, metodologiile formulate respectiv algoritmii propuşi în cadrul capitolelor 3 şi 4.

Implementarea soluţiilor abordează programarea orientată pe obiecte conform strategiei de dezvoltare propuse, utilizându-se elemente specifice programării obiectuale cum sunt încapsularea şi reutilizarea.

Aplicarea criteriilor de realizare a interfeţelor cu utilizatorul precum şi asigurarea unui grad ridicat de utilizabilitate conferă soluţiilor propuse o pronunţată orientare spre utilizator, cu satisfacerea în cât mai mare măsură a cerinţelor acestuia.

Implementarea vizualizării tridimensionale a modelelor proiectate utilizând biblioteca grafică OpenGL pune la dispoziţia utilizatorului un mediu performant de proiectare oferind totodată o reprezentare fotorealistă a modelelor.

Criteriile de evaluare ale unui sistem CAD permit caracterizarea trăsăturilor sistemului de proiectare în general şi a interfeţei cu utilizatorul în particular.

Contribuţiile autorului sunt împărţite în două categorii, teoretice (T) şi aplicative care la rândul lor sunt metodologice (AM) şi produs program (AP).

Contribuţii teoretice

Capitolul 2.

- Analiza schimbărilor evolutive de curente în cadrul dezvoltării de sisteme utilizabile cu identificarea a trei abordări succesive.
- Realizarea unui studiu critic al câtorva din cele mai cunoscute metode de analiză, proiectare şi evaluare din cadrul dezvoltării de sisteme utilizabile.
- Prezentarea critică a cercetărilor din domeniul ergonomiei software, în contextul interdisciplinar al proiectării interfeţelor cu utilizatorul: direcţii de bază, dileme şi condiţii de studiu.

Capitolul 3.

- identificarea domeniului sistemelor cu evenimente discrete ca suport de modelare a interfeţei cu utilizatorul prin sintetizarea principalelor concepte;
- prezentarea critică a principalelor formalisme din cadrul modelării interfeţelor cu utilizatorul;
- analiza critică a trei modele principale de interfaţă cu utilizatorul: sarcină/domeniu, dialog abstract şi interacţiune concretă;

- definirea conceptului de interactor ca mediator de informaţie în contextul interfeţelor cu utilizatorul şi sintetizarea caracteristicilor principale;
- analiza critică a modelării sub formă de interactor a elementelor de interfaţă pe baza de exemple concrete simple şi complexe;

Capitolul 4.

- Considerarea, în cadrul analizei domeniului, a unor criterii de bază de clasificare a claselor, stabilirea de criterii de evaluare a utilităţii claselor de nivel înalt şi identificarea produselor de lucru generate în scopul evaluării efortului de dezvoltare software;
- Analiza utilizării scenariilor, dezvoltarea incrementală şi generarea de prototipuri în contextul etapei analizei cerinţelor sistemului. Identificarea a trei categorii de obiecte corespunzătoare spaţiului informaţional tridimensional, a două tipuri de scenarii pentru generarea modelului de analiză şi a doi factori de risc care pot împiedica obţinerea unor rezultate optime. Enunţarea unor criterii de încheiere a etapei de analiză;
- Identificarea, în cadrul etapei de analiză a cerinţelor software, a trei tipuri de modele în funcţie de metoda utilizată şi enunţarea avantajelor acestei etape;
- Sintetizarea, în contextul etapei proiectării software, a principalelor caracteristici ale software-ului comercial. Identificarea şi analiza a patru categorii de excepţii recomandate a fi tratate şi respectiv a opt reguli de bază pentru prototipizare. Evidenţierea beneficiilor etapei şi prezentarea dezavantajelor evitării *template*-urilor. Introducerea unor criterii de evaluare a tehnicilor de proiectare şi implementare OOP şi a unor criterii de evaluare a rezultatelor proiectării.
- Enunţarea unui criteriu de stabilire a modului de implementare.
- Sintetizarea unor aspecte teoretice şi practice legate de implementarea software, în care activitatea de programare este susţinută de o strategie de testare de detectare a erorilor;
- Realizarea unor analize critice legate de utilizarea *template*-urilor, de procesul de prototipizare şi de implementarea unei componente reutilizabile.
- Structurarea, formularea şi analiza domeniului utilizabilităţii, cu un rol în stabilirea gradului de utilizabilitate a unui sistem – element care stă la baza dezvoltării aplicaţiilor propuse în capitolul 5. Enunţarea definiţiei utilizabilităţii şi propunerea

unui *model stratificat* pe trei niveluri al utilizabilităţii pentru evidenţierea caracteristicilor implicate în evaluarea ei. Caracterizarea analitică a elementelor cu un aport major în asigurarea unui grad ridicat de utilizabilitate.

- Introducerea unor criterii de realizare a interfeţelor grafice, enunţarea unor principii majore pentru o afişare vizuală satisfăcătoare a informaţiei, analiza elementelor de reper care influenţează decisiv calitatea şi performanţele aplicaţiei finale propuse în capitolul 5.

Capitolul 5.

- Stabilirea de principii pentru o serie de caracteristici ale interfeţei cu utilizatorul, pentru a asigura o utilizare cât mai simplă şi o perspectivă cât mai bună asupra modului de introducere a informaţiei.

Contribuţii aplicative de factură metodologică

Capitolul 3.

- modelarea ca SED a unei sarcini "selecţia standard de obiecte", prin identificarea stărilor, a stării iniţiale, a stărilor marcate şi a mulţimii evenimentelor, definirea funcţiei de tranziţie şi a funcţiei eveniment activ;
- elaborarea diagramei de tranziţii de stare ca etapă pregătitoare pentru realizarea schemei logice de funcţionare.

Capitolul 4.

- Conceperea a trei metode de tratare a excepţiilor pentru realizarea unui sistem bun tolerant la defecţiuni;
- Identificarea şi evidenţierea aspectelor practice legate de utilizarea template-urilor, de procesul de prototipizare şi de implementarea unei componente reutilizabile.
- Elaborarea unor recomandări care se constituie ca un ghid pentru proiectarea interfeţelor grafice.
- Propunerea unor reguli de utilizare a mouse-ului şi tastaturii.
- Elaborarea unui set de reguli de proiectare a elementelor de interfaţă personalizate.
- Elaborarea unor algoritmi specifici de implementare pentru:
    - vizualizarea în ferestre multiple;

- utilizarea modului de selecţie şi reacţie (feedback) pentru selecţia şi evidenţierea pe ecran a unei entităţi grafice;
- desenarea în mod *wireframe* şi cu linii ascunse pentru o mai bună percepţie a reprezentării tridimensionale a modelului 3D proiectat;
- triunghiularizarea poligoanelor complexe utilizând biblioteca utilitară a OpenGL.

Capitolul 5.

- Identificarea în cadrul managerului de proiecte precum şi proiectarea şi implementarea în cadrul interfeţei realizate a facilităţilor necesare gestiunii eficiente a proiectelor.
- Dezvoltarea de rutine speciale de implementare a funcţionalităţilor ce nu se pot realiza prin mecanisme proprii sistemului de operare.
- Dezvoltarea interfeţei astfel încât desenarea previzualizării unei poziţii să fie efectuată în paralel, fiind posibilă întreruperea acesteia.
- Sublinierea importanţei evitării utilizării altor biblioteci în cadrul procesului de proiectare şi optimizare a elementelor de interfaţă, prin elaborarea unor elemente de interfaţă personalizate, pentru a obţine o relativă independenţă faţă de mediul de implementare precum şi o reducere semnificativă a problemelor la utilizatori.
- Determinarea, proiectarea şi implementarea de cutii de dialog şi funcţii reutilizabile, specifice fiecărui tip de date, în urma identificării tipurilor de valori pe care utilizatorul trebuie să le specifice aplicaţiei în cadrul procesului de proiectare. Atunci când între valorile introduse există o relaţie de constrângere, la modificarea uneia celelalte două se recalculează automat.
- Determinarea, în cadrul interfeţei, a unui set de parametri configurabili de către utilizator şi punerea lor la dispoziţia acestuia prin intermediul unei cutii de dialog dedicate.
- Proiectarea şi implementarea de bare configurabile de instrumente cu acces direct la funcţii.
- Proiectarea şi implementarea unui model de bază pentru cutiile de dialog de operare cu fişiere.
- Identificarea, proiectarea şi implementarea de elemente de interfaţă pentru funcţiile unui manager de biblioteci de modele.

- Proiectarea şi implementarea de elemente de interfaţă cu utilizatorul privind o modalitate de grupare a prelucrărilor sub forma de punct-simboluri.
- Proiectarea unei modalităţi de generare a listelor de materiale prin stabilirea de criterii de sortare a elementelor, caracterizarea tipurilor de liste, proiectarea cutiei de dialog precum şi sistematizarea în pagină a datelor.
- Conceperea şi descrierea unui proces de selecţie de obiecte respectiv implementarea acestuia sub forma unei clase *template* cu utilizarea parametrilor determinaţi în urma condensării informaţiei aferente.
- Proiectarea unei cutii de dialog virtuale şi implementarea funcţiei de gestiune a acesteia prin simularea cu mai multe cutii de dialog.
- Proiectarea şi implementarea clasei **`CDhpOpenGLWorkspace`**, care realizează toate operaţiile necesare vizualizării OpenGL, precum şi a structurilor de date pentru păstrarea informaţiei de model într-o formă optimizată pentru utilizarea în OpenGL.
- Proiectarea şi implementarea într-un mod eficient şi transparent a procesului de selecţie al unui obiect în cadrul vizualizării OpenGL prin funcţia **`GetObjectsInReticleFromOpenGLWorkspace`**, apelul grefându-se în clasa globală de selecţie pentru a-i asigura funcţionarea corectă în orice mod de vizualizare.
- Sintetizarea a două proceduri, prin desenarea directă şi prin modul de randare **`GL_FEEDBACK`** al OpenGL, pentru evidenţierea obiectului selectat.
- Implementarea paşilor descrişi în cadrul algoritmilor de desenare cu linii ascunse şi de triunghiularizare în urma identificării funcţiilor şi operaţiilor OpenGL necesare.

Contribuţii aplicative de produs program

Capitolul 5.

- Realizarea de elemente de interfaţă grafică cu utilizatorul pentru Managerul de Proiecte din cadrul sistemului CAD Dietrich's.
- Realizarea de elemente de interfaţă grafică cu utilizatorul pentru modulul de proiectare *Bauwerk* din cadrul sistemului Dietrich's.
- Realizarea modulelor pentru vizualizare tridimensională realistă şi manipularea obiectelor pentru modulul de proiectare *Bauwerk* din cadrul sistemului Dietrich's.

Perspective

Se constată că noile tehnologii din cadrul interfeţelor cu utilizatorul la calculatoarele personale sunt relativ puţin utilizate în economie şi administraţie, dar acest fapt se află în momentul de faţă în atenţia cercetătorilor şi a utilizatorilor. Dezvoltările se îndreaptă spre tehnologii multimedia în care sunt combinate secvenţe de text, imagini şi sunete. Pe lângă introducerea datelor cu ajutorul tastaturii, utilizatorul va interacţiona cu sistemul şi prin scrierea de mână şi utilizarea mesajelor vocale. Ca şi posibilităţile date de 'realitatea virtuală', există o tendinţă de simplificare a utilizării sistemelor.

Până în momentul de faţă, ergonomia software a încercat să îndeplinească necesităţile utilizatorului individual. Prin adaptarea organizării muncii din diferite domenii la munca în grup, denumită si "muncă cooperativă asistată de calculator" [Maa1993], a devenit importantă alinierea sistemelor la noile cerinţe. Prin '*Groupware*" este posibil schimbul informaţiei şi accesul la datele comune de către membrii grupului, dacă este necesar chiar şi simultan.

Se mai fac cercetări şi în domeniul organizării iterative şi participative a sistemului. Astfel se pune problema in cadrul procesului de dezvoltare a software-ului a modului cum se poate ţine cont de diferitele grupuri ce vor utiliza sistemul de calcul şi cum pot fi pregătiţi utilizatorii pentru o mai bună operare a sistemului.

Cercetările se vor axa în continuare pe creşterea în continuare a performanţelor interfeţelor şi pe direcţia nouă, a adaptării acestora la munca în grup.

Bibliografie

[Aca2002] *Encyclopaedia of Information Systems*, Academic Press, in press, 2002

[Ale2001] Alexandrescu, A., *Modern C++ Design: Generic Programming and Design Patterns Applied,* Addison Wesley, 2001

[Amb1998] Amber, S.W., *User Interface Design: Tips and Techniques*, http://www.ambysoft.com/essays/userInterfaceDesign.html, 1998

[Ast2004] Astle, D., Hawkins, K., *Beginning OpenGL Programming*, Premier Press, 2004

[Att1989] Attarwala, Y., *Rendering Hidden Lines*, Iris Universe, Summer: 40-41, 1989

[Ban1989] Bansler, J. *Systems Development Research in Scandinavia: Three Theoretical Schools*. Scandinavian Journal of Information Systems, vol. 1, pp. 3-22, 1989.

[Ben1992] Benyon, D., *The role of task analysis in systems design*. Interacting with computers, 4 (1), pp. 102-123, 1992

[Ben1996] Benyon, D. R., *Domain Models in User Interface Design*. Critical Issues in User Interface Systems Engineering, Springer-Verlag, http://www.dcs.napier.ac.uk/~dbenyon/domainmodels.pdf , 1996

[Ber2003] Jason Beres, *Sams Teach Yourself Visual Studio® .NET 2003 in 21 Days*, Sams Publishing, 2003

[Bia1991] Bias, R.C., *Walkthroughs: Efficient Collaborative Testing*. IEEE Software, 8 (5), pp. 94-95, 1991

[Blu2003] Bill Blunden, *Software Exorcism: A Handbook for Debugging and Optimizing Legacy Code*, Apress, 2003

[Boo1999] Booch, G., Rumbaugh, J., Jacobson, I., *The Unified Modeling Language User Guide Second Edition*, Addison Wesley Professional, 2005

[Car1983] Card, S.K., Moran, T.P., & Newell, A., *The Psychology of Human-Computer Interaction*. Hove, England: Lawrence Erlbaum Associates, Inc., 1983

[Car1989] Carroll, J.M., & Kellogg, W. A., *Artifacts as theory-nexus: Hermeneutics meets theory-based desig,* Proceedings of Human Factors in Computing Systems, CHI' 89, pp. 7-14, New York, NJ: ACM, 1989

[Cas2001] Cassandras, C.G., Lafortune, S., *Introduction to Discrete Event Systems*, Kluwer Academic Publishers, 2001

[Cog2004] Jeff Cogswell, *Designing Highly Useable Software*, Sybex, 2004

[Coo2000] J. W. Cooper. *Java Design Patterns, A Tutorial*. Addison-Wesley, Reading, MA, 2000;

[Coo2003] Alan Cooper and Robert Reimann, *About Face 2.0: The Essentials of Interaction Design*, John Wiley & Sons, 2003

[Cro2001] Crowle, S., Hole, L., *Seeing the wood for the trees: A framework for the specification of metaphor in interface design*, http://www.comp.lancs.ac.uk/computing/research/mcg/mmm/papers/crowle.pdf, 2001

[Daw2004] Dawson, M, Beginning *C++ Game Programming*, Premier Press, 2004

[Dem1978] Demarco, T.. *Structured Analysis and System Specification.* New York: Yourdon Press, 1978

[Deu1988] Deutsch, M.S., *Focusing Real Time Systems Analysis on User Operations*, IEEE Software, 1988

[Die2002] DietrichAG *Bauwerk Application Documentation*, 2002.

[Dra2005] Dragomir, T.L., *Elemente de Teoria Sistemelor Vol.1*, Editura Politehnica, Timişoara, 2005

[Ehn1997] Ehn, P., Löwgren, J., *Design for Quality-in-use: Human-Computer Interaction Meets Information Systems Development*, Handbook of Human-Computer Interaction, pp. 299-313, Amsterdam: Elsevier Science B.V., 1997

[Eis2000] Eisenstein, J., Vanderdonckt, J., Puerta, A., *Applying Model-Based Techniques to the Development of UIs for Mobile Computers*, http://www.ximl.org/documents/XIMLMultiChannel.pdf, 2000

[Far2001] Fara, H., *Abordare bazată pe tehnologia orientată spre obiecte a procesului de dezvoltare software, Teza de Doctorat,* Universitatea "Politehnica" Timişoara, România, 2001

[Fav2003] Liliana Favre, *UML and the Unified Process*, IRM Press, 2003

[Fin2004] Finney, K., C., *3D Game Programming All In One*, Premier Press, 2004

[Flo1987] Floyd, C., *Outline of a Paradigm Change in Software Engineering*, Computers and Democracy, pp. 193-210, Avebury, 1987

[Fol1995] Foley, J.D., van Dam, A., *Computer Graphics: Principles and Practice*, Reading, MA: Addison-Wesley, 1995

[Fow1995] Fowler, S.L.; Stanwick, *V.R. The GUI Style Guide*. AP Professional, 1995

[Fow2003] Fowler, M., *UML Distilled: A Brief Guide to the Standard Object Modeling Language*, Third Edition, Addison Wesley, 2003

[Gal2002] Wilbert O. Galitz, *The Essential Guide to User Interface Design-An Introduction to GUI Design Principles and Techniques*, Second Edition, Wiley Computer Publishing, 2002

[Gam1995] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995

[Gar1991] Gardan, Y. *La CFAO. Introduction, techniques, et mise en oevre*. Hermes, Paris, 1991.

[Gau1994] Gaudel, M-C. *Formal specification techniques*, Proceedings of the 16th international conference on Software engineering - ICSE, May 1994, pp. 223-227, IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.

[Gle1991] Gleitman, H., *Psychology (3rd ed.)*. New York: W.W. Norton & Company, Inc., 1991

[Goo1986] Good, M., Spine, T.M., Whiteside, J., & George, P., *User-derived impact analysis as a tool for usability engineering*, Proceedings of CHI' 89 Conference on Human Factors in Computing Systems, pp. 241-246, New York: ACM, 1986

[Gre1991] Greenbaum, J., Kyng, M., *Design at Work: Cooperative Design of Computer Systems*. Hillsdale, NJ: Lawrence Erlbaum Associates., 1991

[Gru1989] Grudin, J., *The Case Against User Interface Consistency*, Communications of the ACM, 32 (10), pp. 1164-1173, 1989

[Gru1991] Grudin, J., *Interactive Systems: Bridging the Gaps Between Developers and Users*. IEEE Computer, 24 (4), pp. 59-69, 1991

[Gru1992] Grudin, J., *Utility and usability: research issues and development contexts*. Interacting with computers, 4 (2), pp. 209-217, 1992

[Gul1993] Gulliksen, J., Johnson, M., Lind, M., Nygren, E., Sandblad, B., *The Need for New Application Specific Interface Elements*, Proceedings of HCI International' 93, pp. 15-20, New York: Elsevier Science B.V., 1993

[Gul1996] Gulliksen, J., & Sandblad, B., *Domain Specific Design of User Interfaces – Case Handling and Data Entry Problems*, Critical Issues in User Interface System Engineering, pp. 21–36, New York: Springer Verlag, 1997

[Gul2004] Gulliksson, H., *Models for exploring design spaces*, http://www.tfe.umu.se/program/DOIT/HITI_webb/HITI%20mediation%20final%201.pdf, 2004

[Hac1986] Hacker, W., *Arbeitspsychologie*, Huber, 1986

[Han2005] Hansen, C.D., Johnson, C.R., *The Visualization Handbook*, Elsevier, 2005

[Har1987] Harel, D., State Charts: *A visual Formalism for Complex Systems, Science of Computer Programming*, Vol. 8., pp. 231-274, North-Holland, Amsterdam, 1987

[Hat1987] Hatley, D.J., and Pirbhai, I.A., *Strategies for Real-Time System Specification*, Dorset House, New York, NY, 1987

[Hen1986] Henderson, A., & Card, S.K., *Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface*. ACM Transactions on Graphics, 5 (3), pp 211-243, 1986

[Her1995] Herrel, R., Baldwin, J., Wilcox, Ch., *High Quality Polygon Edging*, IEEE Computer Graphics and Applications, 15(4), pp. 68-74, July 1995.

[Hoh2003] Luke Hohmann, *Beyond Software Architecture: Creating and Sustaining Winning Solutions*, Addison Wesley, 2003

[Hol1991] Holleran, P.A., *A methodological note on pitfalls in usability testing*. Behaviour & information technology, 10 (5), pp. 345-357, 1991

[Hol1993] Holtzblatt, K., & Jones, S., *Contextual Inquiry: A Participatory Technique for System Design,* Participatory Design: Principles and Practices, pp. 177-210, Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers, 1993

[ISO1995] ***, ISO/DIS 9241. *Ergonomic Requirements for Office Work with Visual Display Terminals (VDTs) - Part 10: Dialogue Principles, Part 11: Guidance on Usability, Part 13: User Guidance,* Geneva, Switzerland: International Organization for Standardization, 1995

[Jac1992] Jacobson, I., Christerson, M., Jonsson, P., & Övergaard, G., *Object-Oriented Software Engineering. A Use Case Driven Approach*. Wokingham, England: Addison-Wesley Publishing Company, 1992

[Jef1997] Jeffries, R., *The Role of Task Analysis in the Design of Software*, Handbook of Human-Computer Interaction, pp. 347-359, Amsterdam, Elsevier Science B.V., 1997

[Joh1992] Johnson, P., *Human-Computer Interaction, Psychology, Task Analysis and Software Engineering*. London: McGraw-Hill Book Company, 1992

[Joh2000] Jeff Johnson, *GUI Bloopers: Don'ts and Do's for Software Developers and Web Designers*, Morgan Kaufmann Publishers, 2000

[JØR1990] Jørgensen, A.H, *Thinking-aloud in user interface design: a method promoting cognitive ergonomics*. Ergonomics, 33, (4), pp. 501-507, 1990

[Kak2003] Avinash C. Kak, *Programming with Objects: A Comparative Presentation of Object-Oriented Programming with C++ and Java*, John Wiley & Sons, , 2003

[Kal1999] Kalev, D., *ANSI/ISO C++ Professional Programmer's Handbook*, Que, 1999

[Kul2003] Kulak, D., Guiney, E., *Use Cases: Requirements in Context*, Second Edition, Addison Wesley, 2003

[Lam2003] LaMothe, A., *Tricks of the 3D Game Programming Gurus: Advanced 3D Graphics and Rasterization*, Sams Publishing, 2003

[Let1998] Letia, T., Astilean, A., *Sisteme cu evenimente discrete (modelare, analiza, sinteza, control)*, Edit. M Informatica, Cluj, 1998

[Lew1982] Lewis, C., *Using the 'thinking-aloud' method in cognitive interface design.* (IBM Research Report RC 9265, 2/17/82). Yorktown Heights, NY: IBM T.J. Watson Research Center, 1982

[Lif1998] Lif, M., *Adding Usability - Methods for Modelling, User Interface Design and Evaluation.* Acta Univ. Ups., Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science and Technology Uppsala, 359, VIII+42 pp.. Uppsala, 1998

[Lin1991a] Lind, M., *Effects of Sequential and Simultaneous Presentations of Information* (Rep. No. 19, CMD). Uppsala, Sweden: Uppsala University, 1991

[Lin1991b] Lind, M., Nygren, E., Sandblad, B., *Kognitiva arbetsmiljöproblem och gränssnittsdesign [Cognitive work environment problems and design of user interfaces]* (Rep. No. 20, CMD). Uppsala, Sweden: Uppsala University, 1991

[Lip2005] Stanley B. Lippman, Josée Lajoie, Barbara E. Moo, *C++ Primer, Fourth Edition,* Addison Wesley Professional, 2005

[Lun2003] Luna, F., *Introduction to 3D Game Programming with DirectX 9.0*, Wordware Publishing, 2003

[Maa1993] Maaß, S., *Software-Ergonomie, Benutzer- und aufgabenorientierte Systemgestaltung*. Informatik Spektrum, Band 16, Heft 4, August 1993, pp. 191-205, Springer-Verlag, 1993

[Mac1991] Maclean, A., Young, R.M., Bellotti, V.M.E., & Moran, T.P, *Questions, Options and Criteria: Elements of Design Space Analysis*. Human-Computer Interaction, 6, pp. 201-250, 1991

[Mal2000] Malloy, R., *2D to 3D shift, sources for 3D sofware & benefits of 3D over 2D*, http://www.cad-forum.com, 2000

[Mar1996] Markopoulos, P., Rowson, J., Johnson, P., *On the Composition of Interactor Specifications*, Proceedings of the BCS-FACS Workshop on Formal Aspects of the Human Computer Interface, Sheffield Hallam University, 10-12 September 1996, http://ewic.bcs.org/conferences/1996/formalaspects/papers/paper9.pdf, 1996

[Mar1997] Marshak, R.T. *Workflow: Applying Automation to Group Processes*. Groupware - Collaborative Strategies for Corporate LANs and Intranets, pp. 68-97, Prentice Hall PTR, 1997.

[Mcr2005] McReynolds, T., Blythe, D., *Advanced Graphics Programming Using OpenGL*, Morgan Kaufmann, 2005

[Mcs2003] McShuffrey, M., *Game Coding Complete*, Paraglyph Press, 2003

[Mul1997] Muller, M.J., Haslwanter, J.H, & Dayton, T., *Participatory Practices in the Software Lifecycle. În M. Helander*, Handbook of Human-Computer Interaction, pp. 255-297, Amsterdam: Elsevier Science B.V., 1997

[Mye1990] Myers, D.R. *GKS and GKS-3D Primer*, ASD Group -- Program Library Section, Computing and Networks Division, CERN Geneva, Switzerland, http://wwwasdoc.web.cern.ch/wwwasdoc/WWW/gks/gksprimer.html, 1990

[Nie1990] Nielsen, J., & Molich, R, *Heuristic evaluation of user interface*, Proceedings of Human Factors in Computing Systems, CHI' 90, pp. 249-256, New York, NJ: ACM, 1990

[Nie1993] Nielsen, J., *Usability Engineering*. San Diego: Academic Press, Inc, 1993

[Nie1995] Nielsen, K.W., *Software Development with C++, Maximizing Reuse with Object Technology*, Academic Press, 1995

[Nor1986] Norman, D., Draper, S., *User Centered System Design : New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Assoc, 1986

[Nor1986a] Norman, D.A., *Cognitive Engineering*, User Centered System Design, pp. 31-61, Hillsdale, New Jersey: Lawrence Erlbaum Associates, Inc., 1986

[Nyg1996] Nygren, E., Allard, A., & Lind, M., *Skilled Users' Interpretation of Visual Displays*. Report no. 63/96 from Uppsala University Center for Human-Computer Studies, Uppsala, Sweden, 1996

[Ols1993] Olsson, E., Göransson, B., Borälv, E., & Sandblad, B., D*omain Specific Style Guide – Design and Implementation*, Proceedings of the MOTIF' 93 & COSE International User Conference, pp. 133-139, Washington, DC: Open Software Foundation, 1993

[Opp1989] Opperman, R, Murchner, B, Paetau, M, Pieper, M, Simm, H, and Stellmacher, *Evaluation von Dialogsystemen, Der Software-ergonomische Leitfaden EVADIS*, Walter de Gruyter, Leiden, 1989

[Ove1992] Overboom, G.R., *WELBORIX, a Novel Approach towards User-Interface Technology in Well-Performance Simulation* , Artifical Intelligence in the Oil Industry: Knowledge Based Systems, Neural networks, Fuzzy Logic. EuroCAIPEP, 16-18 Oct. 1991, Vol. 47, n°03, 1991

[Öve2004] Övergaard, G., Palmkvist, K., *Use Cases Patterns and Blueprints*, Addison Wesley Professional, 2004

[Pas1997] Pastravanu., O., *Sisteme cu evenimente discrete*, Editura Matrix-Rom, Bucuresti, 1997

[Pay1989] Payne, S.J., & Green, T.R.G., *Task-Action Grammar: the model and its developments*, Task Analysis for Human Computer Interaction, pp. 75-105, Chichester: Ellis Horwood, 1989

[Pfa1985] Pfaff, G.E., *User Interface Management Systems* (Eurographic Seminars), 1985

[Phi2002] Philipps, P., Muresan, V., Fara, H., Savii, G.C., et al., *Custom Graphic User Interface Guidelines*, Dietrich AG, 2002

[Phi2004] Philipps, P., Muresan, V., Fara, H., Savii, G.C. and al., e. *Application Specifications*, 2002

[Pol1992] Polson, P.G., Lewis, C., Rieman, J., & Wharton, C., *Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces*. International Journal of Man-Machine Studies, 36 (5), pp. 741-773, 1992

[Pre1994] Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S., & Carey, T., *Human-Computer Interaction*. Wokingham, England: Addison-Wesley Publishing Company, 1994

[Pri2001] Pribeanu, C., Vanderdonckt, J., Limbourg, Q., Souchon, N., Florins. M., *Task modelling for context sensitive user interfaces,* Proc. Of 8th International Workshop on Design, Specification, Verification of Interactive Systems DSV-IS'2001 (Glasgow, 13-15 Juin 2001), Lecture Notes in Computer Science, pp. 49 – 68, Springer-Verlag, Berlin, 2001

[Pro2004] Prostean, O., *Notite de curs,* Universitatea Politehnica Timisoara, 2004

[Ras1983] Rasmussen, J., *Skills, Rules, and Knowledge; Signals, Signs, and Symbols, and Other Distinctions in Human Performance Models,* IEEE Transactions on Systems, Man, and Cybernetics, 13 (3), pp. 257-266, 1983

[Rea1987] Reason, J.T., *Generic Error-Modeling System (GEMS): A Cognitive Framework for Locating Common Human Error Form*, New Technology and Human Error, pp. 63-83, Chichester, England: Wiley, 1987

[Rom2003] Romanik, P., Muntz, A., *Applied C++: Practical Techniques for Building Better Software*, Addison Wesley, 2003

[Ros2004] Rost, R.J., *OpenGL® Shading Language*, Addison Wesley, 2004

[Sav1997] Savii, G., *Bazele proiectării asistate de calculator*. Mirton, Timişoara, 1997

[Sav2001] Savii, G.C., *Referat de Doctorat, Proiectarea şi Realizarea Interfeţelor Performante în Dezvoltarea Aplicaţiilor CAD/CAM*, Univeristatea Politehnica Timişoara, 2001

[Sav2002] Savii, G.C., Prostean, O., *Evaluation Criteria for Computer Aided Design Systems*, Periodica Politechnica, Transactions On Automatic Control And Computer Science, pp. 84-87, Vol. 47 (61), 2002

[Sav2002b] Savii, G.C., *Principles for Dialog Box Design and Operation Within a CAD System for Wood Construction*, Periodica Politechnica, Transactions On Automatic Control And Computer Science Vol.47 (61), pp. 88-91, 2002

[Sav2003] Savii, G.C., *User Interface Optimizations For Beam Processing Sets*, The 14th International Conference On Control Systems And Computer Science, July 2-5, (Bucharest, 2003).

[Sav2003b] Savii, G.C., *Initiating Remote Control Sessions on Computers behind NAT*, Periodica Politechnica, Transactions On Automatic Control And Computer Science, pp. 52-55, Univeristatea Politehnica Timisoara, Vol.48 (62), 2003

[Sav2004] Savii,G.C., Proştean, O., *Special Management of Dialog Boxes within a Timber Construction CAD System*, Periodica Politechnica, Transactions On Automatic Control And Computer Science, pp. 149-152, Univeristatea Politehnica Timisoara, Vol.49 (63), 2004

[Sav2004b] Savii, G.C, *Standard Selection of Objects within a Wood Construction CAD System*, Proceedings of the 1st Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, pp.125-131, Timisoara, Romania, May 25-26, 2004

[Sav2005] Savii, G.C., *Visualization techniques for models using OpenGL*, Scientific Bulletin Of "Politehnica" University Of Timisoara, Romania, Transactions On Automatic Control And Computer Science, Vol. 50 (64), 2005

[Sav2005b] Savii, G.C., *Advanced Developments in Standard Selection of Objects within a Wood Construction CAD System*, Scientific Bulletin Of "Politehnica" University Of Timisoara, Romania, Transactions On Automatic Control And Computer Science, Vol. 50 (64), 2005

[Sch1977] Schneider, W., & Shiffrin, R.M., *Controlled and Automatic Human Information Processing I*, Psychological Rev., 84, pp. 1-66, 1997

[Sch1983] Shneiderman, B., *Software Psychology:Human Factors in Computer and Information Systems*, 1983

[Sch2004] Schildt, H., *The Art of C++*, McGraw-Hill/Osborne, 2004

[Seg2004] Segal, M., Akeley, K., *The OpenGL Graphics System: A Specification (Version 2.0 - September 7, 2004)* http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf, ultimul acces: 8 octombire 2005

[She1989] Shepherd, A., *Analysis and training in information technology tasks*. În D. Diaper (Ed.), Task Analysis for Human Computer Interaction, pp. 15-55, Chichester, England: Ellis Horwood, 1989

[Shi1981] Shiffrin, R.M., & Dumais, S.T., *The Development of Automatism*., Cognitive Skills and their Acquisition, pp. 111-140, Hillsdale, NJ: Erlbaum, 1981

[Shn1992] Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction (2nd ed.)*. U.S.A: Addison-Wesley Publishing Company, Inc., 1992

[Shr2003] Shreiner, D., Woo, M., Neider, J., Davis, T., *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.4, Fourth Edition*, Addison-Wesley Professional; 2003

[Shr2004] Shreiner, D., *OpenGL® Reference Manual: The Official Reference Document to OpenGL, Version 1.4*, 4/E, Addison Wesley Professional, 2004

[Sht2000] Shtern, V., *Core C++ A Software Engineering Approach*, Prentice Hall PTR, 2000

[Sie2003] Siedersleben, J., *Errors and Exceptions – Rights and Responsibilities*, sd&m Research, Munich, http://www.sdm.de/web4archiv/objects/download/pdf/vonline_siedersleben_eco op03.pdf, 2003

[Sny2003] Snyder, C., *Paper Prototyping: The Fast and Easy Way to Define and Refine User Interfaces*, Morgan Kaufmann Publishers, 2003

[Sol1991] Solso, R.L., *Cognitive Psychology*. Boston: Allyn and Bacon, 1991

[Sol2005] Nicholas A. Solter, Scott J. Kleper, *Professional C++*, Wiley Publishing, Inc., 2005

[Sta2000] Stark, J., *Managing CAD/CAM/CAE*, http://www.johnstark.com, 2000

[Sta2004] Stahler, W., *Beginning Math and Physics for Game Programmers*, New Riders Publishing, 2004

[Sti1999] Stirewalt, R.E.K., *Separating concerns in direct manipulation user interfaces*, Proceedings of the IEEE International Conference on Automated Software Engineering (ASE'99), pp. 199-, Cocoa Beach, Florida, http://citeseer.ist.psu.edu/cache/papers/cs/23863/http:zSzzSzwww.cse.msu.eduzSz~stirezSzPaperszSzase99.pdf/stirewalt99separating.pdf October 1999

[Str1997] Stroustrup, B., *The C++ Programming Language, 3rd Ed.*, Addison Wesley, 1997

[Sze1996] Szekely, P., *Retrospective and Challenges for Model-Based Interface Development*. In [CADUI'02], 1996

[Tan1954] Tanner, W.P., & Swets, J.A., *A decision-making theory of visual detection.* Psychological Review, 61 (6), pp. 401-409, 1954

[Tay1992] Taylor, D. L. *Computer-Aided Design.* Addison-Wesley, 1992.

[Tho2004] Thorn, A., *DirectX 9 user interfaces : design and implementation*, Wordware, 2004

[Ton2005] Tonella, P., Potrich, A., *Reverse Engineering of Object Oriented Code*, Springer, 2005

[Tri2005] Trifunovic, N. *How To Organize Template Source Code*. Project, T.C. ed., http://www.codeproject.com/cpp/templatesourceorg.asp, 2005.

[Tuf2001] Edward R. Tufte, *The Visual Display of Quantitative Information, 2nd edition*, Graphics Press, 2001

[Ulr1991] Ulrich, R. S., *Effects of health facility interior design on wellness: Theory and recent scientific research.* Journal of Health Care Design, 3, pp. 97-109, 1991

[Van1999] Vanderdonckt, J.M., Puerta, A.R., *Introduction to Computer-Aided Design of User Interfaces*, Preface of [CADUI'99], Louvain-la-Neuve, Kluwer Academic Publishers, 1999

[Van2001], Vanderdonckt, J., Florins, M., Oger, F., *Model-Based Design of Mobile User Interfaces*, Université catholique de Louvain, http://www.isys.ucl.ac.be/bchi/publications/2001/Vanderdonckt-MobileHCI01.pdf, 2001

[Van2002] Vandevoorde, D., Josuttis, N.M., *C++ Templates: The Complete Guide*, Addison Wesley, 2002

[Vol1975] Volpert, W. *Handlungsstrukturanalyse als Beitrag zur Qualifikationsforschung*. Köln: Pahl-Rugenstein, 1975

[Wal1993] Wallace, M.D., & Anderson, T.J., *Approaches to Interface Design*. Interacting with Computers, 5 (3), pp. 259-278, 1993

[Wau1965] Waugh, N.C., & Norman, D.A., *Primary memory,* Psychological Review, 72 (2), pp. 89-104, 1965

[Wel2000] Welie, M. van, Veer, G. van der, Eliens, A., *Patterns as Tools for User Interface Design,* , International Workshop on Tools for Working with Guidelines, pp. 313-324, Springer Verlag, 2000.

[Wix1990] Wixon, D., Holtzblatt, K., & Knox, S., *Contextual Design: An emergent View of System Design*., Proceedings of Human Factors in Computing Systems, CHI' 90. ACM/SIGCHI. Reading, MA: Addison-Wesley Publishing Company, Inc., http://www.cs.indiana.edu/~connelly/Usability/Local/contextual-inquiry-paper.pdf, 1990

[Wri2004] Wright Jr., R.S., Lipchak, B., *OpenGL® SuperBible*, Third Edition, Sams Publishing, 2004

[YAC2003] Yacoub,S.M., Ammar, H.H., *Pattern-Oriented Analysis and Design: Composing Patterns to Design Software Systems*, Addison Wesley, 2003