ARHITECTURI DE SISTEME DE BAZE DE DATE CARE CONȚIN MEDIUL VIRTUAL ȘI BIBLIOTECI DE OBIECTE TRIDIMENSIONALE

Cuprins

Abrevieri

API	Application Programing Interface
BD	Bază de date
DBMS	Sistem de administrare a bazelor de date – (data base management system)
DDBS	Sistem de baze de date distribuite
LDD	Limbajul de Definire a Datelor
OOA	Analiza orientată obiect – (Object Oriented Analyse)
OOD	Proiectare orientată obiect (Object Oriented Design)
OOP	Programare orientată obiect - (Oriented Object Programing)
RV	Realitate virtuală
SGBD	Sistem de gestiune a bazelor de date
SQL	Structured Query Language
TE	Tabele de entități
TÎ	Tabele de numere întregi
TR	Tabele de date reale
VRML	Virtual Reality Modeling Language
2D	Două dimensiuni
3D	Trei dimensiuni

INTRODUCERE

În primul capitol al materialului se prezintă procesul de proiectare al unei aplicații de realitate virtuală ***(RV),*** care presupune executarea unor etape progresive, pentru completarea cu detalii de realizare şi soluții optime pentru proiectele conceptuale, preliminare şi de ansamblu. În cazul proiectării iterative, etapele procesului de realizare efectivă a unui produs de grafică 3D sunt generalizate în faze comune, fiecare nivel al proiectării fiind complet dezvoltat prin analiza şi reiterarea evaluării principale a procesului şi apoi, prin rafinarea modelului.

Fiecare dintre modelele procesului de proiectare (în paşi consecutivi sau în iterații) urmăresc punctul de vedere tradițional care ilustrează etapele proiectării, urmate de elaborarea produsului grafic final. Sub presiunea dinamicii cerințelor pieței, producătorii de software de ***(RV)*** sunt nevoiți să analizeze, proiecteze, dezvolte şi să facă pregătirile de lansare pentru noile produse - în paralel, mecanism numit motor simultan sau motor concurent. Astfel se reuşeşte lansarea de produse grafice noi la intervale mici de timp.

În practică, proiectantul utilizează o combinație de modele, selectate funcție de particularitățile proiectului, precum şi de cerințele concrete ale utilizatorului. Pe parcursul elaborării unui produs de ***RV***, proiectantul are de rezolvat o multitudine de probleme legate de structura şi forma componentelor produsului, precum şi complexitatea asamblării obiectelor care compun mediul virtual.

Pentru mulți proiectanți de software de ***RV***, o mare parte a activității lor constă în definirea modelului şi elaborarea specificațiilor pentru modulele specializate care sunt componente ale proiectului. Elementele de execuție sunt repartizate unor proiectanți diferiți care trebuie să coopereze astfel încât, la sfârşitul procesului de proiectare, elementele realizate să se îmbine perfect. De aici rezultă necesitatea administrării comunicării.

Lucrarea abordează şi problema structurilor interne de date pentru produse de grafică 3D, deoarece acestea anticipează sortări foarte lungi şi operații dificile de căutare, extragere, afişare date.

Literatura de specialitate semnalează încercări de a crea baze de date universale, accesibile din orice sistem grafic. Această soluție trebuie privită cu precauție. O idee mai bună este utilizarea bazelor de date distribuite. Se recomandă evitarea utilizării unor formate de date dependente de limbaj sau de sistemul de operare.

Este adesea dificil să se determine ce grad de interactivitate este de preferat într-un sistem de ***RV***. Proiectarea unei structuri de date interne pentru un program de grafică trebuie să aibă în vedere în primul rând, anticiparea operațiilor de sortare şi căutare. O altă problemă de egală importanță, o reprezintă necesitatea de a face față prelucrărilor asupra bazelor mari de date care constituie o particularitate esențială pentru aplicațiile de ***RV***. Introducerea conceptelor prezentate în proiectarea mediilor pentru ***RV*** are efecte subtile şi pe termen lung.

Realitatea virtuală este un domeniu dinamic, cu investiții mari în echipamente periferice speciale şi implicând forță de muncă cu nivel foarte ridicat de pregătire. Este privită ca factor de avansare industrială şi socială. Acesta combină cunoştințe şi informații din matematică şi fizică, multă muncă de documentare, aplicarea standardelor şi metodologiilor internaționale.

Multe din arhitecturile referitoare la organizarea sistemelor grafice se bazează pe "sateliți grafici", care încearcă să obțină un răspuns rapid al terminalelor grafice cu

răspuns lent, prin ataşarea lor la servere puternice ("satelitul" răspunde rapid cerinţelor simple şi menţine într-o memorie tampon datele pe care le va transmite apoi sistemului central, reducând numărul de accesări ale acestuia).

Capitolul 2 a dezvoltat problematica optimizării, abordată din punctul de vedere al proiectanţilor de aplicaţii grafice. Dificultatea în proiectare este datorată necesităţii de a stabili geometrii de obiecte grafice complexe şi traiectorii complicate ale acestora. S-a exemplificat aplicarea algoritmilor de căutare aleatorie investigaţi pentru produsul de reconstrucţie virtuală Cetatea de Scaun Suceava.

Multe sisteme grafice sunt produse software foarte mari, greu de întreţinut şi depanat. Această dificultate a fost depăşită prin utilizarea proiectării orientate obiect, care pune accent mai puţin pe structura de date şi structura procedurilor, cât mai ales pe descrierea obiectelor, rolul pe care acestea îl au în sistem şi pe natura comunicaţiei dintre ele. Transmiterea mesajelor între obiectele grafice exploatează posibilitatea folosirii asocierii între entităţi.

În Capitolul 3 se prezintă conceptele de „bază de date" şi „sistem de gestiune a bazelor de date", cu exemple de reprezentări de date pentru modelele reţea, relaţional şi SQL, în cazul concret al aplicaţiei de reconstrucţie virtuala a Cetăţii medievale de la Suceava. În acest capitol s-a investigat oportunitatea utilizării modelelor relaţionale la definirea şi manipularea datelor care compun obiecte grafice complexe. Acestea sunt recomandate pentru: simplitatea conceptelor şi a schemei de definire, nivelul ridicat de independenţă a datelor, limbajul de manipulare de nivel înalt (SQL), integritate şi confidenţialitate asigurate, manipulare date puternic integrate.

O consecinţă importantă a lucrului cu staţii de lucru grafice este faptul că interfeţele om-maşină au devenit din ce în ce mai performante, ceea ce face ca, deseori, un sistem de calcul să fie judecat după calităţile şi defectele interfeţelor sale. Aceste interfeţe permit unui utilizator nespecialist să acceseze aplicaţii complexe. În mod special interesează modul în care un ***SGBD*** poate gestiona datele grafice şi interfeţele. Unul dintre cele mai puternice limbaje structurate pentru interogarea bazelor de date relaţionale îl constituie SQL, devenit standard pentru SGBD.

Standardizarea limbajului SQL este acceptată de marea majoritate a SGBD, care recunosc principalele instrucţiuni ale SQL. ANSI recunoaşte oficial SQL ca standard, acceptând din acesta următoarele aspecte: definirea, interogarea şi manipularea datelor, procesarea tranzacţiilor, integritatea informaţiilor complexe, joncţiunile externe. Majoritatea marilor producători de ***SGBD*** furnizează propriile extensii ale limbajului SQL, asigurându-şi astfel exclusivitatea (Oracle, Sybase, IBM, Informix, Microsoft).

Aplicaţiile de realitate virtuală au particularităţi care influenţează proiectarea bazelor de date, precum şi structura ***SGBD,*** printre care şi aceea că datele manipulate sunt foarte complexe, eterogene şi aflate în corelaţii complicate între ele. S-au prezentat principalele caracteristici ale unei aplicaţii de ***RV*** care interesează proiectarea bazelor de date.

Aspectul eterogen al datelor se adaugă dificultăţilor induse de volumele mari de date care se prelucrează. O astfel de aplicaţie trebuie să permită selecţii extrem de rapide de imagini stocate, ceea ce combină tehnicile de căutare multicriterială, de acces optimizat la bazele de date, de tratare imagini 3D etc.

Capitolul 4 prezintă paradigma orientării pe obiecte, care susţine că o entitate din lumea reală poate fi modelată ca un obiect (instanţă a unei clase), care are un număr de atribute (proprietăţi) şi servicii (operaţii sau proceduri) aplicabile obiectului.

Această reprezentare a dus la apariția unui nou stil de programare. Orientarea pe obiect comută concentrarea procesului de la proceduri la obiecte, definite drept unități de sine-stătătoare care includ atât datele, cât şi procedurile care acționează asupra datelor.

În Capitolul 5 se prezintă modelul bazelor de date obiectuale, care se bazează pe conceptele orientării pe obiecte (reprezentarea datelor prin clase, atribute şi servicii etc.) şi pot fi stocate / regăsite de aplicații în forma naturală, fără a fi modificate pentru stocarea în tabele relaționale. Acestea asigură performanțe bune la lucrul în timp real, deoarece cele mai multe aplicații moderne prezintă arhitecturi client-server şi sunt programate în termeni de obiecte. De asemenea, se pretează la procesarea distribuită, sunt foarte rapide în cazul cererilor complexe şi acceptă structuri de date heterogene.

Se prezintă o paralelă între modelul relațional (arată ca un tabel de linii şi coloane - obiect 2D) şi modelul orientat pe obiecte, care respectă caracteristica reală a obiectelor complexe 3D (este ideal pentru Internet, jocuri video, aplicații multimedia, comunicații). Denumirea corectă ar trebui să fie "bază de obiecte" şi nu „bază orientată obiectual", deoarece scopul nu este de a stoca, manipula şi returna datele înglobate în obiecte, ci stocarea, manipularea şi returnarea chiar a obiectelor.

Obiectele unui model grafic sunt entitățile asupra cărora operează comenzile aplicației. Aceste obiecte pot fi obiecte simple sau obiecte complexe, agregate. Obiectele sunt caracterizate prin structură, caracteristici şi comportament. Componentele unui obiect grafic sunt primitivele grafice. Atributele obiectelor din model caracterizează aspectul, forma, dimensiunile. Principalul tip de atribut este atributul grafic. Atributele grafice pot fi: identificare, aspect, forma, poziție. Atributele grafice precizează caracteristicile grafice legate de contextul de prezentare a scenei.

A fost tratat comportamentul obiectelor, definit ca evoluția spațială şi temporală abstractă a obiectelor dintr-o scenă de obiecte. Asocierea unui comportament la un obiect din scena de obiecte concretizează comportamentul la un obiect, poziție şi evoluție. Fiecare obiect, simplu sau complex, din cadrul unui agregat, are o evoluție dependentă de propria sa definiție a comportamentului, dar şi de evoluția agregatului din care face parte.

S-au detaliat câteva tipuri de traiectorie cunoscute în proiectarea aplicațiilor de realitate virtuală: traiectorie moştenită de la agregatul părinte, punct, polilinie, ciclică, cu regulă implicită şi graf orientat.

În capitolul 6 se definesc teoretic arhitecturile experimentale ale bazelor de date de mari dimensiuni, de tip distribuit. Acestea sunt definite ca fiind structuri informatice care permit culegerea, prelucrarea, analiza şi difuzarea unor colecții de date heterogene. Se definesc şi ca fiind colecții de mai multe baze de date corelate logic, fiecare fiind asociată unui nod al unei rețele şi unui mecanism de acces, care face această colecție transparentă pentru utilizator.

Dezvoltarea bazelor de date distribuite a fost puternic impulsionată de apariția stațiilor grafice şi a calculatoarelor paralele. Un sistem cu prelucrare distribuită poate fi un sistem expert extensibil, adaptabil, fizic distribuit şi logic integrat. Noile direcții de cercetare cuprind şi aspecte specifice inteligenței artificiale. Bazele de date relaționale distribuite vor fi înlocuite cu baze de cunoştințe distribuite şi cu baze de date distribuite orientate obiect.

Diversitatea şi complexitatea informațiilor care descriu mediile virtuale se caracterizează prin: multitudinea de surse ale datelor şi de factori de decizie logică, dispersia geografică a surselor de date, necesitatea agregării şi/sau descentralizării informațiilor pe diferite nivele şi clase de evenimente, dinamica mare a cererilor de

informare pe diferite nivele ierarhice şi clase de evenimente şi în diverse areale geografice şi autonomia cooperativă a surselor şi ţintelor. Aceste caracteristici impun proiectarea unor sisteme de gestiune a bazelor de date distribuite, deschise şi evolutive. Soluţia propusă pentru tratarea problematicii evenimentelor virtuale este abordarea ierarhică, recomandată pentru sistemele mari, complexe.

Sistemele de gestiune a bazelor mari de date reprezintă sisteme software specializate în stocarea şi prelucrarea volumelor mari de date. Termenul de *"bază mare de date"* se referă la volumul, cantitatea datelor de prelucrat şi modul de organizare al acestora pe suportul fizic de memorare, iar gestiunea bazelor mari de date este acţiunea de memorare şi prelucrare a acestor volume mari de date.

Din punct de vedere al limbajelor pe care le utilizează, sistemele de gestiune pentru baze mari de date care utilizează *limbaje gazdă*, realizează activităţile de creare, actualizare şi interogare bază de date utilizând limbajele de nivel înalt sau extensii ale acestora; limbajele de nivel înalt oferă posibilităţi complexe, dar au dezavantajul că formularea cerinţelor este dificil de implementat. Sistemele de gestiune baze mari de date cu *limbaj autonom*, folosesc un limbaj independent de limbajul gazdă şi au o independenţă totală faţă de platforma pe care rulează; prezintă avantajul portabilităţii mari şi a unei simplităţi sporite în formularea cerinţelor, motiv pentru care sunt sistemele cu cea mai mare utilizare.

Concepţia, elaborarea, exploatarea şi întreţinerea unui sistem distribuit de baze mari de date, aşa cum sunt sistemele de ***RV***, impun concentrarea unor resurse umane şi materiale importante pentru o perioadă îndelungată de timp. Aceste greutăţi sunt generate atât de dimensiunea şi complexitatea problemelor de rezolvat, cât şi de faptul că tehnologiile specifice sunt încă limitate. În acelaşi timp, cerinţele aplicaţiilor de ***RV*** privind criteriile de performanţă, fiabilitate, interfaţă de utilizare şi raportul performanţă/cost, reprezintă tot atâtea elemente de dificultate puse în faţa celor care proiectează o aplicaţie de ***RV.***

În adoptarea unui sistem de baze de date distribuite, trebuiesc analizate şi dezavantajele. Acestea pot ridica atât probleme de sincronizare, de consistenţă şi integritate a datelor, cât şi probleme legate de controlul accesului.

O problemă importantă pentru sistemele de ***RV*** este eliminarea paralelismelor în actualizarea datelor şi redondanţei în înmagazinarea acestora, iar prin multiplicarea criteriilor de regăsire şi micşorarea complexităţii operaţiei de identificare, se poate scurta timpul de răspuns la cererile de regăsire. O primă accepţiune este aceea că, fiind sisteme deschise, sistemele de baze de date pentru medii virtuale necesită un mediu de dezvoltare de aplicaţii bazat pe standarde de interfaţă ce permit portabilitatea aplicaţiilor, portabilitatea utilizatorului şi inter-operabilitatea.

Integritatea datelor într-o bază mare de date relaţională trebuie menţinută mai ales în timpul accesului multiplu la date. Concurenţa reprezintă procesul de distribuire a resurselor între mai mulţi utilizatori sau programe şi module de aplicaţie, ce rulează în acelaşi timp. Pentru a asigura integritatea datelor, se impune ca modificările să se facă cât mai rapid, dacă se poate chiar simultan. În acest sens, se recomandă ca un subsistem de interfaţare să asigure modificarea datelor numai în versiunea activă.

Realitatea virtuală este un domeniu foarte dinamic, conducând la investiţii mari în echipamentele speciale şi implicând multă forţă de muncă cu nivel ridicat de pregătire, pentru elaborarea şi implementarea noilor tehnologii. Ea este în multe cercuri privită ca factor prim pentru avansarea industrială şi socială şi este un suport atractiv pe termen lung.

1. STADIUL ŞI TENDINȚELE GENERALE ALE PROCESULUI DE PROIECTARE PENTRU APLICAȚII DE REALITATE VIRTUALĂ

1.1. Modele de proiectare pentru aplicații de realitate virtuală

Principial, procesul de proiectare al aplicațiilor de realitate virtuală (*RV*) se poate prezenta printr-o diagramă formată din mai multe blocuri, corespunzătoare fazelor care se parcurg în paşi consecutivi (*Figura 1.1.*):

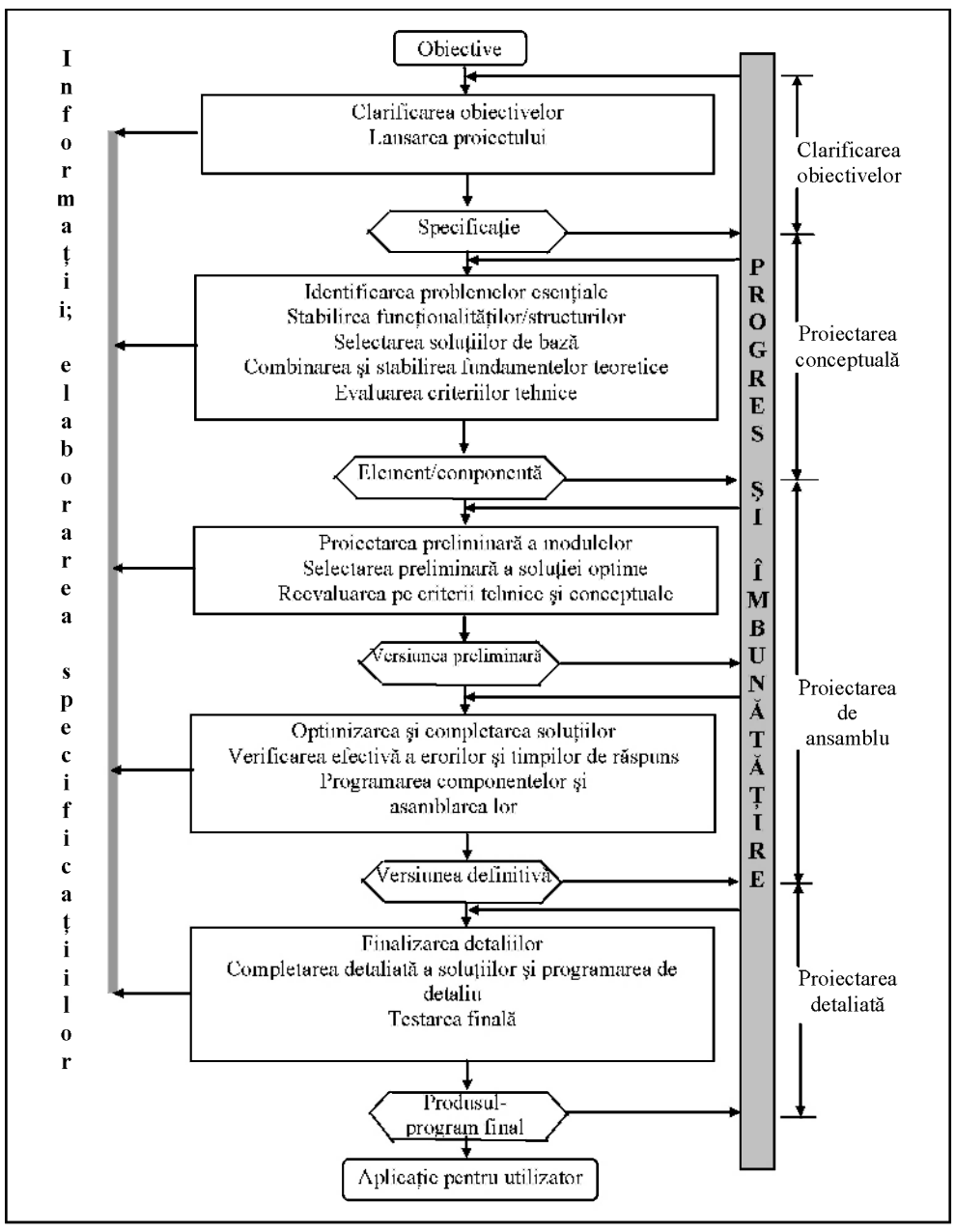

Figura 1.1. Procesul de proiectare în etape consecutive

Fazele consecutive rezumate în *Figura 1.1. [12]* sunt următoarele:

- *specificarea obiectivelor* (informații despre cerințe şi restricții);
- *proiectarea conceptuală*, (stabilirea funcțiilor, identificarea şi dezvoltarea soluțiilor; cerințele speciale ale unui produs de ***RV*** sunt impuse de necesitățile utilizării sale şi implică procese decizionale complexe şi multă creativitate);
- *proiectarea de ansamblu (preliminară)*, (soluția conceptuală este dezvoltată cu detalii suplimentare, se fundamentează soluțiile tehnice şi operaționale, se clasifică aspectele insuficient tratate);
- *proiectarea detaliată*, (componentele şi modulele se specifică prin detalii).

În *Figura 1.1.* se delimitează fiecare etapă (secvență) a procesului. În practică, etapele nu sunt întotdeauna clar respectate şi este necesară parcurgerea unor iterații.

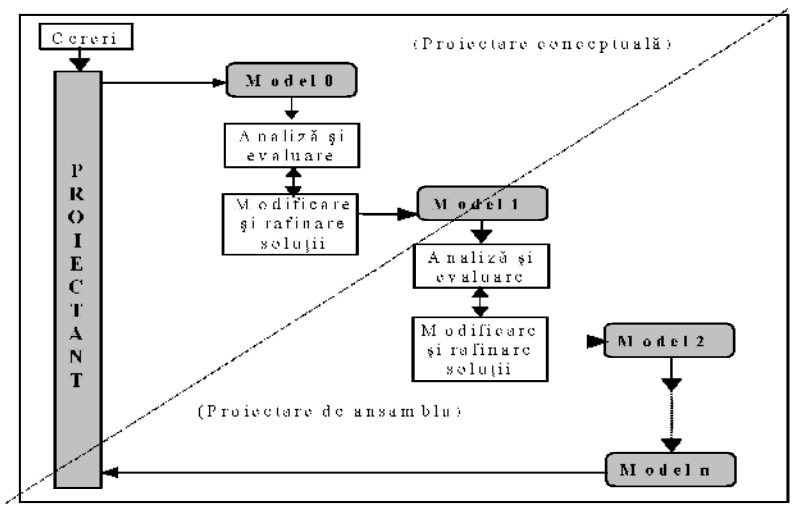

Figura 1.2. Proiectarea preliminară iterativă

Procesul de proiectare presupune executarea unor etape progresive, pentru completarea cu detalii de realizare şi soluții optime proiectul conceptual şi preliminar. În cazul proiectării iterative, diversele etape ale procesului de realizare efectivă a unui produs de grafică sunt generalizate în faze comune, fiecare nivel al proiectării fiind complet dezvoltat prin analiza şi reiterarea evaluării principale a procesului şi apoi, rafinarea modelului (*Figura 1.2.*).

La debutul etapei de proiectare, soluția este propusă de beneficiar. Aceasta este evaluată şi, conform posibilităților tehnice, conceptuale şi operaționale reale, adaptată la cerințe. Dacă soluția propusă de beneficiar este inacceptabilă, ea este modificată. Acest proces este repetat până când produsul grafic ajunge la o formă care poate fi dezvoltată în adâncime şi etapele de programare efectivă pot să demareze, dar – în egală măsură, satisface beneficiarul. Proiectul este abordat la primul nivel de detaliere, iar eventualele transformări şi modificări repetate conduc spre nivele de detaliere suplimentare. În final, se ajunge la definitivarea fiecărei componente şi asamblarea tuturor modulelor produsului grafic complex.

Fiecare dintre modelele procesului de proiectare (în paşi consecutivi sau în iterații) urmăresc punctul de vedere tradițional care ilustrează etapele proiectării, urmate de elaborarea produsului grafic. Sub presiunea dinamicii cerințelor pieței, producătorii de software de realitate virtuală sunt nevoiți să analizeze, proiecteze,

dezvolte şi să facă pregătirile de lansare pentru noile produse - în paralel. Acest mecanism este cunoscut sub denumirea de motor simultan sau motor concurent şi datorită acestuia, se reuşeşte lansarea de produse grafice noi la intervale mici de timp.

Modelele geometrice necesare proiectării unei aplicații de ***RV***.pot fi obținute pe diferite căi. Dacă obiectivul propus este simplu, imaginea unei anumite componente este uşor de obținut, dar în cele mai multe cazuri, imaginea obiectului de reprezentat este complexă, fiind formată din foarte multe detalii. Elementele de execuție pot fi repartizate unor proiectanți diferiți care însă trebuie să coopereze, astfel încât, la sfârşitul procesului de proiectare, elementele realizate să se îmbine perfect. De aici rezultă necesitatea comunicării, a faptului că modificarea unei componente elaborate de un proiectant, trebuie comunicată şi celorlalți membri ai echipei.

La nivel elementar, acestea sunt utilizate pentru înregistrarea şi prelucrarea ideilor conceptuale şi furnizarea modelului de bază necesar următoarelor etape. Un proiect este rareori încredințat unui singur executant şi de aceea, modelele de proiectare sunt impotante în comunicarea între cei ce participă la procesul de proiectare şi cei implicați în dezvoltarea şi utilizarea ulterioară a produsului final.

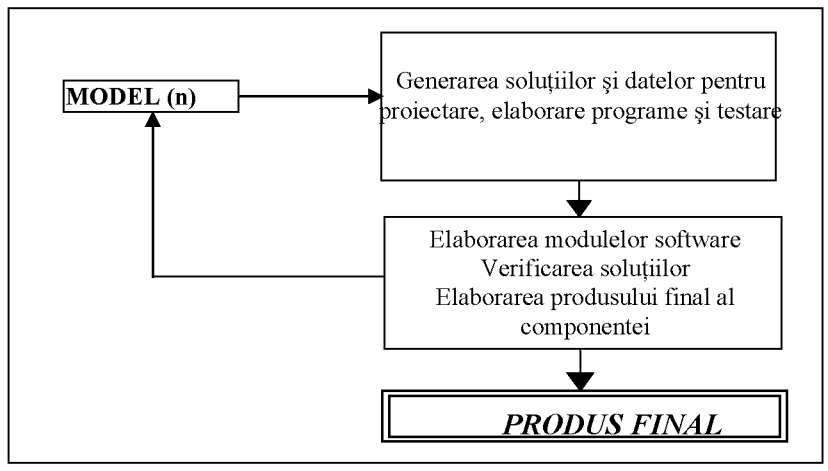

Figura 1.3. Proiectarea detaliată iterativă

Specificațiile fazelor generează de obicei noi modele al componentei, care se comunică din nou echipei care lucrează „în aval" pentru detaliere, secvență ce se repetă până la detaliile cele mai amănunțite, aşa cum se ilustrează în *Figura. 1.3.* Modelul procesului de proiectare pentru aplicații de ***RV***. implică o mare varietate a reprezentărilor, utilizând expresii ca "dezvoltarea preliminară a componentei" şi "completarea detaliată a modelului". În practică, proiectantul utilizează o mulțime de modele diferite care depind de particularitățile proiectului, precum şi de cerințele concrete ale utilizatorului. Pe parcursul elaborării unui produs de ***RV***, proiectantul are de rezolvat o multitudine de probleme legate de structura şi forma componentelor produsului, precum şi complexitatea asamblării obiectelor care compun mediul virtual.

Modelarea proprietăților de tip „formă" şi „structură" are o importanță particulară în aplicațiile de ***RV***. Ca metode de reprezentare se utilizează atât cele de grafică tradițională, cât şi cele de sinteză grafică computerizată. Pentru mulți proiectanți de software de ***RV***, o mare parte a activității lor constă în definirea modelului şi elaborarea specificațiilor pentru modulele specializate care sunt componente ale proiectului. Acestea pot fi realizate convențional folosind modelul transformărilor, aşa cum se prezintă în *Figura 1.4.*

Interesează asamblarea elementelor primare, modul în care aceste elemente sunt conectate împreună şi legăturile între părţi. Această conectare este numită ingineria sistemului. Sunt definite două nivele de utilizare *[8]* şi anume:

- *nivelul de bază*, care utilizează calculatoarele pentru asistarea automată a sarcinilor asemănătoare şi repetabile ale proiectării aplicației/componentei/modulului şi generarea listei componentelor proiectului;
- *nivelul avansat*, care furnizează tehnici specifice şi oferă proiectantului facilități de lucru în echipe multidisciplinare şi comunicare pe verticală şi orizontală.

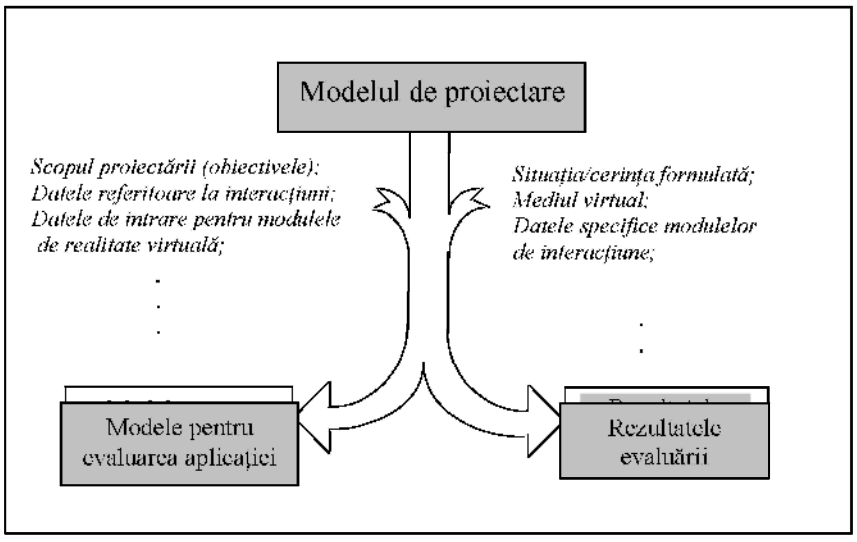

Figura 1.4. Modelul transformărilor în proiectarea aplicațiilor de realitate virtuală

Se prezintă utilizarea tipurilor de modele în *Figura 1.5.*

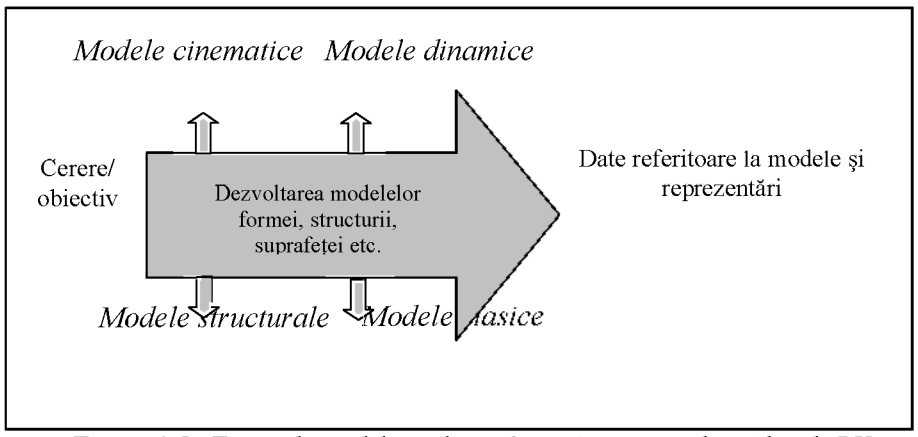

Figura 1.5. Tipuri de modele utilizate în proiectarea aplicațiilor de RV

Un sistem de ***RV***, este alcătuit din:

- *componente hardware*: calculatorul şi echipamentele periferice asociate, între care: tastatura, cursorul, creionul optic, digitizorul, scanner-ul, display-ul grafic, imprimanta, plotter-ul, perifericele specifice domeniului ***RV*** (videocască, mănuşi de date, mouse spațial, combinezon de date, scaun cu vibrații etc);
- *componente software*: sistemul de operare şi programele de aplicație; interfața cu utilizatorul; sistemul de comunicație.

- *date*: structura, organizarea şi accesul la datele create şi prelucrate;
- *activitatea şi cunoştinţele* utilizatorului: realizarea interactivităţii între utilizator şi sistem, prin interfaţa fizică; memorarea – prelucrarea – regăsirea datelor; elaborarea în timp real a unor ieşiri utilizabile.

Funcţia de memorare, prelucrare şi regăsire a datelor include subfuncţiile de prezentare date (pe display, monitoarele binoculare, în căşti audio etc.) şi introducere date (tastatură, mouse, mănuşi de date etc.), precum şi subfuncţiile de analiză, calcule şi prelucrări pe bază de algoritmi şi modele. Regăsirea şi memorarea asociativă se referă la efectuarea operaţiilor de introducere/extragere date, conform unei structuri pe care o definesc asocierile „identificator–mulţime" sau „identificator–element". Postprocesarea realizează convertirea rezultatelor prelucrării, într-o reprezentare conform altei structuri, specifice altei faze/subfuncţii necesare utilizatorului.

1.2. Fluxul proiectării unei aplicaţii de realitate virtuală

Diagrama generală a fluxului în proiectarea unei aplicaţii de ***RV*** este:

1. *definirea modelului*, care include şi specificarea elementelor geometrice pentru modelarea componentelor formelor elementare şi complexe;
2. *manipularea modelului*: mutarea, copierea, ştergerea, multiplicarea sau alte modificări ale elementelor modelului proiectat;
3. *generarea graficii* : generarea imaginilor modelului proiectat;
4. *utilizarea interactivă*: la introducerea de către utilizator a comenzilor trebuiesc prezentate informaţii conforme cu modul de utilizare a funcţiilor;
5. *managementul bazelor de date*: managementul colecţiilor şi datelor ce alcătuiesc bazele de date (administrarea, organizarea, stocarea, accesul la date etc.);
6. *structurarea bazelor de date*: utilizarea tuturor tipurilor de structuri;
7. *elaborarea modulelor software interne*: realizarea acelor componente ale aplicaţiei care conţin elemente software care nu pot fi modificate de către utilizator, dar care pot fi apelate pentru generarea altor module ale aplicaţiei;
8. *înglobarea utilitarelor*: utilizarea componentelor software care nu pot afecta direct modelul proiectat, dar care modifică sistemul pe diferite căi (de exemplu, selectarea standardului de culoare pentru afişare sau unitatea de măsură).

Instrumentele sunt grupate în clasele:

a) analiza: ca instrumente de analiză se folosesc: calculul proprietăţilor de bază (arii, volume, momente de inerţie etc.); generarea şi utilizarea metodelor de element finit (propagarea căldurii, deformări elastice etc.); verificarea relaţiilor între părţi (îmbinarea părţilor; coerenţa în funcţionarea subansamblelor); instrumentele de analiză acţionează asupra modelelor geometrice existente, conducând la corecţiile necesare.

b) validarea, care se poate face: static (prin calcule specifice de evaluare pentru fiecare componentă) şi dinamic (simulând funcţionarea sistemului proiectat, asemănător unei secvenţe de film tehnic de animaţie);

c) documentarea constă în elaborarea documentaţiei operaţionale însoţitoare sub formă de: proiecţii, mesaje, instrucţiuni; liste de facilităţi, funcţiuni specifice, etape de parcurs, timpi de acces, necesar de periferice, configuraţie etc.

Arhitectura generală a proiectării unui sistem de ***RV*** este redată în *Figura 1.6.* Un produs de ***RV*** nu înseamnă doar îmbinarea unei colecţii de programe. Aceste aplicaţii trebuie să aibă o bună organizare internă.

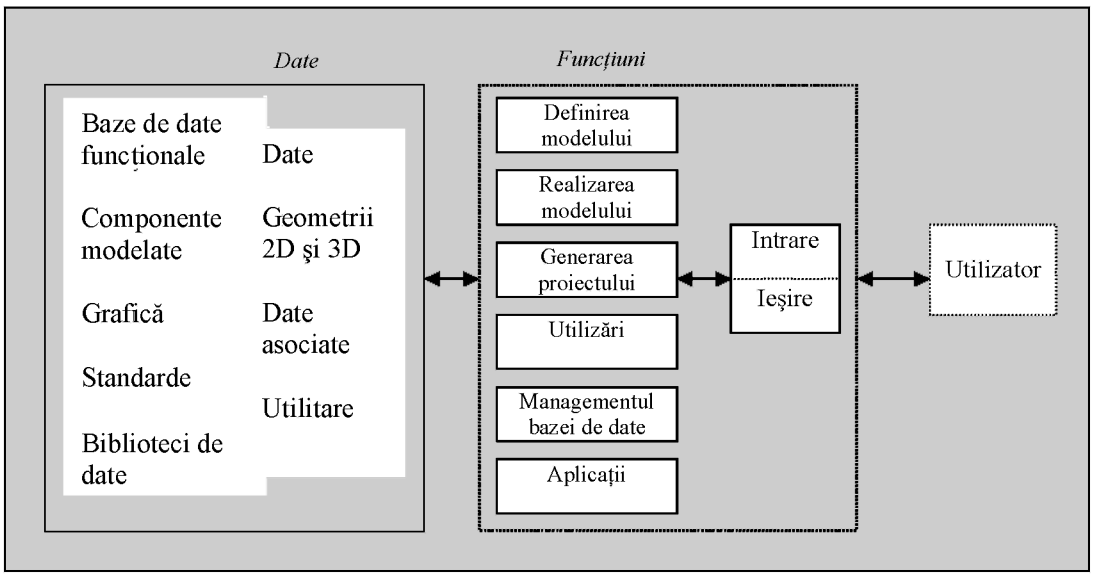

Figura 1.6. Arhitectura generală a unui sistem de realitate virtuală

Câteva din problemele proiectării unui astfel de sistem sunt următoarele:

- Cât de bine va asigura sistemul grafic procesele de imersiune în mediul virtual?
- Care sunt considerentele economice şi comerciale care afectează proiectarea aplicației? De exemplu, care sunt rezultatele maxime sau imediate obținute prin lansarea de piață a aplicației?
- Ce subsisteme sunt necesare?
- Care sunt structurile de date necesare?
- Cum ar trebui să comunice utilizatorul cu sistemul? Care ar fi limbajul de comandă optim?
- Ce programe de aplicație ar trebui scrise? Care este cel mai potrivit limbaj de programare? Ce software de bază este necesar înainte ca aceste programe să fie scrise?
- Ce tip de hardware, inclusiv periferice, este necesar? Care sunt interfețele hardware necesare şi cum pot fi conectate?
- Cât de rapid poate fi instalat sistemul grafic în cadrul aplicației şi cât poate fi extins?
- Ce pregătire a utilizatorului este necesară?
- Ce documentație va fi necesară?
- Care este pregătirea minimă necesară întreținerii sistemului?

Este esențial ca în proiectarea produsului grafic, structurile de date şi interfața cu utilizatorul să fie considerate un tot unitar. În particular, gradul de interacțiune între ele trebuie să fie stabilit încă din etapele inițiale ale proiectării aplicației, deoarece utilizarea acestor interacțiuni reciproce permite folosirea redusă a algoritmilor deterministici în programele de aplicație.

În decursul documentării, s-a observat aceasta în programele de monitorizare acces la traseele configurabile de vizitare în mediul virtual şi în programele de

planificare a spațiului arhitectural. Acestea au ocupat cca. 70% din funcțiuni. Programele interactive necesită date structurate intern mai flexibil şi care pot să se extindă în dimensiune sau complexitate. Această cerință impune să se folosească limbaje de programare diferite, în funcție de tipul şi gradul de interacțiune al componentelor *[19]*.

Proiectarea structurilor interne de date pentru un sistem de ***RV*** necesită şi anticiparea sortărilor foarte lungi şi a operațiilor dificile de căutare.

O problemă la fel de importantă este cerința de a distribui colecțiile foarte mari de date, ca părți ale unor module de grafică interactivă.

Sunt semnalate încercări de a crea baze de date universale, ce pot fi accesate de orice program al unui sistem grafic. Această soluție trebuie utilizată cu precauție şi numai dacă este necesar. O idee mai bună ar fi utilizarea bazelor de date distribuite. Multe procese de acces sunt inerent secvențiale şi adesea este suficientă separarea formatului datelor pentru fiecare etapă a accesului la ele.

Se recomandă evitarea utilizării unor formate dependente de limbaj sau de sistemul de calcul. În proiectarea unei aplicații de ***RV***, structura sa de date şi interfața trebuie să fie înglobate. Din acest motiv, gradul de interactivitate trebuie să fie determinat într-un stadiu timpuriu al proiectării. Programele interactive solicită structuri de date interne flexibile, care se pot dezvolta pe măsură ce creşte mărimea sau complexitatea.

Apar adesea constrângeri care forțează proiectanții unui astfel de sistem să aleagă o anumită configurație hardware, fără să beneficieze de un software corespunzător. Situațiile cele mai frecvente sunt cele în care echipamentelor le lipsesc articole importante din software-ul de bază, spre exemplu, un compilator pentru limbajul dorit sau un sistem avansat de depanare. Echipamentele având configurații speciale sunt potențial puternice, dar nu sunt operaționale dacă nu au software specializat pentru administrarea lor.

Orice sistem care include intercomunicarea între procesoare şi/sau periferice prezintă probleme severe de programare şi verificare. Acest lucru a afectat succesul a multe din modulele grafice pentru aplicații de ***RV***. Problema asamblării generale a modulelor software funcționale este ignorată uneori în procesul de finalizare şi este oferită adesea de către membri ai echipei de proiectanți cu relativ puține cunoştințe privind ansamblul aplicației. Proiectanții unui astfel de sistem neglijează acest aspect, care contribuie ca aplicația să fie un succes.

Soluția satisfăcătoare de a realiza producție competitivă de software de aplicație pentru sisteme de ***RV*** cere o integrare a întregului proces de proiectare, programare şi administrativ. Fără interfața organizațională nu pot fi găsite soluții viabile. Pot apare situații în care este recomandat să se treacă printr-o perioadă tranzitorie care să permită programatorilor cu experiență să păstreze unele din "secretele personale" în zone confidențiale la care numai ei să aibă acces. Această atitudine contrazice scopurile principale ale proiectului, de cele mai multe ori necesitatea cooperării fiind foarte importantă. Totuşi, aceste probleme trebuie admise ca fiind reale şi dificil de soluționat. Aceste facilități ar trebui să decurgă dintr-o legătură naturală între producția de software de ***RV*** şi nevoile generale de proiectare, care ar aduce beneficii cum ar fi:

- creşterea calității aplicațiilor realizate ;
- selectarea şi utilizarea mai eficientă a modelelor cele mai adecvate;
- micşorarea timpului de elaborare a unei aplicații;
- economisirea resurselor (muncă şi costuri);

- omogenizarea tehnologiilor folosite;
- actualizarea şi dezvoltarea ulterioară mai facilă a aplicației realizate.

Toate acestea ar putea face firmele producătoare de software mai eficiente şi competitive pe piață. Acestea sunt atât obiective cu prioritate sectorială, cât şi la nivelul companiilor. Mijloacele publice necesare sprijinirii dezvoltării tehnologiilor moderne sunt următoarele:

1. promovarea achiziției de cunoştințe.
2. finanțarea unor centre puternice de cercetare.
3. oferirea de stimulente utilizatorilor versiunilor „beta".

Nici un standard internațional acceptat nu a fost încă specializat pentru utilizarea mutuală a tehnicilor de ***RV*** şi a facilităților lor şi nici sisteme candidate pentru astfel de standarde nu au fost încă elaborate. Câteva interfețe orientate sau sisteme de procesare grafică au o utilizare larg răspândită în multe țări şi servesc ca baze pentru schimburi internaționale în sistemele construite în jurul lor *[21]*.

Se impune promovarea şi dezvoltarea cooperării internaționale în scopul dezvoltării şi utilizării sistemelor de ***RV***, precum şi corelarea cu alte domenii de cercetare în creşterea calității vieții. Paşi importanți se fac spre o distribuție a capacităților de manufacturare la subcontractanții specializați. Proiectarea devine particularizată şi poate fi realizată mai mult local. Componentele principale ale unei astfel de colaborări ar putea să fie:

- selectarea şi studierea câtorva arii pentru utilizarea generală a sistemelor de ***RV*** (protecția mediului, sursele de apă, părți maşină, genetică, astronomie etc.);
- stabilirea unui set de standarde, alegerea protocoalelor de comunicare şi interfețelor grafice ale sistemelor şi pregătirea unor documentații utilizator;
- crearea unui cadru de lucru financiar şi legislativ pentru cazul particular al sistemelor de ***RV***, în rețelele internaționale;
- implementarea, operarea, exploatarea, evaluarea şi testarea aplicațiilor de ***RV*** în mod public.

Se consideră că o astfel de cooperare ar putea să aibă o influență benefică în standardizarea internațională şi transferul mutual al rezultatelor.

1.3. Aspecte generale ale utilizării sistemelor grafice în aplicații de realitate virtuală

În fazele de proiectare, interactivitatea survine la fiecare ciclu şi implică calculatorul într-o foarte mare cantitate de calcule de evaluare în urma cărora proiectantul face modificări ale structurilor de date; fiecare pas tipic implică numai un singur element al proiectării şi astfel, precizia finală este mult mai mare. Câteva dintre caracteristicile şi atributele majore ale sistemelor grafice prin care poate fi măsurată calitatea lor de a deservi aplicații de ***RV*** sunt:

Structurile de date: aplicațiile pentru sisteme de ***RV*** solicită întotdeauna structuri complexe bazate pe date heterogene, care pot fi uşor implementate cu ajutorul structurilor relaționale. Structurile simple sunt adecvate pentru tehnicile simple de stocare a datelor. Dacă formele relaționale ale organizării datelor nu sunt disponibile, interactivitatea este practic imposibilă, de vreme ce nu poate fi conservată continuitatea dintre o sesiune de lucru şi următoarea.

Flexibilitatea intrărilor/ieşirilor: sistemul trebuie să aibă capacitatea de a

administra o largă varietate de periferice, într-un mod care permite, dar nu forțează, ca toate perifericele şi dispozitivele de manipulare a datelor să fie active. Dispozitivele interactive complexe pun probleme speciale şi nu este recomandat să se restricționeze comportamentul lor, doar pentru a realiza compatibilitatea cu alte periferice.

Robustețea: este importantă, deoarece sistemele interactive, în special cele cu acces multiplu, suferă de o fragilitate care se manifestă ca frecvente "căderi".

Securitatea: este extrem de importantă în acest domeniu. Datele trebuie să fie bine asigurate față de posibilitatea de acces neautorizat şi mai ales, în cazul aplicațiilor cu acces multiplu.

Compatibilitatea grupurilor cu acces multiplu: programele rulate în cadrul unui grup sunt adesea mult mai uşor dezvoltate şi sunt testate mai sistematic prin rularea lor în mod "*grup*". Pe de altă parte, în exploatare, sistemele "*grup*" ar trebui să autorizeze anumite variante, alocând un acces limitat al perifericelor interactive şi asigurându-se că programele pot fi rulate fără alterări de natură interactivă sau de subordonare unui grup.

Rolul stațiilor grafice în aplicațiile de ***RV*** este un subiect controversat. Pe parcursul documentării, s-au găsit surse care considerau termenii "stație grafică" şi "grafică de sinteză" ca fiind sinonimi. S-a încercat rezolvarea acestei controverse prin descompunerea în următoarele două probleme:

- Ce rol au stațiile grafice în sistemele de ***RV***?
- Care sunt necesitățile proiectanților de aplicații de ***RV*** pe care stațiile grafice le pot rezolva?

Aproape toate aplicațiile de ***RV*** sunt foarte strâns legate de grafică, dar şi de comunicare, ceea ce presupune înmagazinare de informații. Costul pregătirii, modificării şi înmagazinării imaginilor este extrem de ridicat. O tehnică foarte utilizată este scanarea datelor vizuale.

Tehnicile complexe de sinteză grafică folosite la definirea şi editarea geometriilor complexe sau reprezentarea relațiilor tipologice între datele de intrare sunt absolut obligatorii pentru elaborarea aplicațiilor de realitate virtuală. Exemplele din literatura de specialitate includ mai ales specificații pentru geometriile plane şi spațiale simple. Studiul utilizării producției grafice în aplicații de ***RV*** a condus la formularea unor observații interesante. S-a constatat că există în cadrul firmelor de software un mare interes pentru producția grafică, dar echipamentul pentru sistemele grafice este scump, iar utilizarea sa este pretențioasă. *[3].*

Echipamentul hardware pentru ***RV*** este rareori proiectat în conlucrare cu programatorul produsului grafic, iar concepția sistemelor grafice arată deseori o lipsă de înțelegere a cerințelor specifice domeniului ***RV***. Se impune necesitatea realizării unor dispozitive hardware mai simple în exploatare şi a unui software de bază mult mai flexibil; există vizibile semne de progres în aceste direcții.

Un subiect căruia i s-a acordat o atenție deosebită este maniera în care utilizatorii de aplicații de ***RV*** pot să comunice cu sistemele grafice. Mediul în care lucrează utilizatorul unui sistem este impus de sistemul de operare al calculatorului pe care îl foloseşte şi de limbajele de programare ale modulelor de ***RV*** ce rulează pe respectivul sistem. Acesta este un aspect important deoarece, fără o bună comunicare, utilizatorul va avea dificultăți în exploatarea sistemului şi acesta va fi mai puțin eficient. Probleme deosebite apar la apelarea unor module grafice, pentru care cerințele în procesul de comunicare sunt deosebit de severe.

Termenul de "comunicare om-maşină" nu se aplică numai sistemului interactiv.

Cele mai multe dintre modulele grafice înglobate în aplicaţii de ***RV*** sunt interactive. Fluxul de comunicaţii poate fi abordat la mai multe nivele. Fluxul de prelucrări la acest nivel poate fi grupat prin procesare lexicală în "enunţuri elementare", tipurile cele mai importante de "enunţuri elementare" fiind:

- declaraţii pentru controlul informaţiei, echivalente cuvintelor sau selecţiilor de meniu sau cuvintelor-cheie;
- valori explicite (numerice sau aparţinând şirurilor de caractere sau fişierelor vocale);
- nume, care identifică sau selectează părţi ale unui model intern.

Aceste "enunţuri elementare" sunt apoi grupate pentru a forma declaraţii prin procesarea sintactică.

Funcţie de tipul de procesare impus de cerinţele produsului grafic, modul de lucru efectiv poate fi:

- *lucrul cu format fix*, în care nu există un control interactiv al informaţiei şi în care numărul şi tipurile tuturor "enunţurilor elementare" din fiecare declaraţie este fix. Acest tip este utilizat în module mici şi cu un scop special, bine definit;
- *lucrul cu format variabil*, în care fiecare declaraţie începe cu un "enunţ elementar" de control şi valoarea acestuia determină numărul şi tipurile enunţurilor rămase; termenii "operator" şi "operanzi" sunt aplicabili pentru cele două părţi; lucrul cu acest tip de format este foarte important pentru că se utilizează pentru apelări ale procedurilor, iar formatul neregulat este folosit aproape exclusiv pentru comunicarea interfeţelor hardware-software sau în interiorul unui modul-program;
- *modelul automatului finit*, în care "enunţurile elementare" de control determină calea într-o reţea a stărilor ; tipul următorului enunţ este întotdeauna cunoscut din starea curentă şi valorile "enunţurilor elementare" anterioare ; acesta este un tip de limbaj folosit de obicei în sistemele care utilizează intrarea de tip text sau voce;
- *modul de lucru descriptiv*, are o structură internă atât de bogată, încât nu poate fi reprezentată de un singur model cu stare finită ; această clasă include programarea algoritmică.

Multe dintre modelele care s-au investigat au analiza lexicală localizată în interiorul pachetelor de intrare sau în subrutinele de limbaj-maşină. Intrările grafice iniţiază un flux de comunicare care ar putea fi, în principiu, analizat prin tehnici similare cu acelea utilizate pentru fluxurile de text sau voce. Se remarcă diferenţe ale implementării la nivelul lexical şi mai puţin la nivelul sintactic, unde cele trei funcţii (de control, identificare şi aprovizionare a valorilor) sunt echivalente *[6]. [12].*

1.4. Tehnici de analiză şi sinteză grafică asistată

Modelele pentru producţie imagine şi sinteza-grafică depind foarte mult de domeniul de aplicaţie. Există câteva tehnici cu o largă aplicabilitate (spre exemplu, algebra booleană sau ecuaţiile diferenţiale şi geometria coordonatelor), dar acestea sunt folosite în domenii de calcul ştiinţific şi de aceea, nu trebuie să fie privite ca tehnici specifice sistemelor grafice. O tendinţă actuală este aceea de a dezvolta tehnici de analiză şi sinteză pentru fiecare domeniu de aplicaţie. Acestea combină cunoştinţe din matematică şi fizică, documentaţie de cercetare, standarde şi metodologii

internaţionale, reguli de proiectare formulate de forurile naţionale etc.

O mare parte a aplicaţiilor de ***RV*** se bazează pe utilizarea graficii de sinteză de nivel specializat unde fiecare problemă cere propria sa soluţie şi nu predomină generalizările. Dar există şi concepte generale care îşi păstrează caracteristicile lor fundamentale când sunt implementate în moduri diferite. Acestea sunt:

- *Simularea* - aplicată analizei unui sistem care variază în timp, prin metode în care variabile specifice sistemului iau valori ce corespund unor valori ale variabilelor de stare ale sistemului supus analizei.
- *Analiza elementului finit* dezvoltată iniţial ca metodă pentru analiza structurilor statice, dar extinsă în multe alte aplicaţii. Ideea fundamentală a *elementului finit* are o mare aplicabilitate în cazul tuturor situaţiilor unde prezumţia de linearitate este acceptată.
- *Optimizarea* - metodă de automatizare a mai multor variante de implementare posibile, dirijate de un program care să controleze ajustări ale parametrilor pentru a găsi o combinaţie optimă.

Există trei abordări importante ce utilizează simplificări adecvate aplicaţiilor de ***RV*** bazate pe sinteză grafică:

1. *Abordarea exhaustivă* este indicată când parametrii pot lua numai un şir limitat de valori discrete;
2. *Abordarea liniară* găseşte optimul pentru un criteriu linear atunci când limitările asupra valorilor parametrilor sunt toate liniare;
3. *Abordarea neliniară* converge spre un optim local al unui criteriu neliniar de la un punct de pornire apropiat. Nu este neapărat necesar să se găsească un optim global.

Ca metodă specifică mai trebuie menţionată şi *sinteza directă*. Deşi tehnicile variaţionale sunt valabile încă de la Euler, sunt puţine aplicaţiile care folosesc metode pentru comutarea directă a criteriului de optimizare în forma şi parametrii unei proiectări propriu-zise. În practică se îmbină elemente ale acestor metode pe segmente de proiect *[2]. [12].*

1.5. Reprezentări şi structuri de informaţii

Ar fi de aşteptat ca în ceea ce priveşte reprezentările, situaţia să fie la fel de flexibilă şi nestructurată precum cea care implică metodele de analiză. Reprezentările sunt: funcţionale, tipologice şi geometrice. O altă clasificare a reprezentărilor este dată de numărul de detalii pe care le conţin. Relaţiile tipologice sunt mai ordonate, incluzând tehnici şi scheme de înmagazinare a indicatorilor.

Reprezentările geometrice referă trei domenii de tratare teoretică. Primele două sunt bine dezvoltate, iar al treilea constituie subiectul cercetărilor actuale *[5], [18]*:

- *Geometria solidului*: este o problemă de manipulare a corpurilor solide într-un spaţiu ortogonal rigid (fie bidimensional sau tridimensional). Ea apare, de exemplu, la aşezarea componentelor într-o scenă.
- *Geometria suprafeţelor*: încearcă să rezolve problema de a reprezenta numeric caracteristici de suprafaţă (cum ar fi, de exemplu, netezimea, relieful, materialul etc.). Simplificările utilizate au în vedere faptul că în mod normal suprafeţele sunt individuale. Orice interacţiune dintre ele este cerută explicit de către utilizator. Principala dificultate operaţională în lucrul cu suprafeţele neregulate constă în atribuirea unui sistem controlabil de interacţiune cu ele. Modelul unei suprafeţe trebuie să fie maleabil, astfel

încât sistemul să poată să îndeplinească repede schimbările cerute pentru o formă şi implicit să dea acelei forme caracteristicile calitative cerute prin modificări (atingere, iluminare, interacţiune).

- *Geometria fragmentelor simple*: principala problemă este aceea a reprezentării fragmentelor mărginite teoretic şi practic de suprafeţe geometrice care au configuraţii reale complexe.

În ceea ce priveşte structurile de informaţii, acestea pot fi abordate pe două nivele. Unul este nivelul logic, aşa cum este văzut de proiectantul aplicaţiei, iar altul este nivelul intern, reprezentând detaliile de implementare a structurilor. În cadrul nivelului logic trebuie prevăzute funcţiile care să opereze la nivelul intern; deoarece aceste funcţii operează cu seturi de structuri, manipularea lor trebuie să fie în relaţie strânsă cu tabelele legăturilor către sistem.

Pentru programatorul unei aplicaţii, este de mare importanţă măsura în care limbajul de programare îl ajută sau îl restricţionează în proiectarea structurilor de informaţii. Există limbaje care oferă programatorului facilităţi de manipulare a structurilor la nivelul logic prin construirea de seturi, alocarea de atribute şi setarea asociaţiilor între componente. Alte limbaje nu prevăd structuri logice şi oferă un grad redus de asistenţă în implementarea manipulării datelor. Unele utilizează liste simple ca structură internă de bază şi permit utilizarea de proceduri de acces a funcţiilor la nivel logic. Punctul central al atenţiei programatorilor de aplicaţii de ***RV*** este descrierea structurilor interne.

Au fost propuse metode de lucru cu structuri largi, cele mai multe orientându-se spre "paginare". Acestea includ "paginarea" structurilor de liste, paginarea structurilor asociative şi tehnici de memorie virtuală. Dintre acestea cele cu memorie virtuală par mai interesante, dar utilizarea acestora depinde de echipamente hardware speciale, mulţi proiectanţi neputând să le folosească. Aceştia trebuie să implementeze structurile largi cu ajutorul accesului aleator sau secvenţial la colecţii de date heterogene.

1.6. Organizarea prelucrărilor în sisteme grafice

Există mai multe modalităţi de acces la modulele software în aplicaţiile de ***RV***, incluzând sisteme batch şi sisteme cu acces multiplu, mari sau mici, precum şi posturi dedicate unui singur utilizator. Sistemele cu acces multiplu deservesc câteva tipuri particulare de produse grafice pentru ***RV***:

- sisteme cu distribuirea timpului (timeshared): acestea sunt sisteme cu acces multiplu, accesate de doi până la câteva sute de utilizatori;
- sisteme pentru un singur utilizator, folosind echipamente speciale, cum ar fi mănuşile de date, casca de realitate virtuală etc.;
- sisteme distribuite, eventual cu mai multe procesoare, fiecare executând o altă sarcină;
- sisteme grafice "satelit", un caz special al calculului distribuit.

Multe din arhitecturile referitoare la organizarea sistemelor grafice se bazează pe "sateliţi grafici", care încearcă să obţină un răspuns rapid al terminalelor grafice cu răspuns lent, prin ataşarea la servere puternice. Ideea este ca "satelitul" să răspundă rapid cerinţelor simple şi să menţină într-o memorie tampon datele pe care le va transmite apoi sistemului central, reducând numărul de accesări ale acestuia.

Principalele teme ale proiectării aplicaţiilor de ***RV*** bazate pe inteligenţă artificială sunt de a explora reprezentarea formală a cunoştinţelor şi de a dezvolta

tehnici şi modele noi. Sistemele bazate pe inteligenţă artificială care se concentrează pe dezvoltarea unor reprezentări sunt cunoscute ca sisteme expert, sau mai general sisteme bazate pe cunoştinţe *[1]*. În literatura de specialitate, se arată că sistemele bazate pe inteligenţă artificială conţin descrieri ale lumii reale, realizate astfel încât o maşină inteligentă să poată aduce concluzii noi despre mediul său prin simpla manipulare a acestor descrieri, ori aceasta le recomandă şi pentru aplicaţii de ***RV***.

Au fost investigate mai multe tehnici de reprezentare a cunoştinţelor, trei dintre ele fiind reprezentative pentru metodele folosite în sistemele comerciale de software bazat pe cunoştinţe. Acestea sunt: sisteme de producţie grafică, sisteme ferestre şi sisteme de grafuri şi reţele.

În sistemele de „producţie grafică", cunoştinţele sunt reprezentate ca o colecţie de perechi de acţiuni, numite reguli de producţie, care sunt implementate în bazele de cunoştinţe pentru a fi completate în continuu. Acolo unde sistemele de producţie sunt utilizate pentru a stoca reguli euristice şi fapte, „sistemele fereastră" reprezintă cunoştinţele pentru utilizarea unor obiecte-prototip. „Ferestrele" constituie clase, cum ar fi colecţiile de obiecte şi date încadrate în ferestre, în multe cazuri analog cu modul de manipulare a structurilor de date în limbajele de programare. *[10]*.

În multe din problemele de ***RV*** pe care inteligenţa artificială şi le poate asuma, greutăţile pot fi în mare parte reduse prin identificarea tipului generic de produs grafic, descompunerea în detaliu a mediului şi a aranjării componentelor (rafinarea planurilor) *[1]*. Materialele particulare, dimensiunile şi tratamentul suprafeţei pot fi alese pe baza conceptului general că o proiectare particulară se află la un moment dat într-un spaţiu multidimensional şi că graniţele acelui spaţiu care definesc proiectele fezabile sunt impuse de restricţii. Ideea de bază a restricţiilor este că abordările pot fi modelate într-o reţea a atributelor proiectării.

Clasificarea instrumentelor de proiectare bazate pe cunoştinţe este:

- Instrumente automate de proiectare (în care sunt analizate metodele euristice şi algoritmice);
- Instrumente analitice (conţin experienţa în aplicarea analizei);
- Instrumente expert (concretizează experienţa în domeniul proiectării).

Între abilităţile unui expert în proiectare de aplicaţii ***RV***, se situează şi capacitatea de a opera cu conceptele geometrice. O parte a dificultăţii interpretării geometrice este faptul că metodele utilizate pentru modelarea tridimensională nu sunt semantic foarte bogate. Aceste metode ar trebui să includă: generarea instrumentelor pentru modele numerice, modelarea componentelor care vor fi asamblate, modelarea ansamblului şi generarea modelelor elementului finit.

2. METODE DE OPTIMIZARE A PROIECTĂRII

2.1. Optimizări ale funcţiilor de proiectare

Optimizarea este ramura matematicii aplicate care se ocupă cu tehnica de a obţine cea mai favorabilă soluţie pentru o problemă dată. Poiectanţii de aplicaţii grafice sunt obligaţi să lucreze cu numeroase constrângeri, pentru a creşte eficienţa şi a scădea costurile exploatării produselor grafice.

Dificultatea în proiectare este datorată necesităţii de a stabili geometrii de obiecte grafice şi traiectorii complicate. Cercetările iniţiate în optimizarea acestora au vizat domeniul cinematicii. S-au dezvoltat proceduri pentru echipamente periferice pentru care redarea obiectelor grafice este independentă de volumul încărcării şi de

viteza cerută. Mulți cercetători au folosit tehnici de căutare aleatoare, în special pentru a rezolva probleme de cinematică. Căutarea aleatorie euristică combinatorie a fost folosită pentru analiza interacțiunilor neliniare, dar nu există documentații oficiale care să explice în detaliu metodele folosite *[2], [14].*

Ecuațiile constrângerilor folosite în proiectarea modulelor complexe sunt neliniare. O încercare de a implementa o tehnică de programare liniară sau o metodă bazată pe gradient ar putea fi încununată de succes numai prin liniarizarea problemei. Pentru a ajunge la acest rezultat, o metodă ar fi implementarea de căutări aleatorii, care păstrează funcția originală şi ecuațiile constrângerilor intacte. În literatura de specialitate, posibilitatea căutării aleatorii pentru o soluție "acceptabilă" şi în acelaşi timp, a menținerii nelinearității problemei este prezentată doar teoretic fără a se descrie implementări în practică. Strategia folosită în algoritmul căutării aleatorii este de a genera cât mai multe soluții posibile bune şi de a selecta cea mai bună soluție dintre ele. Metoda folosită ar fi bazată pe căutarea aleatorie pas cu pas.

Ca majoritatea tehnicilor de optimizare, căutarea aleatorie este un algoritm iterativ, dar, spre deosebire de metodele bazate pe gradient, nu este o metodă de determinare numerică, ci o căutare ordonată în direcție aleatorie. Tehnica de căutare aleatorie *[5]* ar putea uneori să fie prea înceată în atingerea unui optim. Aceasta poate fi de asemenea, înceată în calculele pentru un model neliniar. Pe de altă parte, este capabilă să rezolve problema optimizării neliniare. De aceea, metoda căutării aleatorii se recomandă a fi folosită pentru inițierea soluției posibile de început. În descrierea algoritmului de căutare aleatorie sunt folosiți următorii termeni.

- *Epocă*: o iterație completă sau căutare făcută pentru un număr special aleator.
- *Iterație*: de fiecare dată, mărimea unui pas este modificată.
- *Mărimea ,pasului*: mărimea cu care valoarea unei variabile se schimbă la fiecare iterație.
- *Direcția căutării*: un număr aleator generat între – 1 şi + 1 (panta unei linii drepte).
- *Depăşirea constrângerii*: mărimea cu care valoarea constrângerii a depăşit limitele permise din ecuațiile de constrângere.

Deoarece sistemele de modelare 3D sunt foarte complexe, există un număr substanțial de soluții posibile şi minime locale. Pentru determinarea unei soluții optime, este recomandată găsirea a cât mai multe soluții concrete posibil şi determinarea optimului global prin comparație directă. Cu cât numărul soluțiilor din spațiul minim local creşte, creşte şi posibilitatea de a găsi un minim global. Algoritmul căutării aleatorii care a fost investigat foloseşte o mărime de pas variabilă pentru examinarea valorilor constrânse. Trebuiesc specificate mărimea pasului şi criteriul de convergență. Această metodă şi-a dovedit eficiența pentru problemele de mare complexitate şi anvergură, unde ar fi greoi de făcut o optimizare convențională, folosind apeluri multiple pentru a determina gradientul Hessian al matricilor.

Premisa de bază pentru abordarea propusă este că, dând o soluție de start, căutarea este condusă în mai multe direcții aleatorii în regiunea posibil de investigat.

Dacă soluția de start este în afara regiunii posibile, parametrii proiectării sunt modificați prin schimbarea mărimii pasului şi căutarea continuă, până ce se obține o soluție acceptabilă. În funcție de direcția aleatorie, poate fi obținut un minim pentru acea direcție. Toate soluțiile acceptabile de acest gen formează setul de soluții posibile. Prin efectuarea unui număr mare de căutări, poate fi determinat un minim global.

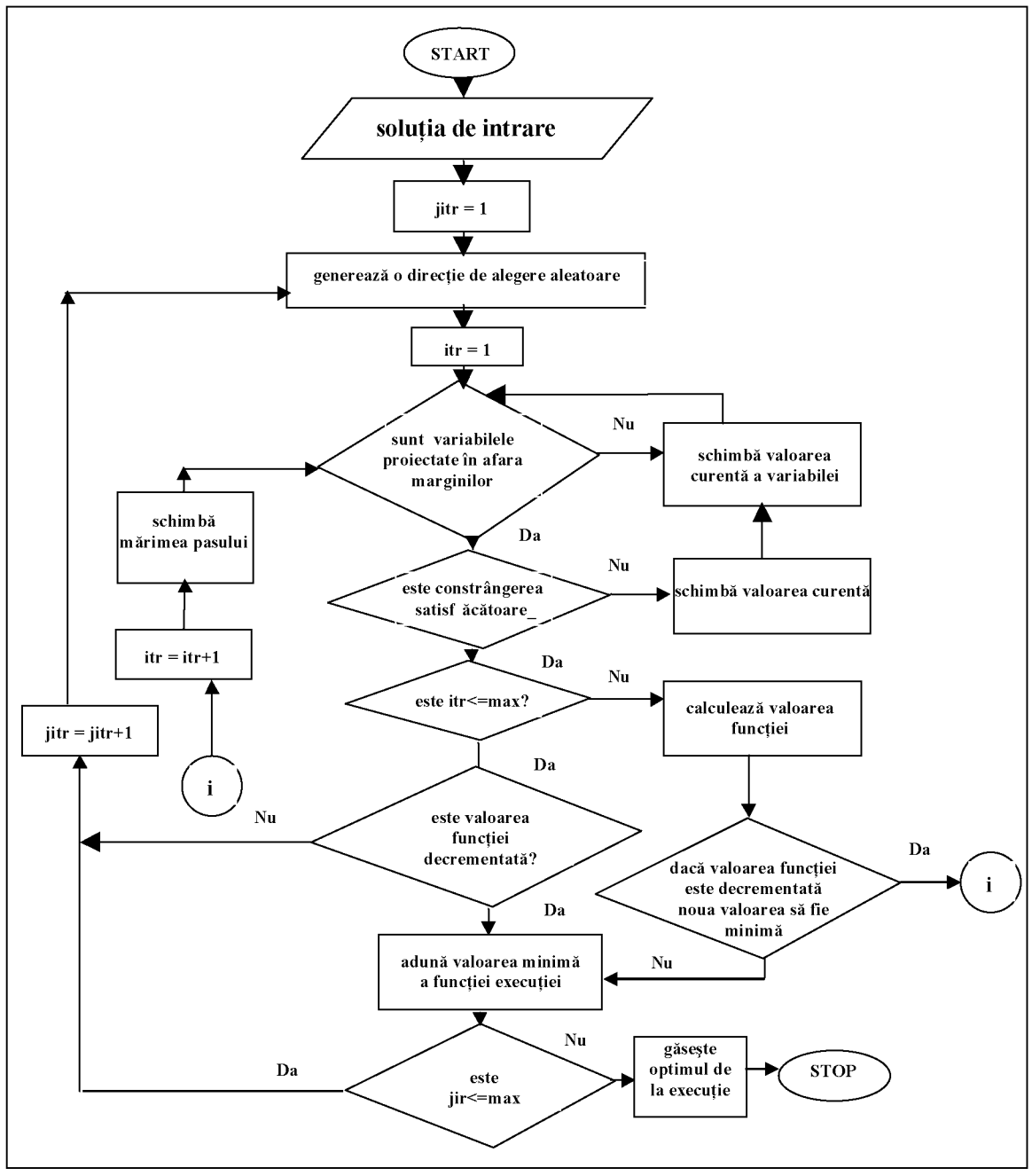

Figura 2.1. Tehnica de căutare aleatorie

Schema logică a tehnicii de căutare aleatorie investigate este prezentată în *Figura 2.1*. Factorul critic în folosirea algoritmului căutării aleatorii este definirea criteriului de terminare. Aceste criterii sunt de obicei numărul de direcții aleatorii generate şi numărul căutărilor conduse în fiecare direcție. Dacă nici un optim acceptabil nu este găsit, numărul de iterații care urmează să fie efectuat trebuie să fie crescut şi procedura repetată. Însă dacă numai foarte puține optime sunt găsite, este din nou necesară creşterea numărului de iterații pentru a se asigura că un minim global este atins. Un insuficient număr de iterații s-ar putea să nu genereze suficiente direcții aleatorii pentru a atinge o soluție optimă.

Este recomandabilă adoptarea unei secvențe pentru executarea unei căutări aleatorii. Metoda folosită în această testare a utilizat o procedură de mărime de pas

adaptivă. Dacă, în fiecare iterație constrângerile nu ar fi depăşite, mărimea pasului ar fi crescută. Aceasta ar accelera atingerea unui optim. Oricum, mărimii pasului nu îi este permis să depăşească maximul permis de mărimea de pas stabilită de către proiectant.

Metoda investigată se referă la problema satisfacerii constrângerilor şi la problema de a rămâne în limitele variabilelor proiectate la fiecare iterație. Într-o direcție aleatorie dată, cu o măsură a pasului particulară, constrângerile şi funcția trebuie să fie evaluate de mai multe ori pentru a observa orice schimbare. Orice schimbare de funcție va cauza o schimbare potrivită în direcția căutării, astfel încât să se deplaseze înspre optim.

În algoritmul curent, pentru fiecare direcție aleasă, o căutare completă este condusă, fără ca direcția să fie schimbată. Aceasta ajută la găsirea a cât mai multor soluții posibile.

2.2. Prelucrarea modelului

În producția grafică asistată de calculator construirea desenului consumă acelaşi timp ca şi prin metodele clasice (poate chiar mai mult), dar când sunt necesare schimbări în proiect, asistarea calculatorului este mai avantajoasă (atât din punct de vedere al timpului necesar, cât şi al acurateței operației). Facilitățile oferite de manipularea modelului prin tehnici computerizate pot fi grupate astfel:

- Cele care se aplică transformărilor (translației, rotației şi scalării) elementelor unui model. Aceasta poate implica mutarea sau copierea pentru crearea unui sau mai multor duplicate în structura de date;
- Cele care permit utilizatorului modificarea elementelor geometrice, la aranjarea sau extinderea lor în cazul intersectării cu alte elemente;
- Funcții de ştergere permanentă sau temporară a unei entități din model (utilizate de obicei pentru simplificarea desenului, pentru îmbunătățirea performanțelor sau pentru a face selectarea sau vizualizarea mai uşoară);
- Diferite alte funcții care permit, de exemplu, gruparea unor entități.

2.2.1. Transformările obiect

Poziția unui obiect definită într-un sistem de coordonate poate fi exprimată într-un alt sistem, prin transformarea coordonatelor. În acest tip de transformare, obiectul poate fi considerat staționar, iar sistemul de coordonate mobil. Când entitățile unui model sunt manipulate pentru mutare sau pentru copiere într-o nouă poziție, are loc un proces similar. În acest caz, sistemul de coordonate este staționar, iar obiectul este mobil. Reprezentările care constituie suportul matematic necesar manipulării, numite *transformările obiect*, sunt similare transformărilor de coordonate.

Componentele principale ale transformării-obiect sunt:

- *Translarea*: care poate fi descrisă ca o scădere de vectori;
- *Rotația*: care poate fi descrisă ca o înmulțire de matrici;
- *Scalarea*: care poate fi descrisă ca o translare de matrici;

2.2.2. Rearanjarea (trim) şi funcții extinse

Al doilea grup de funcții pentru manipularea entităților implică rearanjarea sau extinderea (numită uneori relimitare) entităților aflate la intersecția cu alte elemente geometrice. Rearanjarea implică eliminarea unor părți ale entităților, mărginite de una sau mai multe intersecții. Rearanjarea sau extinderea pot fi aplicate tuturor figurilor geometrice *[5]*.

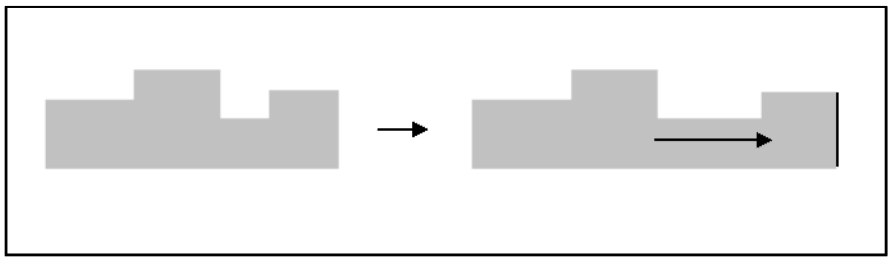

Figura 2.2. Rearanjarea obiectelor

Observații referitoare la aceste funcții sunt:

- în anumite cazuri, operația de rearanjare poate modifica stilul liniilor sau al fonturilor pentru a indica de exemplu, o linie ascunsă;
- cursorul este utilizat pentru a indica acea parte a entității supusă modificării şi pentru a rezolva orice ambiguitate referitoare la intersecția utilizată;
- aspectul cel mai important al operațiilor de rearanjare sau extindere este calcularea intersecțiilor dintre entități; acest calcul este deosebit de complex, rearanjarea suprafețelor este o opțiune inclusă numai în sistemele sofisticate.

Programele de stocare sunt seturi de algoritmi şi funcții care acționează asupra unor structuri de date. Se cunosc mai multe moduri de a stoca un model. S-au studiat următoarele aspecte particulare:

- structurarea datelor pentru modelarea interactivă utilizând cadre 2D şi 3D şi geometria suprafețelor (unde relațiile dintre entitățile geometrice sunt mai puțin importante decât în geometria solidelor);
- stocarea imaginilor vectoriale în fişiere de vizualizare;
- asocierea entităților geometrice cu cele utilizate în construcția lor (asociativitatea dintre entități);
- asocierea datelor non-geometrice cu modelul geometric (utilizare atribute).

2.2.3. Structuri de date în modelarea interactivă

Elaborarea detaliată a specificațiilor structurilor de date este suport pentru modelarea interactivă. Cerințele generale ale modelării obiectelor complexe utilizând tehnicile interactive sunt:

- să permită manipularea interactivă a datelor;
- să suporte orice tip de date (geometrice, text, voce, alfanumerice, etichete)
- să permită asocieri între date diferite, acolo unde este necesar;
- să permită anumitor proprietăți definite (de exemplu, număr şi stil de linii sau culori) să fie asociate cu elementele geometrice ale obiectului grafic;
- să ofere posibilitatea recuperării datelor "evacuate" în urma ştergerii sau modificării modelului;
- să ofere facilitatea stocării unor forme geometrice, pentru repetate referințe la forma geometrică respectivă, reutilizare;
- să permită compactarea (minimizarea capacității de stocare);
- să permită lucrul cu modele de diferite anverguri;
- să accepte definirea unor combinații de entități;

O structură de date care acoperă cerințele unei cantități arbitrare de date pentru fiecare entitate şi pentru combinații arbitrare de entități, cuprinde o listă sau un tabel de entități cu referințe încrucişate (sau pointeri) de la această listă la tablouri separate

de întregi, numere reale sau alte date specifice entității (ca în *Figura 2.3.*).

Tabelele de această structură sunt de tipul: *tabelul entității* (TE), *tabele de date reale* (TR) şi *tabele de date întregi* (TÎ). În (TE) o serie de intrări, fiecare conținând un număr fix de elemente ale ariei utilizate pentru tabel, sunt asignate câte una pentru fiecare entitate. Acestea conțin date generale, aplicabile oricărui tip de entitate ca: tipul entității, stilul liniei şi culoarea, împreună cu pointeri spre tipurile de date specifice entității. De exemplu, o linie va avea numere reale pentru punctul de start (x,y,z) şi pentru punctul de stop. O curbă Spline va avea numărul de noduri într-o (TÎ), în timp ce parametrii în fiecare nod sunt stocați în (TR). Anumite entități, ca arcele de cerc şi secțiunile conice, sunt plane şi de aceea este necesară stocarea unor informații referitoare la orientarea planelor de construcție împreună cu datele ce definesc mărimea şi localizarea entității.

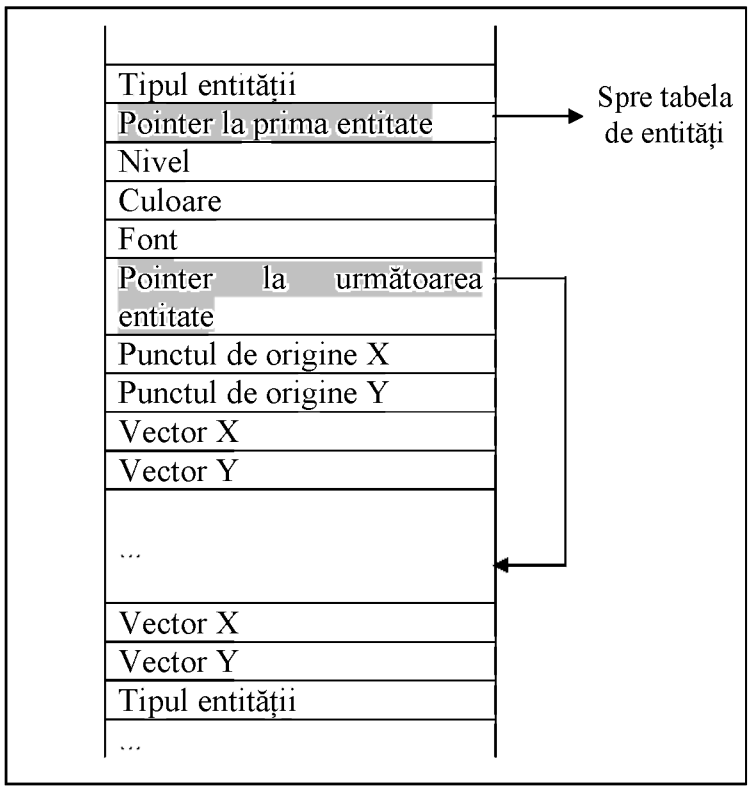

Figura 2.3. Structuri de date şi tabele

2.3. Geometrie asociativă şi atribute

Structura de date definită anterior stabileşte o colecție aleatoare de linii şi arce de cerc, cu nici o dată referitoare la relațiile dintre acestea, în afară de gruparea lor. În modelarea solidelor sunt definite geometriile şi topologiile unor forme. Relațiile între ele şi modelarea suprafețelor sunt mai puțin cuprinzătoare, dar este des utilizată asocierea entității cu relațiile utilizate în propria definiție.

O cale de a realiza acest lucru este de a extinde descrierea tipului entității pentru a include un sub-tip sau formă. De exemplu, o linie poate fi definită:

- *cazul 1*: între două coordonate introduse;
- *cazul 2*: între două puncte;
- *cazul 3*: tangentă la două curbe.

Se memorează pointeri la entitățile utilizate în construcția şi localizarea

entității. Cu un astfel de aranjament este posibil ca modificarea unei entități să se reflecte şi în entitățile dependente, în mod automat sau la cererea utilizatorului. Această facilitate, cunoscută drept conceptul de geometrie asociativă, este utilizată în particular la redesenarea dimensiunilor pentru a reflecta schimbările din desen. Este dificil să se construiască un model fără a se profita de avantajele geometriei asociative.

În plus, la asocierea entităților între ele, se pot asocia date non–geometrice, prin utilizarea atributelor. Acestea sunt de obicei perechi nume-valoare, unde "numele" este un şir alfa-numeric şi "valoarea" poate fi un şir sau un număr. Ele sunt legate din nou de entități prin pointeri. Fiecare entitate poate fi asociată cu un număr de atribute şi fiecare atribut cu un număr de entități ("many-many arrangement").

2.3.1. Proiectarea "orientată obiect" pentru sisteme grafice

Multe sisteme grafice comerciale sunt produse software foarte mari, greu de întreținut şi depanat. În afară de întreținerea programului, este foarte greu să se reutilizeze sistemul software care a fost scris pentru un anumit scop şi de aceea un efort deosebit este risipit pentru rescrierea unor părți de program pentru funcții care au fost deja realizate. Unul dintre motivele pentru care apare această constrângere este faptul că unele proceduri tind să fie dependente de structurile de date folosite şi invers. (de exemplu, un sistem grafic foloseşte o anumită structură de date; dacă se doreşte introducerea unei noi entități geometrice care nu se potriveşte cu această structură de date, dezvoltatorii sistemului vor trebui să reprogrameze sistemul existent, pentru a include noua entitate). Această dificultate a fost depăşită prin proiectarea orientată obiect [7]. Accentul în proiectarea orientată obiect ***OOD*** este pus mai puțin pe structura de date şi structura procedurilor, cât mai ales pe descrierea obiectelor, rolul pe care acestea îl au şi pe natura comunicației dintre ele. Trimiterea mesajelor este unul din cele trei elemente pe care este construit un sistem bazat pe *OOD*. Celelalte două sunt conceptele de „clasă" şi „moştenire", care permit modulelor să fie definite similar cu obiecte cu care împart caracteristici comune.

Un anumit obiect poate fi o instanță a unei clase de obiecte şi, din această clasă, poate moşteni atribute care pot fi date sau proceduri. Un obiect poate moşteni atribute din mai multe clase, organizate într-o ierarhie de clase. Din acest motiv, un proiectant lucrează în termeni generici, iar detaliile de implementare pot fi definite diferit pentru diferite clase de obiecte.

Cele mai multe sisteme de modelare 3D exploatează *OOD*. De exemplu, entitățile geometrice din anumite sisteme grafice sunt reprezentate ca obiecte care sunt instanțe ale claselor de suprafețe Spline, linii sau arce de cerc. Mesajele tipice pentru aceste sisteme grafice pot instrui obiectele să se deseneze, să se mute sau să-şi schimbe culoarea. În aceste sisteme, pot fi adăugate fără dificultate funcții noi şi acestea pot moşteni anumite proceduri din clasa de entități. Mai mult, transmiterea mesajelor dintre obiectele grafice exploatează posibilitatea folosirii asocierii între entități.

2.3.2. Generalități privind bazele de date grafice

În sistemele grafice este nevoie să se memoreze modelele individuale în "fişiere desen", de unde acestea pot fi apoi restaurate *[19]*. Mai mult, în aceste fişiere trebuie să se poată memora şi alte tipuri de date ca şi componente standard sau date simbolice, programe, date numerice, informații referitoare la elemente finite etc. Anumite sisteme memorează modelele sau secțiuni din modele, ca fişiere care reproduc structurile de date interactive. O clasă larg răspândită de tabele permite colecțiilor de entități să fie definite prin selectare dintr-un fişier de secțiuni şi ulterior să fie inserate (de obicei la orice scară, orientare şi locație) în orice tabel de secțiuni.

Anumite sisteme sunt structurate pe forma simbol-instanță (de exemplu: uşile, ferestrele şi alte componente standard ale unei încăperi sunt definite o singură dată şi orice scenă utilizează numai o referință la ultima definire a componentei). O situație similară este întâlnită în ingineria generală asistată de calculator unde se evită duplicarea unor date în baza de date. O scenă de ansamblu este o colecție de referiri la modele aflate în altă parte în baza de date, împreună cu transformările necesare pentru orientare şi localizare a modelului în ansamblul mediului virtual. O scenă poate fi compusă din vederile necesare modelului, împreună cu dimensiunile şi alte adnotări. Listele de componente sunt mai curând generate direct din interogarea ansamblului tridimensional, decât din atributele ataşate fiecărei entități.

3. DEFINIREA ŞI REPREZENTAREA DATELOR

3.1. Conceptele de "baze de date" şi "sistem de gestiune a bazelor de date"

În ultimul timp, datorită utilizării telematicii, a sistemelor distribuite geografic, a accesului partajat la resursele sistemelor de calcul, conceptul de "baze de date" a devenit familiar şi marelui public. Percepția foarte generalizată a acestui concept este de "colecție de date administrate de un calculator şi accesibile mai multor utilizatori in acelaşi timp". Noțiunea de "bază de date" (***BD***) s-a dezvoltat începând cu deceniul 70, perioadă în care s-au dezvoltat aplicații care accesau cantități mari de date, ceea ce a determinat dezvoltarea suporților fizici şi a tehnologiilor de arhivare evoluate. Interacțiunea cu bazele de date este asigurată de programe speciale care compun sistemul de gestiune a bazelor de date (***SGBD***). Acesta permite utilizatorilor să definească datele, să le consulte sau să le actualizeze *[3]*.

Obiectivele generale ale unui sistem de gestiune a bazelor de date sunt: definirea legăturilor între date; coerența datelor; suplețea accesului la date; securitatea; partajarea datelor; independența datelor; administrarea şi controlul.

Într-un ansamblu de date conținând un volum important de cunoştințe, este extrem de importantă asignarea coerenței datelor stocate, ceea ce poate permite utilizatorului să-şi definească reguli raționale, satisfăcute funcție de proprietățile definite. Accesul la date este posibil prin intermediul limbajelor declarative şi de nivel înalt. Datele trebuiesc protejate față de agresiunile exterioare. Acestea pot fi agresiuni fizice, cum ar fi pana unui periferic pentru stocare sau o eroare software. Agresiunile exterioare asupra datelor pot fi şi de natură umană, cum ar fi manipularea intenționat defectuoasă din partea unui utilizator. Pentru a evita efectele asupra datelor produse de intervenții distructive, un ***SGBD*** trebuie să permită „lucrul cu puncte de reluare" care să relanseze sesiunea de lucru şi să revină la o stare satisfăcătoare, astfel încât să se evidențieze în mod coerent actualizările istorice asupra datelor, înainte de a accepta sau anula orice intervenție asupra datelor.

Una din cerințele cele mai importante pentru o bază de date, a fost cea de accesare partajată a datelor. Diferite aplicații care operează asupra aceloraşi date, trebuie să poată executa orice sesiune de lucru ca şi cum ar fi unicul operator asupra datelor respective. Un ***SGBD*** are sarcina de a oferi mijloacele pentru a controla partajarea datelor şi eventualele conflicte de acces care ar putea exista între mai mulți utilizatori sau mai multe aplicații şi de a oferi instrumentele pentru a le rezolva. Un aspect important oferit de ***SGBD*** este independența datelor. O aplicație care manipulează datele prin intermediul fişierelor este puternic dependentă de datele sale,

deoarece aplicația trebuie să cunoască structurarea acestora şi metodele de acces la acestea. Sistemul poate evolua având în vedere eventuale noi cerințe, fără a se impune rescrierea aplicațiilor deja implementate.

Independența datelor este foarte importantă pentru evoluția şi mentenanța unei aplicații. Independența fizică trebuie să permită modificarea structurii de stocare sau a metodelor de acces la date, fără ca acestea să afecteze aplicația. Se poate adăuga sau suprima un index al unei colecții şi se poate schimba reprezentarea internă a datelor numerice. Independența logică trebuie să permită modificarea organizării datelor fără a afecta utilizatorii. Acest nivel de independență permite îmbogățirea unei baze de date existente având în vedere noile structuri, fără a le afecta pe cele deja existente. Independența logică permite satisfacerea unor noi cerințe, ceea ce este o condiție indispensabilă pentru sistemele grafice, dacă se ține cont de faptul că o bază de date cu obiecte grafice este adesea un model al lumii reale şi de faptul că lumea reală, la fel ca şi cerințele utilizatorilor, se schimbă în timp. Principiul asigurării independenței datelor este dificil de realizat şi din cauză că sistemele care le accesează au nivele diferite de independență.

Un ***SGBD*** trebuie să administreze volume mari de date şi să ofere timpi de acces rezonabili pentru utilizatori. Această cerință de performanță impune ca o mare parte a preocupărilor legate de ***SGBD*** să fie consacrate ameliorării tehnicilor de acces şi de optimizare. O bază de date care conține obiecte grafice trebuie să poată fi accesată simultan de mai mulți utilizatori, care pot avea cerințe uneori incompatibile, motiv pentru care se impune ca administratorul bazei de date să fie implicat, încă din etapa de proiectare, în următoarele activități:

- definirea structurii datelor conținute în baza de date cu obiecte grafice şi a evoluției lor eventuale pentru a satisface noi cerințe;
- definirea structurii fizice a datelor şi a strategiilor de acces;
- asigurarea independenței fizice a datelor grafice;
- definirea autorizărilor acordate utilizatorilor;
- definirea punctelor de reluare şi de salvare sistematică;
- optimizarea organizării fizice pentru a îmbunătăți performanțele globale ale sistemului sau pentru a accepta noi specificații.

Pentru a defini independența datelor se consideră trei nivele de reprezentare şi anume:

a. Nivelul conceptual.

Acest nivel este partea cea mai importantă a unui ***SGBD*** deoarece definirea schemei conceptuale corespunde unei activități de modelare care transcrie în termeni abstracți entitățile lumii reale sau care reproduc lumea reală. Pentru a realiza această modelare, ***SGBD***-ul oferă un model de date căruia i se asociază un limbaj de definire a datelor care permite specificarea schemei conceptuale în interiorul modelului. Realizarea şi utilizarea unui model de date grafice are repercursiuni foarte importante asupra naturii aplicațiilor care pot suporta un astfel de ***SGBD***, precum şi asupra modalităților concrete de interogare. Din punct de vedere al modelării datelor grafice, şi pentru acestea se utilizează modelele cunoscute, şi anume: modelul ierarhic, modelul rețea şi modelul relațional.

Modelul rețea este o extensie a modelului ierarhic, în care graful obiectelor grafice nu este limitativ. Permite reprezentarea partajării interogării obiectelor grafice şi, de asemenea, legăturile ciclice între obiecte. O schemă conceptuală a acestui model este alcătuită din definițiile înregistrărilor entităților, din legăturile dintre aceste entități şi din ansamblul exprimat de legăturile multivalente între înregistrări.

Modelul relațional este fondat pe noțiunea matematică de relație şi permite reprezentarea datelor sub formă de tabele ale căror dimensiuni şi structuri sunt predefinite.

b. Nivelul fizic

Nivelul fizic defineşte schema fizică a bazei de date, adică reprezentarea pe suportul de stocare utilizat de sistem. Schema fizică este definită în termen de tabele şi înregistrări, rutine de gestiune a tabelelor şi rutine de exploatare a capacității de calcul, incluzând şi gestiunea perifericelor şi suporților.

c. Nivelul extern

Un ***SGBD*** este utilizat simultan de mai mulți utilizatori. Schema conceptuală reprezintă totalitatea informațiilor cunoscute de sistem, dar fiecare utilizator foloseşte individual doar o parte din acestea. Un ***SGBD*** oferă la nivel extern, conceptul de vizualizare care permite să se prezinte fiecărui utilizator porțiunea din schema conceptuală generală care corespunde nevoilor sale sau drepturilor sale de acces. Noțiunea este mai generală decât o simplă restricție a schemei conceptuale, oferă utilizatorului viziunea datelor sale ca o sinteză a informației reale reprezentate în baza de date. Nivelul extern este cel care permite definirea independenței logice a datelor.

3.2. Limbajul de definire a datelor

Limbajul de definire al datelor ***LDD*** este utilizat pentru a specifica schema conceptuală a bazelor de date; este legat de modelul de date utilizat de ***SGBD*** şi permite definirea şi modificarea ansamblului conceptelor pe care modelul are capacitatea de a le reprezenta. Nu conține informații asupra organizării fizice a datelor şi nici asupra suportului de stocare.

Exemplul nr. 1 conține reprezentarea „personajelor medievale" cu numele lor şi numărul de persoane din grupul respectiv, conform unui model rețea. Schema conceptuală descrie organizarea logică a datelor şi nu este afectată de modalitatea în care datele sunt administrate fizic. De această separare depinde în mod direct nivelul de independență fizică pe care o oferă sistemul.

Exemplul nr. 1

```
area name is personaje-medievale
record name is personaj
        location mode is system default
        within personaje-medievale;
        identifier is nume in personaj
        05 număr; type is fixed decimal 10;
        05 nume; type is character 30;
```

Exemplul nr. 2

```
create table personaje-medievale (
număr integer,
nom char (30))
```

Definiția prezentată în *Exemplul nr. 1* (reprezentare conform modelului rețea), a fost definită utilizând modelul relațional ca în *Exemplul nr.2*. Un ***SGBD*** recomandat pentru aplicații cu obiecte grafice trebuie să ofere un limbaj care să permită specificarea organizării fizice a datelor grafice. Un ***SGBD*** va trebui să ofere un limbaj care să permită căutarea, consultarea şi actualizarea datelor din bazele de date, în mod independent de modelul utilizat. Într-un sistem utilizând un model rețea, limbajul de manipulare a datelor este compus din comenzi elementare de tipul: *find, get, modify,* care permit "navigarea" în ansamblul general al obiectelor grafice din baza de date, pentru a căuta anumite date prin acces direct sau asociativ (*find*), prin căutarea lor (*get*) sau pentru actualizarea lor (*modify*).

Exemplul nr. 3

```
find first grup-personaje record
while not fail do begin
        get personaje-medievale; nume; număr
        if număr > 10 then print nume;
        find next personaje-medievale record
end
```

Secvența descrisă în *Exemplul nr. 3* prezintă un set de comenzi care permit afişarea listei numelor grupelor de „personaje medievale" din bazele de date, care au mai mult de 10 membri în grup. De observat că limbajele de manipulare a datelor ca cele prezentate în *Exemplele 1, 2 şi 3*, nu sunt decât liste de comenzi care asigură o "pătrundere" în limbajele de nivel înalt.

Limbajele de manipulare asociate modelelor relaționale sunt mai mult declarative decât fundamentate pe o schemă logică.

În limbajul SQL (Structured Query Language), utilizat pentru cea mai mare parte a sistemelor relaționale, căutarea descrisă în *Exemplul nr. 3*, se scrie mult mai simplu, conform secvenței prezentate în *Exemplul nr. 4*.

Exemplul nr. 4

```
select nume
from personaj-medieval
where număr > 10
```

3.3. Programele de aplicație pentru baze de date relaționale

Programele de aplicație pentru baze de date relaționale, cu obiecte grafice se alcătuiesc pornind de la nivelul extern, de la cunoaşterea cerințelor utilizatorului. Aplicația se poate descrie într-un limbaj clasic şi poate comunica cu ***SGBD***-ul prin transmiterea unor comenzi scrise în limbaj de manipulare a datelor. Transferul propriu-zis al datelor între aplicație şi ***SGBD*** se face într-o zonă tampon în care datele sunt citite şi / sau scrise. (*Figura nr. 3.1.*).

O astfel de aplicație nu înseamnă o simplă schemă conceptuală apelată de un limbaj de manipulare a datelor. Ea include părți importante de module program care accesează datele din baza de date cu ajutorul limbajului de manipulare a datelor. În

ceea ce priveşte nivelul de performanţă, modalitatea în care datele tranzitează între limbajul general de programare şi ***SGBD***, este determinantă pentru performanţele sistemului, mai ales în situaţia în care se doreşte accesarea partajată a datelor care compun obiecte grafice.

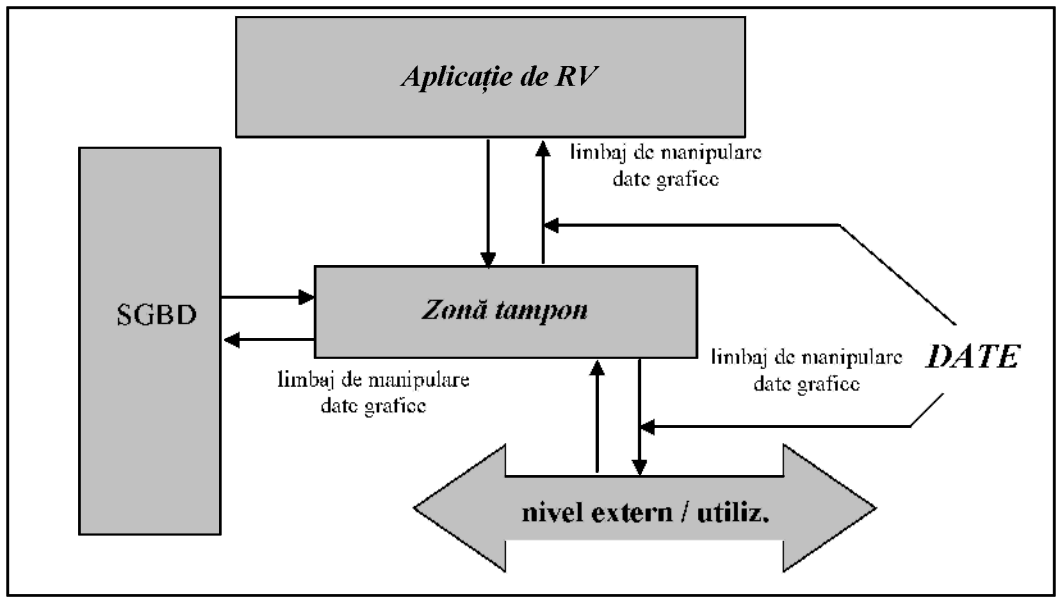

Figura nr. 3.1. Comunicaţia de date între SGBD şi o aplicaţie cu baze de date grafice

3.4. Definirea şi prezentarea comparativă a modelului relaţional

Lansat de Codd, în 1979, modelul relaţional a câştigat în popularitate, datorită bunei sale utilizări în numeroase sisteme comerciale, surclasând modelele precedente şi, mai ales, modelul reţea, mult timp utilizat. Un model de date are scopul de a reprezenta lumea reală/ virtuală în interiorul sistemului. În cea mai mare parte a aplicaţiilor grafice, modelul trebuie să reprezinte entităţi grafice (de exemplu, obiecte de mobilier, corpuri constructive ale ansamblurilor arhitectonice etc.)

Se prezintă în continuare câteva exemple de rutine, pentru gestionarea obiectelor de mobilier din principalele încăperi ale *Cetăţii medievale de la Suceava*. Schema conceptuală a trebuit să definească în mod particular noţiunile de „*piesă de mobilier*”, „*încăpere*” şi „*ansamblu arhitectural*”, precum şi legăturile care există între aceste entităţi. *Ansamblul medieval Cetatea de Scaun Suceava* include numeroase „*încăperi*”, iar o „*încăpere*” din „*cetate*” conţine mai multe „*piese de mobilier*”. În *Exemplul 5.a.* se prezintă schema de definire a datelor utilizând modelul reţea.

Cele trei clauze *record*, definesc entităţile reprezentând “*ansamblu cetate*”, “*încăperi*” şi “*mobilier*”. Clauzele *set* definesc legăturile multivalente care există între “*ansamblu_cetate*” şi “*încăperi*”, pe de o parte, şi între “*încăperi*” şi “*mobilier*”, pe de altă parte. Aceste clauze exprimă relaţia de apartenenţă a mai multor “*încăperi*” la “*ansamblu_cetate*” şi relaţia de apartenenţă a mai multor articole “*mobilier*” la o “*încăpere*”. Clauzele *insertion* şi *retention* descriu comportamentul pe care trebuie să îl aibă sistemul la crearea şi actualizarea entităţilor proprietare (*owner*) şi membre (*member*) ale legăturii. Această descriere combină într-o manieră destul de greoaie, descrierea schemei conceptuale, cu descrierea fizică a datelor. Clauzele din liniile 6, 14, 20, 32, 36, 43 şi 47 sunt descrieri ale căilor de acces şi de stocare. Faptul că trebuie

specificate în acelaşi timp, schema de organizare şi modalitatea de implementare, contravine principiului independenței fizice a datelor, care este extrem de important pentru performanța unui produs cu obiecte grafice complexe.

Exemplul 5.a.

```
1.  schema name is Cetatea_medievală_Suceava
2.  area name is încăperi
3.  area name is mobilier
4.  area name is ansamblu_cetate
5.  record name is încăpere
6.          location mode is system default
7.          within încăperi
8.          indentifier is număr încăperi
9.          02 număr încăperi; type is character 5
10.         02 nume; type is character 30
11.         02 suprafață; type is fixed decimal 10
12.         02 ansamblu_cetate; type is character 5
13. record name ansamblu_cetate
14.         location mode is system default
15.         within ansamblu_cetate
16.         indentifier is număr articol in ansamblu_cetate
17.         02 număr articol; type is character 5
18.         02 nume; type is character 30
19. record name mobilier
20.         location mode is system default
21.         within mobilier
22.         identifier is număr_mobilier
23.         02 număr_mobilier; type is character 5
24.         02 nume; type is character 30
25.         02 supafață; type is fixed decimal 4
26.         02 amplasare
27.         03 înălțime; type is fixed decimal 3
28.         03 culoare; type is character 30
29.         03 număr_încăpere; type is character 5
30. set name is ansamblu_cetate_încăperi
31.         owner is ansamblu_cetate
32.         order is permanent sorted by defined keys
33.         member is încăperi
34.             insertion is automatic
35.             retention is fixed
36.             key is ascending nume in ansamblu_cetate
37.             duplicates are not allowed
38.              nulls are not allowed
39. set selection is thru ansamblu_cetate_încăperi owner
40.             identified by identifier număr in ansamblu_cetate
41. set name is încăperi_mobilier
42.         owner is încăperi
43.         order is permanent sorted by defined keys
44.         member is mobilier
45.             insertion is automatic
46.             retantion is fixed
47.             key is ascending nume in încăpere
48.             duplicates are not allowed
49.             nulls are not allowed
50.         set selection is thru obiecte_încăperi owner
51.              identified by identifier număr_încăpere in
```

Pentru a implementa funcţiunile de manipulare a datelor, limbajul de manipulare trebuie să asigure comenzi în interiorul unui limbaj de programare de nivel înalt (de exemplu, C, C++, Cobol etc.). Pentru exemplul prezentat anterior, schema de definire a datelor este următoarea:

Exemplul 5b

```
find încăperi record
find first mobilier record in current încăperi_mobilier
while not fail do begin
    get mobilier; suprafața ocupată
    număr = număr + 1
    find next mobilier record in current încăperi_mobilier set
end
find first mobilier record in current încăperi_mobilier set
while not fail do begin
    get mobilier; descriere
    if număr > număr + 1 then
            print nume
    find next mobilier record in current încăperi_mobilier set
end
```

Este evidentă complexitatea limbajului de manipulare a datelor, care poate exprima descrieri şi relaţii. Prima parte a programului trebuie să iniţializeze variabilele intermediare. Apoi, combinaţia "*încăperi_mobilier*" va trebui parcursă în manieră secvenţială pentru a putea accesa "*descriere*", ceea ce se realizează cu ajutorul unui cursor manipulat de comenzile *find first* şi *find next*. Ultima parte a programului caută în mod secvenţial în acelaşi ansamblu "*încăperi_mobilier*", pentru a selecţiona o anumită "*descriere*" de "*mobilier*" şi pentru a o afişa. În acest exemplu, limbajul de manipulare a datelor este complet procedural, deoarece nu consideră decât o succesiune de comenzi (*find*, *get* etc.) integrate unor instrucţiuni-cod imperative.

Aceeaşi schemă conceptuală şi manieră de manipulare a datelor, pot fi exprimate simplu, prin intermediul modelului relaţional. Se prezintă în *Exemplul nr. 6*, definiţia obţinută prin utilizarea limbajului de definire a schemei relaţionale SQL:

Exemplul nr. 6

```
create table ansamblu_cetate (
număr 1 char (5),
nume 1 char (30) )

create table încăperi (
număr 1 char (5),

suprafața integer,
număr 1 char (5) )

create table mobilier (
număr 3 char (5),
nume 3 char (30),
culoare integer,
înălțime integer,
descriere char (30),
încăpere char (5) )

create table încăperi ale cetății (
număr 1 char (5),
număr 2 char (5) )

create table mobilier al încăperii (
```

Se remarcă faptul că modelul relațional prezintă în aceeaşi manieră entitățile "*ansamblu_cetate*", "*încăperi*" şi "*mobilier*", iar legăturile între entități sunt: "*încăperi ale cetății*" şi "*mobilier al încăperii*". Caracteristicile care completează o înregistrare apar ca nişte "sub-înregistrări" ale înregistrării pe care o referă. În plus, această descriere a schemei conceptuale a aplicației nu conține nici o informație asupra modalității de stocare fizică a datelor. Comenzi separate pot defini indecşii pentru unul sau mai multe atribute ale unei relații.

Manipularea datelor în modelul relațional se poate face în manieră declarativă. Construcția *select ... from ... where* a limbajului SQL permite să se exprime operații de filtrare complexă într-o sintaxă compactă *[9], [11]*.

3.5. Filtrarea structurilor grafice utilizând modelul relațional

3.5.1. Model relațional, schemă relațională

Modelul relațional este fundamentat pe conceptul matematic de relație, definită ca ansamblul produselor carteziene ale diferitelor domenii. Interesează doar relațiile finite, chiar dacă domeniile pe care sunt constituite sunt infinite *[3]*.Fiecare element al relației, adică fiecare relație elementară, se numeşte n-uplet. Numărul coloanelor din matricea unei relații se înlocuieşte cu nume, iar acestea sunt atributele. Fiecărui atribuit îi este asociat un domeniu. O relație este ansamblul funcțiilor parțiale (fiecare funcție reprezintă un n-uplet), care asociază atributelor elemente din domeniul lor.

Relațiile se regăsesc în scheme relaționale, care conțin un nume de relație şi o listă a atributelor care o alcătuiesc, pentru domeniul asociat. Schema relațională conține şi definițiile constrângerilor de integritate (regulile cu rolul de a specifica instanțele semnificative pentru aplicație). O parte importantă a teoriei sistemelor relaționale se ocupă cu studiul constrângerilor din punct de vedere al capacității lor de a influența modelarea unui proces şi, de asemenea, maniera în care un sistem poate să le verifice. Un exemplu particular important este cel al dependenței funcționale.

Modelele care utilizează pentru reprezentare schema conceptuală, folosesc noțiunea de "cheie", definită ca valoare care permite să se identifice şi/sau să se regăsească una sau mai multe înregistrări de un tip determinat. În contextul foarte formalizat al modelului relațional, se poate da o definiție mai precisă a "cheii" ca fiind: un ansamblu de atribute X este o cheie a relației R dacă nu există doi n-upleți diferiți în R, având aceeaşi valoare pentru atributele X şi dacă nici un subansamblu al lui X, nu verifică această proprietate.

Dacă utilizatorul unui ***SGBD*** introduce noi date în bazele de date, nu există întotdeauna o informație completă asupra datelor noi introduse. Pentru a rezolva problema informației incomplete, care este o problemă practică foarte des întâlnită, se pot utiliza valorile nule. Numeroase lucrări teoretice au fost consacrate definirii semanticii acestor valori nule şi comportamentului lor în condițiile de interogare sau actualizare a bazei de date *[4], [7], [9], [11]*.

3.5.2. Sisteme relaționale, standarde

Algebra relațională este o metodologie care permite definirea şi reprezentarea ansamblului operatorilor care permit manipularea datelor. Se definesc cinci operatori primitivi şi, pornind de la aceştia, se pot defini alții derivați. Fiecare operator generează o nouă relație. Operatorii primitivi cu care operează algebra relațională sunt:

a. Proiecția Operatorul pentru proiecție, notat π, suprimă coloanele unei relații. Proiecția relației "*încăperi*" pe atributul "*nume*" se notează "$\pi_{nume}(încăperi)$". Această proiecție dă ansamblul numelor încăperilor şi ignoră informațiile conținute în relație.

b. Selecția Operatorul pentru selecție, notat cu σ, permite caracterizarea unui subansamblu al relației utilizând un predicat.

c. Reuniunea Operatorul pentru reuniune consideră argumentul pentru două relații având aceleaşi atribute şi reuneşte n-upleții acestor două relații. Domeniile cu atribute corespondente în aceste două relații, trebuie să fie identice.

d. Diferența Operatorul pentru diferență se aplică în mod egal relațiilor compuse din aceleaşi atribute şi având aceleaşi domenii. Diferența a două relații este alcătuită din n-upleții care se află în prima relație şi nu figurează în cea de-a doua.

e. Produsul cartezian Produsul cartezian găseşte ca argument două relații. Fiecare n-uplet din relația rezultantă este alcătuit dintr-un n-uplet al primei relații şi dintr-un n-uplet al relației a doua.

Pornind de la aceşti cinci operatori primitivi se pot constitui alți operatori care se dovedesc utili şi anume:

f. Intersecția: intersecția a două relații se poate obține pornind de la reuniunea şi diferența lor;

g. Diviziunea: dacă R şi S sunt două relații constituite cu atributele A_1, A_2 .... A_n şi B_1, B_2 .... B_m, respectiv, atunci relația R ÷ S constituită pe atributele A_1, A_2 .... A_n este compusă din acei n-upleți a_1, a_2 .... a_n care, pentru toți n-ulpleții b_1, b_2 .... b_p ai lui S, are n-upleții a_1 .... a_nb_1 .... b_p aflați în R.

Unul din punctele tari ale teoriei relaționale este că algebra relațională şi calculul relațional au aceeaşi putere de expresie. Orice ansamblu care se poate caracteriza printr-o formulă de calcul, poate să fie obținut printr-o expresie algebrică şi reciproc. Calculul relațional se află la baza limbajelor utilizate în sistemele relaționale şi permite specificarea datelor regăsite într-o manieră neprocedurală.

Securitatea datelor în SQL este asigurată de un mecanism de autorizare care utilizează comanda *grant*, care permite să se suprime sau să se acorde unui utilizator drepturi asupra bazei de date. În exemplul utilizat în acest capitol, proprietarul relației "*obiecte de mobilier*" (utilizatorul având creată această relație) poate controla accesul la relație, scriind:

Exemplul nr. 7

```
grant select insert on mobilier to operator
grant all on încăperi to operator with grant option
```

Prima comandă autorizează pe "*operator*" să interogheze relația "*mobilier*" şi să-i adauge n-upleți, dar acesta nu poate nici şterge, nici modifica date. Cuvintele cheie *delete* şi *update* pot fi utilizate pentru a defini aceste drepturi speciale.

Performanțele limbajelor relaționale sunt relativ limitate. Limbajele relaționale (cum este SQL) sunt relativ limitate în definirea intrărilor/ieşirilor şi, mai general, în definirea dialogului cu utilizatorul.

Inițierea programării unei aplicații cu BD este făcută de utilizatorii finali ai sistemului finit, aceştia accesând datele prin intermediul unor interfețe specializate.

Pentru a atenua aceste deficiențe şi a permite construirea de interfețe specializate, este necesar să se utilizeze un limbaj de programare general şi să se acceseze acesta cu limbajul de manipulare a datelor relaționale.

Exemplul următor (Exemplul nr. 8) prezintă modalitatea în care poate "plonja" limbajul SQL în interiorul limbajului C, pentru a calcula închiderea tranzitivă a unei relații:

Exemplul nr. 8

```
1.  exec sql begin declare section
2.      int dimensiune;   /* o variabilă C cunoscută de SQL */
3.      exec sql end declare section
4.      int ultima_dimensiune;    /* o variabilă C necunoscută de SQL */
5.      exec sql execute immediate
6.          create table Traiectorie (.
7.                  pornire char (20),
8.                  sosire char (20))
9.      exec sql execute immediate
10.         insert into Traiectorie
11.                 select *
12.                 from Etapă
13. exec sql prepare o_iteraţie from
14.         insert into Traiectorie
15.                 select Traiectorie.pornire, Etapă.sosire
16.                 from Traiectorie, Etapă
17.                 where Traiectorie.sosire = Etapă.pornire
18. exec sql prepare calculează_dimensiune from
19.         select count (distinct *)
20.         from Traiectorie
21. dimensiune = 0
22. ultima_dimensiune = -1
23. while (dimensiune != ultima_dimensiune) {
24. ultima_dimensiune = dimensiune;
25.         exec sql execute o_iteraţie;
26.         exec sql declare cursor cursor for calc_dimesiune;
27.         exec sql open cursor;
28. /* Se ştie că rezultatul conţine un singur n-uplet */
29.         exec sql fetch cursor into: dimensiune
30.         exec sql close cursor;
31. }
```

Asupra utilizării modelelor relaţionale la definirea datelor care compun obiecte grafice complexe, se pot concluziona următoarele:

1. *simplicitatea conceptelor şi a schemei:* modelul reţea amestecă definiţia schemei conceptuale şi a celei fizice; în plus, schemele obţinute sunt greu de înţeles şi de modificat; ele pot fi elaborate de programatori experţi; în sistemul relaţional, schema se compune dintr-o listă de tabele, iar informaţiile fizice se definesc separat;

2. *suportul teoretic solid:* modelul relaţional a propus o teorie simplă şi uşor de asimilat, care a permis dezvoltatorilor de ***SGBD***-uri să lucreze într-o manieră similară celei care se utilizează de mult timp pentru limbaje de programare;

3. *nivelul ridicat de independenţă a datelor:* acest nivel este interesant mai ales din punct de vedere al independenţei fizice a datelor şi reprezintă un progres important în raport cu sistemele precedente;

4. *limbajul de manipulare de nivel înalt:* limbajul de manipulare a datelor în modelul reţea se rezumă la o colecţie de comenzi care sunt total procedurale; limbajele relaţionale (de exemplu, SQL) sunt declarative, exprimă informaţiile care să fie extrase din bazele de date şi nu modalitatea prin care se obţin aceaste informaţii;

5. *integritatea şi confidenţialitatea:* sistemele care utilizează modelul reţea sunt deficitare în ceea ce priveşte soluţionarea problemei securizării datelor; sistemele relaţionale, prin utilizarea limbajelor de nivel înalt care permit specificarea cerinţelor de integritate, asigură facilităţi suplimentare, în acest domeniu;

6. *optimizarea accesului la bazele de date:* optimizarea este un element esenţial într-un cadru relaţional; în practică, sistemul trebuie să-şi găsească el însuşi strategiile

de optimizare şi metodele de acces; o mare parte din tehnologia relațională a fost consacrată ameliorării tehnicilor de optimizare;

7. *manipularea datelor în mod global:* modelul relațional oferă limbaje de manipulare a datelor care permit lucrul cu ansamble de date; avantajele sunt: gestionarea automată a dublurilor şi utilizarea paralelismului pentru manipularea ansamblelor mari.

Din păcate, comunicarea între un limbaj ca SQL şi un limbaj general se face "n-uplet" cu "n-uplet" şi nu "ansamblu" cu "ansamblu". Se pierd astfel beneficiile oferite de manipularea globală a datelor de către SQL. Principalele limitări ale sistemelor relaționale provin din faptul că oferă un model de date prea simplu şi limbaje de manipulare prea limitate. În mod particular, modelele relaționale fac față bine reflectării legăturilor complexe existente în aplicații care manipulează volume mari de date. Un aspect negativ este necesitatea imersiunii acestor limbaje în limbajele generale, cărora le provoacă disfuncționalități.

În mod special interesează modul în care un ***SGBD*** poate gestiona datele grafice şi în care poate suporta interfețe foarte elaborate.

Unul dintre cele mai puternice limbaje structurate pentru interogarea bazelor de date relaționale îl constituie SQL (Structured Querz Language). Acesta a devenit ***standard pentru SGBD***, limbajul SQL permițând comunicarea complexă şi rapidă a utilizatorului cu bazele de date.

Standardizarea limbajului SQL este acceptată de marea majoritate a SGBD, care recunosc principalele instrucțiuni ale SQL (de exemplu, Oracle, Access, Sybase) *[11], [13], [16], [20], [21], [23], [25], [26], [27].*

ANSI (American National Standards Institute) recunoaşte oficial SQL ca standard, acceptând din acesta următoarele aspecte: definirea, interogarea şi manipularea datelor, procesarea tranzacțiilor, integritatea informațiilor complexe, joncțiunile externe. Majoritatea marilor producători de ***SGBD*** furnizează propriile extensii ale limbajului SQL, asigurându-şi astfel exclusivitatea. Între aceştia: Oracle, Sybase, IBM, Informix, Microsoft. Ca exemplu, *tehnica grafică QBE* (Query by Example) din *Access 2000* permite proiectarea de interogări complexe.

Un alt exemplu îl constituie *serverele Oracle 7* care administrează baze de date cu toate avantajele unei structuri relaționale, având în plus capacitatea de a stoca obiecte grafice complexe. Acestea asigură un motor PL/SQL în interiorul serverului Oracle, care permite facilități grafice şi de comunicare superioare. *JDBC (Java Database Connectivity*) este o interfață standard SQL de acces la baze de date cu obiecte complexe. Acesata conține seturi de clase şi de interfețe scrise în Java, care furnizează mecanisme standard pentru proiectanții aplicațiilor cu baze de date.

3.6. Particularitățile proiectării bazelor de date pentru aplicații de realitate virtuală

3.6.1. Cerințe pentru aplicații de realitate virtuală

Aplicațiile de ***RV*** au particularități care influențează proiectarea bazelor de date şi structura ***SGBD,*** printre care şi aceea că datele manipulate sunt foarte complexe, eterogene şi aflate în corelații complicate între ele. O aplicație de ***RV*** conține şi o parte importantă de informație de tip documentar. Se prezintă principalele caracteristici ale unei aplicații de ***RV*** care interesează proiectarea bazelor de date:

- *Datele din bazele de date sunt clone ale obiectelor din lumea reală:*

Există o legătură directă şi explicită între entitățile administrate de aplicație şi

obiectele lumii reale; deseori aplicațiile de ***RV*** conțin module care asigură conexiunea cu sisteme fizice, ceea ce impune proiectarea accesului la/ de la date şi periferice.

- *Datele sunt organizate în ierarhii complexe:*

Un sistem de ***RV*** manipulează relații, legături, comunicații de componente şi subcomponente, precum şi o ierarhie a compoziției; această ierarhie joacă un rol foarte important în prelucrările care se realizează, eficacitatea lor punând în evidență modul în care ierarhia condiționează performanțele globale ale sistemului.

- *Elaborarea specificațiilor tehnice şi conceptuale este un proces interactiv:*

Interactivitatea are numeroase consecințe asupra sistemului care trebuie să suporte aplicația, esențială fiind calitatea interfeței utilizator; sistemul trebuie să suporte modificările pe care le antrenează această activitate; trebuie să asigure capacitatea de a modifica nu numai datele din baza de date, ci şi meta-datele, adică schema de ierarhie, care este variabilă; o altă consecință a interactivității este faptul că sistemul trebuie să furnizeze un mecanism de generare a versiunilor care să permită definirea şi manipularea alternativelor de soluții conceptuale, precum şi să arhiveze versiunile anterioare; de cele mai multe ori, aplicația de ***RV*** nu suprimă date, ci acumulează în versiuni succesive întregul istoric al procesului derulat.

- *Lucrul simultan cu date partajate:*

Această caracteristică a aplicațiilor de ***RV*** justifică, în plus față de volumul imens de date pe care le tratează, utilizarea unui ***SGBD*** performant ca suport; partajarea datelor de o manieră coerentă este o cerință esențială a acestui tip de aplicație.

Controlul concurenței la utilizarea datelor şi partajarea acestora sunt concepte diferite şi se pot realiza în maniere diferite. Aplicațiile de ***RV*** administrează şi date netradiționale (imagini, sunet, text documentar etc). Aspectul eterogen al datelor se adaugă dificultăților induse de volumele mari de date. Un ***SGBD*** care să suporte aplicații de ***RV*** incluzând trasee geografice (***GIS***), trebuie să poată administra atât tratamentul clasic asupra imaginii tridimensionale, cât şi legăturile de orice tip între ele. Trebuie să permită selecții rapide de imagini stocate. Aceasta combină tehnicile de căutare multicriterială, de acces optimizat la bazele de date, de tratare imagini 3D etc.

Produsele de ***RV*** sunt atât de specifice, încât sistemele relaționale nu reuşesc să le rezolve în totalitate, ceea ce determină definirea unor sisteme specifice, dedicate funcțiunilor ***RV***. Este dificil de alcătuit rețete pentru a rezolva specificitatea problematicii sistemelor ***RV,*** deoarece acestea impun posibilitatea de a defini în mod imediat legături complexe între obiecte grafice complicate, în acelaşi timp cu dirijarea accesului partajat şi distribuit la date, ceea ce este relativ simplu atunci când se manipulează relații, dar este mai dificil de realizat atunci când se manipulează înregistrări şi atribute complexe.

3.6.2 Obiective pentru asigurarea funcționalității aplicațiilor de realitate virtuală

Conceptele de baze de date, de rețea, de sistem distribuit capătă noi dimensiuni prin apariția interfețelor om-maşină. Acestea impun dezvoltarea unor noi generații de produse software, care să răspundă mai bine cerințelor de integrare: integrarea limbajelor de programare pentru sistemele de exploatare, integrarea tehnologiilor pentru interfețe om-maşină cu bazele de date, integrarea instrumentelor specifice permițând gestionarea aplicațiilor complexe. ***SGBD***-urile de ultimă generație sunt în acelaşi timp, sisteme integrate, dar şi sisteme integratoare (extensibile), iar obiectivele pe care le satisfac sunt următoarele:

- optimizarea programării, bazată pe integrarea ***SGBD*** cu un limbaj de

programare care să permită rezolvarea problemei tratamentului relațional;

- gestiunea suplă a datelor şi a meta-datelor, permițând aplicații interactive;
- integrarea unor medii de dezvoltare care utilizează realizările recente în domeniul interfețelor om-maşină;
- lucrul cu periferice specifice aplicațiilor de ***RV*** în manieră interconectată; puterea de calcul şi capacitatea de stocare trebuie să asigure necesitățile acestor periferice;
- extensibilitatea (să asigure utilizarea şi integrarea unor aplicații de natură foarte diferită).

4. METODA ORIENTATĂ-OBIECT APLICATĂ ÎN DOMENIUL REALITAȚII VIRTUALE

4.1. Metodele de analiză / proiectare pentru sistemele de *RV*

Se impune cunoaşterea următoarelor definiții şi accepțiuni noționale:

- *problemă* : chestiunea propusă pentru soluționare;
- *domeniu*: sfera sau câmpul de activitate sau influență;
- *sistem*: un set sau un aranjament de obiecte aflate în corelație sau conexiune cu o unitate sau un întreg;
- *responsabilitate*: condiția, calitatea, faptul sau instanța față de care trebuie să răspundă, să se subordoneze sau să se lege un operator, un obiectiv, un serviciu etc.

Se definesc patru metode de analiză, dar se recomandă combinarea acestora pentru rezolvarea unei situații date. Fiecare abordare este definită ca o ecuație pentru recunoaşterea metodei optime.

Descompunerea funcțională. O strategie des folosită este selectarea proceselor *paşi* - *subpaşi* anticipate la noul sistem. Se utilizează experiența previzională pentru sisteme similare, combinată cu examinarea cazurilor de excepție.

Abordarea fluxurilor de date. O altă metodă de a descrie domeniul-problemă într-o reprezentare tehnică, este abordarea fluxului de date sau analiza structurată. Documentația aferentă trebuie să includă specificațiile fluxurilor de date şi transformările între nivele, dicționarul datelor şi specificațiile de proces. Nivelul de bază al diagramei dă o imagine a realității, specificațiile conțin expresia detaliilor.

Modelarea informațiilor. Se utilizează modelarea entităților şi corelațiilor, modelarea informațiilor şi modelarea datelor semantice. Un obiect este simbolul sub care se prezintă una sau mai multe părți ale unei entități. S-au dezvoltat două strategii de modelare. Prima dezvoltă şi identifică lista atributelor, adaugă corelații, defineşte super- şi sub-tipurile (prin extragerea atributelor comune) şi obiectele asociate (prin descrierea conexiunilor). A doua strategie constă în reducerea redundanței datelor şi acționează la fel cu prima, cu excepția primului pas: căutarea obiectelor în lumea reală şi descrierea lor cu atribute.

Orientarea – obiect. Sistemele de ***RV*** se bazează în principal pe tehnologii de dezvoltare integrate în arhitecturi orientate obiect. Abordarea ***OOA*** presupune cinci activități majore: căutarea claselor şi obiectelor; identificarea structurilor; identificarea subiecților; definirea atributelor; definirea serviciilor.

Acestea sunt activităţi obligatorii, nu paşi secvenţiali. Activităţile ghidează etapele de analiză de la nivelele superioare de abstracţie (domeniul-problemă, clase şi obiecte) spre nivelele inferioare de abstractizare (structuri, atribute şi servicii). Aceste cinci activităţi reprezintă cea mai comună abordare a ***OOA***. În fapt, se preferă să se parcurgă calea Clase Obiecte → Atribute → Structuri → Servicii. În alte situaţii, se parcurge traseul Clase şi Obiecte → Servicii → Structuri → Atribute.

Metoda de analiză orientată-obiect (***OOA***) armonizează conceptele modelării informaţiei şi sistemelor de baze de cunoştinţe. ***OOA*** mapează domeniul-problemă şi responsabilităţile sistemului direct într-un model. Nu este recomandată pentru sisteme cu responsabilităţi foarte limitate sau care au doar una sau două clase şi obiecte.

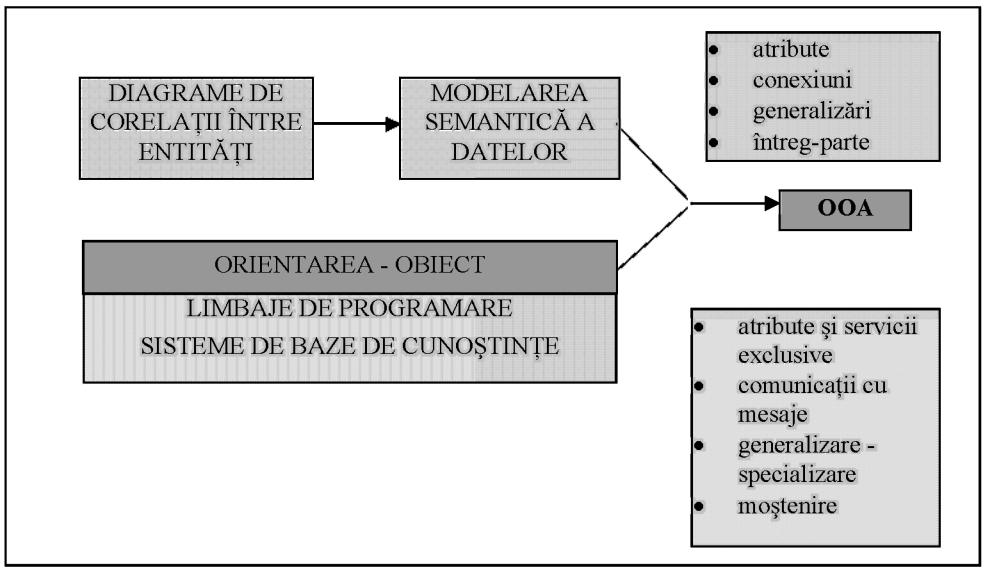

Figura 4.1. Îmbinarea disciplinelor în 00A

O sistematizare a motivaţiilor şi avantajelor folosirii ***OOA*** *[4], [7], [21]*:

1. permite angajarea în domenii mai dificile;

2. îmbunătăţeşte interacţiunea cu expertul în domeniul ***RV,*** organizează analiza şi specificaţiile utilizând metode uşor accesibile reprezentării;

3. creşte consistenţa internă, reduce timpii prin tratarea atributelor şi serviciilor ca un tot intrinsec;

4. reprezintă explicit caracteristicile comune; foloseşte moştenirea pentru a sublinia şi identifica atributele şi serviciile comune, pe care le capitalizează;

5. construieşte specificaţii elastice la schimbare, împachetează părţile dinamice din interiorul problemei, furnizând stabilitate la schimbarea cerinţelor;

6. reutilizează rezultatele prin sistematizarea şi implementarea practică în sistem; organizează rezultatele pe baza construcţiei domeniilui de bază;

7. asigură consistenţa fundamentală a analizei (ce este de construit) şi reprezentării (cum trebuie construit); stabileşte o continuitate a reprezentării pentru expandarea rezultatelor analizei în implementări specifice.

4.2. Analiza orientată- obiect pentru domeniul *RV*

4.2.1. Conceptele şi principiile fundamentale ale metodei

Conceptele majore ale complexităţii domeniului şi ale definirii

responsabilităților sistemului sunt, şi în cazul aplicațiilor de ***RV***, următoarele:

- abstracția;
  - proceduri;
  - date;
- încapsularea;
- moştenirea;
- asocierea;
- comunicarea cu mesaje;
- întrepătrunderea metodelor de organizare;
  - obiecte şi atribute;
  - întreg şi părți;
  - clase şi membri;
- proporția;
- categoriile de comportament;
  - cauzalitatea imediată;
  - schimbarea în timp;
  - similitudinea funcțiilor.

Diferite metode de analiză încorporează o parte sau toate aceste principii şi concepte.

4.2.2. Terminologie şi notații specifice

Obiectul se defineşte ca fiind o abstracție a ceva aflat în domeniul-problemă şi reflectând capabilitățile sistemului de a da informații asupra sa, de a interacționa cu el sau cu întregul; se defineşte drept o încapsulare a valorilor atributelor şi a serviciilor.

Clasa este o descriere a unuia sau mai multor obiecte cu un set uniform de atribute şi servicii, incluzând ceea ce crează noi obiecte în clasă. O primă motivație a necesității identificării Claselor şi Obiectelor este o reprezentare mai tehnică a sistemului de ***RV*** ca imagine conceptuală. Clasa şi Obiectul reprezintă expresia inițială a contextului.

Termenul *structură* este definit pentru a reflecta atât domeniul problemei cât şi responsabilitățile sistemului. Structura este o expresie a complexității domeniului problemei care ține de responsabilitățile sistemului.

a) *Structura "Gen-Spec" (general – particular)*. Poate fi definită ca o încercare de delimitare între clase. Dacă sunt posibile mai multe specializări, este util să se considere cea mai simplă specializare; se pot elabora alte specializări pornind de la aceasta şi se folosesc variante ale uneia deja tratate.

b) *Structura "Întreg-Parte"*. În tratarea domeniului ***RV*** se dovedeşte foarte utilă identificarea claselor şi obiectelor la nivelul cel mai înalt al domeniului problemei şi la nivelul cel mai înalt al responsabilităților sistemului. Structura „*Întreg – Parte*" este reprezentată printr-un obiect "întreg" (din simbolul Clasă - Obiect) în vârf şi apoi un obiect "parte" la bază. Plasarea continuă a "întregului" mai sus şi "părților" mai jos conduce la un model mai uşor de înțeles, dar acest lucru este deliberat pentru că se doreşte evidențierea faptului că un "întreg" are un anumit număr de părți. Un "întreg" poate avea şi tipuri diferite de părți. Fiecare obiect poate fi considerat ca o "parte" potențială şi de asemenea ca un potențial "întreg". Structura "Întreg-Parte" consideră următoarele noțiuni de bază:

- ansamblul – partea;
- conținut – ambalaj;

- colecție - membri (şi cu alte variante).

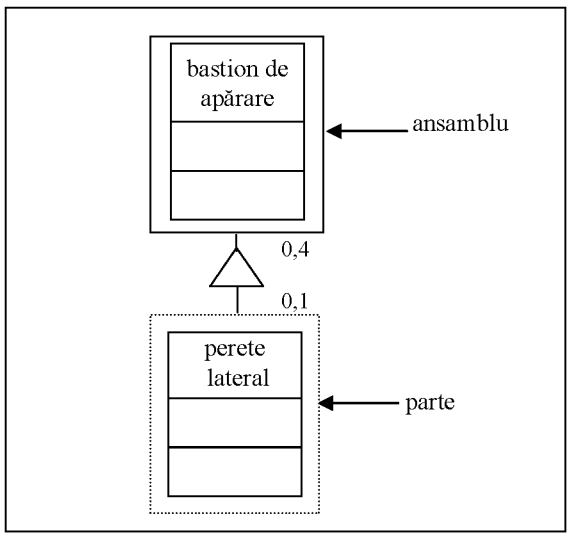

Figura 4.2. Structură ansamblu - parte

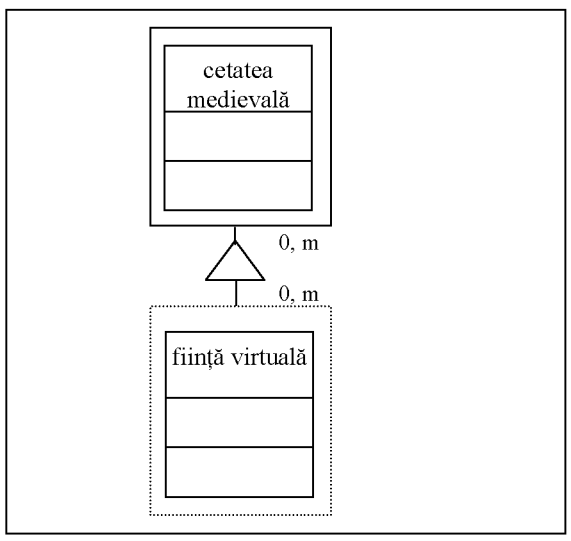

Figura 4.3. Structura container-conținut

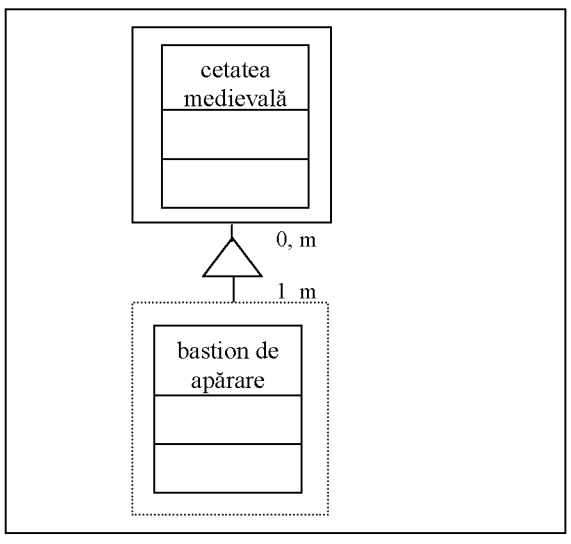

Figura 4.4. Structura colecție-membri

Subiectele. În ***OOA***, termenul "subiect" este definit ca reflectând domeniul-problemă şi responsabilităţile sistemului. Este mecanismul pentru ghidarea utilizatorului spre un model amplu, complex.

Atributele. În analiza orientată pe obiecte, termenul de atribut este definit pentru a reflecta atât domeniul problemei cât şi responsabilităţile sistemului. Un atribut este o dată (informaţie de stare) pentru care fiecare obiect dintr-o clasă are propria sa valoare. Modelul ***OOA*** descris astfel devine mai specific şi mai detaliat. Fiecare clasă şi obiect este descris cu mai multe detalii într-o specializare "Clasă şi Obiect". Atributele adaugă detalii la "Clasă şi Obiect", de aceea alegerea atributelor implică o activitate laborioasă de analiză şi selecţie.

Serviciile. O etapă importantă în definirea „Serviciilor" este definirea comunicării necesare dintre „Obiecte". Comenzile şi cerinţele sunt specifice modului în care natura umană interacţionează cu un sistem şi aceeaşi paradigmă de interacţionare este folosită între părţile modelului *OOA*. Serviciile şi Conexiunile Mesaj sunt specificate în "Clasă şi Obiect", stabilind necesităţile măsurabile şi observabile.

Fiecare obiect care compune mediul virtual trece prin diferite stări, din momentul în care a fost creat şi pe măsură ce evoluează. Starea unui obiect este reprezentată de valorile atributelor sale. Fiecare schimbare a valorilor atributelor reflectă o schimbare de stare. Pentru a identifica starea unui obiect trebuie examinate valorile potenţiale pentru atribute şi examinat dacă aplicaţia include un comportament diferit pentru aceste valori potenţiale. De asemenea, se verifică rezultatele anterioare ale stărilor din acelaşi domeniu, pentru a stabili care stări ale obiectului pot fi direct utilizate. Diagramele de stare ale obiectului prezintă stări sau moduri de comportament în decursul timpului şi tranziţiile; comportamentul detaliat şi schimbările în comportament sunt definite în strânsă legătură cu specificarea Serviciilor.

4.3. Proiectarea alocării resurselor globale

Proiectantul unei aplicaţii de ***RV*** trebuie să identifice resursele globale şi să determine mecanismele optime pentru a controla accesul la aceste resurse. Resursele globale includ: unităţi fizice (de la procesoare, la sateliţi de comunicaţie); spaţiul (spaţiul de disc, ecranul unei staţii de lucru sau butoanele unei mănuşi de date); nume logice (obiecte, nume de fişiere, nume de clase); accesul la date publice, cum ar fi bazele de date etc. Dacă resursa este un obiect fizic, aceasta poate fi controlată prin stabilirea unui protocol de acces printr-un sistem concurent. Dacă resursa este o entitate logică (cum sunt bazele de date), apare pericolul unui conflict de acces.

Task-urile independente pot folosi simultan acelaşi obiect. Fiecare resursă globală trebuie să fie în "proprietatea" unui "obiect gardian" care îi controlează accesul către el. Un astfel de obiect poate controla mai multe resurse; întregul acces la resurse trebuie să treacă prin obiectul gardian. O resursă logică poate fi de asemenea partiţionată, asfel încât subseturile să fie repartizate la diferite obiecte gardian, pentru asigurarea unui control independent. De exemplu, o strategie pentru generarea unui obiect într-un mediu paralel distribuit, ar fi aceea de a prealoca un şir de stări posibile către fiecare procesor dintr-o reţea. Fiecare procesor alocă stări ale obiectului în interiorul şirurilor pre-alocate, fără a fi nevoie de o sincronizare globală.

Într-o aplicaţie de ***RV*** care se desfăşoară în timp-critic, costul trecerii întregului acces printr-un "obiect gardian" este uneori prea ridicat şi clienţii trebuie să acceseze resursa direct. În acest caz, cheile pot fi plasate în subseturile resursei. O „cheie" este

un obiect logic asociat cu un subset (sau mai multe) definit al unei resurse care îi dă unui deținător de chei dreptul de a accesa resursa direct. Totuşi, trebuie să existe un „obiect gardian" pentru a aloca cheile, dar după ce are loc interacțiunea cu „gardianul" pentru a obține o cheie, utilizatorul resursei poate accesa resursa în mod direct.

Această abordare este periculoasă şi folosirea accesului direct la resurse ar trebui descurajată în aplicațiile de ***RV***, în afara cazurilor în care este neapărat necesară. Se foloseşte reprezentarea pentru vizualizarea rezolvării problemei, mai întâi la un nivel de ansamblu şi apoi la nivele ce cresc progresiv în detalii.

Aceste considerente influențează structura şi arhitectura generală a sistemului. Arhitectura prevede contextul în care decizii detaliate sunt luate în faze de reprezentare târzii. Prin luarea deciziilor ce pot fi aplicate întregului sistem, se împarte problema în subsisteme. Reprezentarea unui sistem de ***RV*** trebuie să respecte următoarele:

- organizarea sistemului în subsisteme;
- identificarea inerentelor suprapuneri;
- alocarea de subsisteme la procesoare, task-uri şi periferice;
- alegerea unui management potrivit pentru stocările de date;
- asigurarea accesului printr-un supervizor la resursele globale;
- alegerea modalităților de control al perifericelor specifice (ochelari, mănuşi de date, mouse spațial etc);
- setarea priorităților etc.

Pentru toate aspectele, primul pas este divizarea sistemului într-un număr mai mic de componente (subsisteme). Un subsistem nu este un obiect şi nici o funcție, ci un pachet de clase, asociații, operații, evenimente şi constrângeri ce sunt înrudite între ele. Subsistemul are o interfață bine definită cu alte subsisteme. Un subsistem este identificat prin serviciile pe care le oferă. Un serviciu este un grup de funcții înrudite ce împart scopuri comune. Un subsistem defineşte un mod coerent de a privi un aspect al problemei. Fiecare subsistem are o interfață bine definită cu restul sistemului. Această interfață specifică forma tuturor interacțiunilor şi circulația informației de-a lungul granițelor subsistemului, dar nu specifică cum este implementat în interior subsistemul. Fiecare subsistem poate fi reprezentat independent, fără să afecteze alte subsisteme. Subsistemele ar trebui să fie definite astfel încât majoritatea interacțiunilor să aibă loc în interiorul subsistemelor şi nu de-a lungul granițelor acestora, pentru a reduce dependențele între subsisteme.

Relația dintre două subsisteme poate fi de tip client-server sau de tip punct cu punct. În cazul relației client-server, clientul apelează server-ul care îi îndeplineşte cereri de servicii şi îi întoarce un rezultat. Clientul trebuie să cunoască interfața server-ului, dar pentru server nu este necesar să ştie interfețele clienților săi, pentru că toate interacțiunile sunt inițiate de către clienți folosind interfața server-ului. În cazul relației punct cu punct, fiecare subsistem trebuie să apeleze celelalte subsisteme. O comunicare de la un susbsistem la altul nu este în mod necesar urmată de un răspuns imediat. Relațiile punct cu punct sunt mult mai complicate, deoarece subsistemele trebuie să îşi cunoască unul altuia interfețele.

Fiecare subsistem concurent trebuie alocat unei unități hardware, unui procesor sau unei unități funcționale specializate. Alocarea resurselor trebuie să realizeze:

- să estimeze performanțele şi resursele necesare pentru a le satisface;
- să aleagă implementările hardware şi software optime pentru subsisteme;

- să aloce subsisteme software procesoarelor ce satisfac nevoile şi să minimizeze comunicarea interprocesor;
- să asigure conectivitatea unităţilor fizice care sunt implementate în sistem.

În aplicaţiile de ***RV,*** hardware-ul poate fi privit ca o formă rigidă a software-ului. Fiecare driver este un obiect care operează în acelaşi timp cu alte obiecte (alte drivere sau module software). Proiectantul trebuie să decidă ce subsisteme (funcţii) vor fi implementate în hardware şi care în software. Anumite funcţiuni sunt implementate permanent hardware din următoarele motive:

- hardware-ul prezintă funcţionalitatea cerută (de exemplu, este mai uşor să să cumpere un set de ochelari şi căşti 3D şi să se implementeze aceştia);
- pe parcursul exploatării, sunt cerute performanţe mai mari decât poate prevedea proiectantul, este necesară o configuraţie hardware eficientă;
- anumite task-uri sunt cerute pentru locaţii fizice specifice, pentru a controla hardware-ul sau pentru a permite operaţii independente sau concurente;
- timpul de răspuns sau rata fluxului de informaţie întrece comunicarea valabilă între un task şi o piesă hardware;
- ratele de calcul sunt prea mari pentru un singur procesor, de aceea cerinţele task-urilor trebuie repartizate mai multor procesoare, acele subsisteme care interacţionează cel mai frecvent trebuie atribuite aceluiaşi procesor, pentru a minimiza costurile de comunicare, iar subsistemele independente trebuie repartizate unor procesoare diferite.

5. MANAGEMENTUL STOCĂRII DATELOR

5.1. Principiile metodei "orientate-obiect" în organizarea datelor

Modul de organizare al datelor este impus de: cerinţele concrete ale aplicaţiei; limbajele de programare utilizate şi de tipul şi structura efectivă a datelor cu care lucrează aplicaţia. Influenţat de acestea, modul de organizare a datelor a evoluat de la câmpuri fixe la tabele interconectate, care folosesc limbaje de generaţia a patra (de exemplu, 4GL etc.) *[7], [12], [23],*(*Figura 5.1.*).

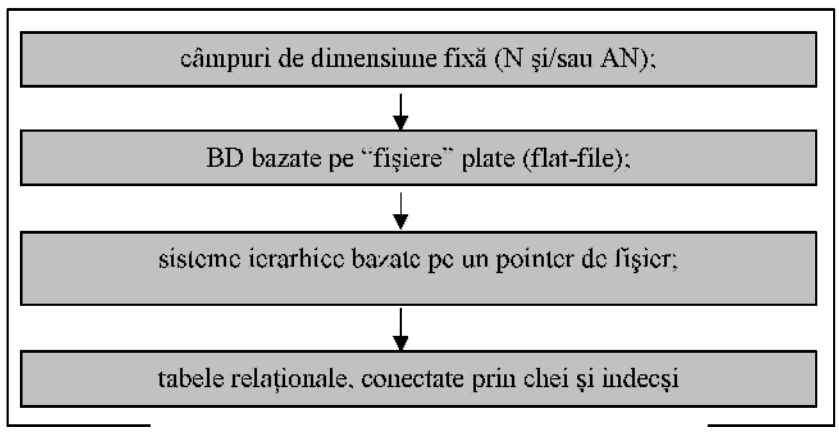

Figura 5.1. Evolutia organizării datelor

Datele cu care operează aplicaţiile ***RV*** sunt extrem de complexe şi provin din surse diferite; se manipulează cantităţi foarte mari de date eterogene (exemplu: date

audio, video, tridimensionale, bidimensionale, liniare, texturi, documente compuse, informații geografice, date geometrice / nongeometrice etc.). Toate acestea impun asigurarea unor capacități de stocare foarte mari şi sisteme de regăsire care să funcționeze ca rețele globale.

Soluția este utilizarea sistemelor cu Baze de Date Orientate-Obiect; acestea prezintă următoarele dezavantaje: complexitate foarte mare, schimbarea metodelor de lucru şi atitudinii proiectanților, impuse de noile utilizări şi sunt relativ scumpe;

Aceste dezavantaje sunt contracarate de următoarele aspecte ale utilizării lor:

- se integrează metodelor orientate obiect;
- se pretează arhitecturilor multi-strat;
- conțin interfețe pentru limbaje moderne, bazate pe obiecte;
- sunt disponibile pentru platforme diverse UNIX , Win NT;
- servesc bine şi aplicații bazate pe Web şi Internet;
- sunt alcătuite din blocuri modulare, care se pot asambla funcție de cerințele concrete ale aplicației;

Modelul bazelor de date relaționale a fost construit pe baza conceptelor teoriei mulțimilor, a tabelelor monolitice şi a unor gramatici simple pentru interogări ad-hoc (sunt reprezentate cel mai bine de SQL); acestea lucrează bine în timp real.

Modelul bazelor de date obiectuale se bazează pe conceptele orientării pe obiecte (reprezentarea datelor prin clase, atribute şi servicii etc.) şi pot fi stocate / regăsite de aplicații în forma naturală, fără a fi modificate pentru stocarea în tabele relaționale. Acestea asigură performanțe bune la lucrul în timp real, deoarece cele mai multe aplicații moderne prezintă arhitecturi client-server şi sunt programate în termeni de obiecte. De asemenea, se pretează la procesarea distribuită, sunt foarte rapide în cazul cererilor complexe şi acceptă structuri de date diferite.

Modelul relațional arată ca un tabel de linii şi coloane (obiect 2D). Modelul orientat pe obiecte respectă caracteristica reală a obiectelor complexe 3D şi este ideal pentru Internet, jocuri video, aplicații multimedia, comunicații.

*Criterii generale pentru alegerea unui SGBD pentru o aplicație de **RV***

Suport pentru limbaj: Ce limbaj este necesar - Java, C++ sau SQL ? Unele limbaje proprietare sunt mai rapide decât SQL, dar alegerea unui produs care foloseşte un limbaj standardizat este o soluție mai flexibilă şi mai uşor portabilă.

Scalabilitate: Care este cea mai mare bază de date pe care produsul este în stare să o suporte? Care este cea mai mare bază de date existentă care foloseşte produsul fără probleme? Câți utilizatori pot accesa simultan baza de date?

Securitate: Cum vor fi implementați algoritmii de securitate: la nivel de utilizator, grup sau la ambele nivele?

Metode: Cum stochează metodele?

Clase de colecții: Ce clase de colecții pot manipula baza de date? Bibliotecile Oracle, Java şi altele au anumite clase comune de colecții, iar utilizarea claselor de colecții standardizate creşte portabilitatea şi flexibilitatea.

Denumirea corectă ar trebui să fie "bază de obiecte" şi nu „bază orientată obiectual", deoarece scopul nu este de a stoca, manipula şi returna datele înglobate în obiecte, ci stocarea, manipularea şi returnarea chiar a obiectelor. Bazele de date relaționale permit efectuarea de interogări complexe asupra unui set de date. Bazele de date orientate obiect asigură interogări simple asupra unui set de date complexe. O bază de date orientată obiect clasică are metode, clase şi caracteristici ce descriu

modelul în motorul bazei de date. Obiectele sunt active. Datele din bazele relaționale sunt pasive şi este nevoie de un alt program pentru a lucra cu ele.

5.2. Modelul obiectelor active

5.2.1. Modele grafice

Modelele grafice trebuie să asigure posibilitatea manipulării entităților abstracte tratate: viteză, timp, evenimente ulterioare, acțiuni paralele, relații dintre obiectele active, sincronizarea comportamentului etc. Elementele grafice oferă o percepție intuitivă a legăturilor dintre obiectele abstracte manipulate în aplicații. Traiectoria este o evoluție spațio-temporală a cărei reprezentare poate fi manipulată.

Modelul obiectelor active are în vedere tratarea următoarelor probleme:

- descrierea unui sistem ce permite programarea vizuală, descrierea comportamentului obiectelor active;
- stabilirea unui set consistent de entități care permit o evoluție dinamică şi activă;
- descrierea unei structuri consistente, definirea obiectelor active, aplicațiilor interactive, evoluțiilor concurente şi manipularea obiectelor într-un spațiu abstract;
- descrierea unor entități: comportament, interactor, agenți;
- descrierea unui model capabil să formuleze comportamentul interactiv şi dinamic al unor entități active;
- modalități de definire, de modelare a structurii şi a noțiunilor: comportament, traiectorie, poziție, stare, regulă, condiție, expresie, acțiune, operație, parametri;
- definirea comportamentului spațio-temporal, folosind noțiunea de traiectorie spațială şi temporală;
- asigurarea unei prezentări grafice şi posibilitatea manipulării directe a noțiunilor abstracte;
- modelarea unui set minimal de acțiuni simple, care să permită construirea unor comportamente complexe;
- execuția concurentă într-o manieră pseudo-paralelă, folosind firele de execuție;
- sincronizarea executării unei acțiuni cu accesul la resursele partajate (structuri de date, entități model, fire de execuție).

Modelul obiectelor active are la bază un set de entități active şi entități pasive. Entitățile active pot lua următoarele forme: obiecte, variabile, atribute, poziție, stare, expresie, traiectorie, flag-uri etc. Entitățile pasive sunt: comportament, regulă, algoritm, condiție, operație, acțiuni, etc. Entitățile active au un comportament bine definit, caracterizat de o evoluție spațială şi temporală. Comportamentul obiectelor este sesizabil, în urma parcurgerii unei traiectorii caracterizată de un set de stări. Traiectoria asigură posibilitatea manipulării directe a entităților modelului şi a elementelor acestora. Prin manipulare directă, este posibilă definirea acțiunilor, a regulilor, a condițiilor care definesc fiecare stare de pe traiectoria entității.

În cadrul modelului obiectelor active, evoluția modelului este influențată de evoluția paralelă şi concurentă a entităților active. Fiecare entitate (obiect, flag, obiect complex, comportament) este supravegheată de un proces thread (fir de execuție). O mulțime de fire de execuție duc la definirea comportamentului, mişcării şi a

interacțiunii dintre entitățile active. Execuția unui comportament este o succesiune de acțiuni ale entității curente sau a entităților delegate. Entitățile comunică prin mesaje.

5.2.2. Interfața modelului

Modelul este o colecție de obiecte active care sunt guvernate de un comportament aleatoriu. Pentru definirea modelului se foloseşte manipularea directă asupra entităților componente. În acest mod sunt create, şterse şi instanțiate entități model, le este conturat comportamentul. Entitatea comportament este definită printr-un set de acțiuni condiționate. În funcție de satisfacerea condițiilor, în fiecare stare a traiectoriei se execută un anumit grup de acțiuni. Starea în care se află modelul poate fi alternată (modificată) de evoluția modelului sau de evenimente exterioare modelului. Evenimente exterioare modelului reactualizează starea modelului prin intermediul interfeței model-utilizator.

Evenimentele legate de interfață sunt manipulate de către *interactori* (agenți). Interactorii sunt obiecte speciale care supraveghează comunicația spre şi dinspre modulele externe, având rolul de a trimite şi recepționa mesaje. În acest context, evenimentul reprezintă modificarea unei stări. Interactorii transformă comunicația asincronă dintre model şi modulele sale externe, într-o comunicare de mesaje în interiorul modelului. Funcționalitatea şi structura modelului obiectelor active se bazează pe următoarele principii:

- entitățile fundamentale ale modelului sunt obiectele active;
- un obiect are ataşat un comportament;
- comportamentul, acțiunile şi mişcarea obiectului sunt executate de fire de execuție;
- entitățile model sunt obiecte încapsulate care comunică prin mesaje (asincron);
- entitățile model, precum şi atributele, parametrii, componentele, comportamentul, regulile, pozițiile entității pot fi modificate dinamic;
- comportamentul obiectelor din model este ordonat de timp şi spațiu.

Una din proprietățile intrinseci ale modelului este evoluția. Toate obiectele sunt entități active, care au un comportament bine definit. Comportamentul fiecărei entități este definit în mod dinamic prin operare directă asupra scenei de obiecte. Fiecărui obiect i se poate asocia un anumit comportament sau comportamentul poate fi moştenit. Evoluția modelului constă dintr-o competiție paralelă şi concurentă a entităților din care este format, fiind determinată de starea curentă şi de structura comportamentului obiectelor. Comportamentul obiectelor este format dintr-o secvență de acțiuni, executate de către obiect sau de un obiect delegat, în fiecare stare a traiectoriei. Acțiunile operează asupra intrărilor, generând elemente de ieşire. Elementele asupra cărora se acționează sunt: atribute obiect, comportament obiect, valoare variabilă, poziție traiectorie, acțiune asociată unei poziții.

5.3. Obiecte grafice

5.3.1. Obiecte simple şi agregate

Obiectele unui model grafic sunt entitățile asupra cărora operează comenzile aplicației. Aceste obiecte pot fi obiecte simple sau obiecte complexe, numite *agregate*.

Obiectele sunt caracterizate prin structură, caracteristici şi comportament.

Elementele grafice fundamentale sunt primitivele grafice, existente în majoritatea sistemelor grafice: punct, linie, polilinie, cerc, elipsa, primitive de bitmap (şablon, arie de celule etc.), primitivă generalizată etc. Componentele unui obiect

grafic sunt primitivele grafice. Atributele obiectelor din model caracterizează aspectul, forma, dimensiunile obiectelor. Principalul tip de atribut este atributul grafic. Atributele grafice, la rândul lor, pot fi: identificare, aspect, forma, poziție. Atributele grafice precizează caracteristicile grafice legate de contextul de prezentare a scenei de obiecte:

- atributele "*identificare*" definesc numele obiectului sau altă entitate utilizată pentru identificare;
- atributele "*aspect*" definesc prezentarea grafică a obiectului în contextul de afişare. Astfel de atribute sunt: culoarea, tipul liniei de trasare, modelul de umplere a unui poligon, grosimea liniei, fontul textului, tipul conturului etc.;
- atributele "*formă*" definesc dimensiunea şi orientarea unui obiect în contextul de afişare; atributele "*dimensiune*" pot fi: lățimea, înălțimea, factorul de scalare relativ la alt obiect sau absolut; atribute "*orientare*" pot fi: unghiul de rotație față de un alt obiect agregat, rotația față de un sistem;
- atributele "*poziție*" definesc poziția obiectului: poziția absolută față de un sistem de referință al scenei, poziția relativă în cadrul unui agregat etc.

5.3.2. Comportamentul obiectelor

Comportamentul defineşte evoluția spațială şi temporală abstractă a obiectelor dintr-o scenă de obiecte. Asocierea unui comportament la un obiect din scena de obiecte concretizează comportamentul la un obiect, poziție şi evoluție. Fiecare obiect, simplu sau complex, din cadrul unui agregat, are o evoluție dependentă de propria sa definiție a comportamentului, dar şi de evoluția agregatului din care face parte.

Un obiect instanțiat moşteneşte comportamentul obiectului prototip. Dacă se doreşte modificarea comportamentului, se creează un comportament specific prin instanțierea unui comportament prototip şi se asociază noul comportament la obiectul considerat. La activarea obiectului, acesta va evolua pe traiectoria comportamentului asociat, conform regulilor definite pentru comportamentul abstract, dar concretizate la obiectul şi scena curentă.

Activarea unui obiect simplu pe traiectorie poate determina modificarea atributelor, componentei sau chiar a comportamentului său. De asemenea, poate contribui la modificarea altor obiecte din scena de obiecte. Modificările obiectului asociat sau ale comportamentului său, sunt implicite, se precizează numai atributele sau componentele sale implicate.

De exemplu, dacă la un moment dat se comandă modificarea atributului grafic culoare, acesta se referă la *"culoarea*" obiectului asociat comportamentului.

Algoritmul de activare a unui obiect pe traiectorie este următorul:

- *se determină poziția de start a traiectoriei comportamentului; poziția inițială este aceeaşi cu poziția curentă a obiectului;*
- *se instantiază elementele traiectoriei la obiectul asociat;*
- *se actualizează şi prezintă obiectul în poziția curentă, pentru toate pozițiile traiectoriei;*
- *algoritmul de activare evaluează condițiile şi execută acțiunile specifice, corespunzătoare condițiilor îndeplinite, actualizează şi prezintă scena de obiecte în starea curentă, determină poziția următoare a obiectului pe traiectorie, actualizează poziția curentă.*

Un obiect agregat are asociat un comportament asemenea unui obiect simplu. La activarea unui obiect agregat, poziția inițială a comportamentului asociat se determină funcție de poziția curentă a agregatului. De asemenea, poziția obiectelor componente este funcție de poziția curentă a agregatului. Pe parcursul evoluției,

fiecare obiect din componența agregatului, care are un comportament specific asociat, va evolua conform comportamentului asociat lui.

Luând în considerare un agregat şi un obiect din componența sa, există următoarele alternative:

a) agregat şi obiect fără comportamente asociate (la activare vor avea o evoluție nulă);
b) agregat cu comportament, obiect fără comportament asociat (la activare agregatul va evolua conform comportamentului asociat); poziția absolută a obiectului se determină relativ la poziția curentă a agregatului pe traiectorie;
c) agregat fără comportament asociat, obiect cu comportament asociat (la activarea agregatului acesta are o comportare nulă, iar obiectul din componența sa va evolua conform comportamentului său specific);
d) agregat şi obiect cu comportamente specifice asociate (la activare se determină poziția inițială pentru agregat şi obiect; în continuare, fiecare evoluează conform comportamentului specific); în general, orice condiție de oprire sau temporizare referitoare la agregat, determină evoluția obiectelor componente; de exemplu, o oprire a agregatului determină oprirea obiectelor componente, iar abandonarea evoluției agregatului, determină abandonarea evoluției pentru toate obiectele componente; orice parametru, nespecificat explicit la comportamentul unui obiect component, se moşteneşte la execuție de către comportamentul obiectului de la comportamentul agregatului (viteza de deplasare, temporizarea în pozițiile traiectoriei; aceste temporizări pot corespunde în experimentele de simulare cuantelor de timp ale paşilor de simulare).

5.3.3. Traiectorii şi comportamente

Traiectoria defineşte pozițiile unui obiect pe parcursul evoluției specifice de către comportamentul asociat. Între o poziție curentă şi o poziție următoare, obiectul va avea o evoluție în poziții intermediare, obținute prin interpolare.

În fiecare poziție, se analizează anumite condiții şi, corespunzător descrierii evoluției, se execută anumite acțiuni. Utilizarea traiectoriei în programarea vizuală are următoarele avantaje:

- permite o tehnică de lucru naturală în abordarea programării vizuale, în care obiectele, acțiunile şi condițiile grafice (aspect şi formă) sunt preponderente față de structurile abstracte fără prezentare vizuală;
- simplifică editarea prin operare directă a noțiunilor abstracte, cum sunt: comportament abstract, prototip şi instanțiere, relații între obiecte, constrângeri etc;
- simplifică editarea condițiilor care determină acțiuni:
  - permite vizualizarea pozițiilor şi condițiilor viitoare,
  - permite construirea expresiilor grafice,
  - permite construirea condiționărilor legate de poziție, formă şi aspect;
- măreşte viteza de evaluare şi determinare a evoluției: sunt verificate numai condițiile şi acțiunile legate de poziția curentă;
- permite definirea comportamentului prin demonstrație şi înregistrarea, codificarea informațiilor introduse.

Se vor detalia câteva tipuri de traiectorie cunoscute în proiectarea aplicațiilor de realitate virtuală:traiectorie moştenită de la agregatul părinte; punct; polilinie; ciclică; cu regulă implicită; graf orientat.

Traiectoria moştenită. Un obiect cuprins într-un agregat poate să nu aibă definită o traiectorie sau chiar un comportament. În acest caz, traiectoria este moştenită de la nodul părinte din arborele de agregare. Poziția curentă a obiectului se determină

relativ la poziția agregatului, care poate avea o evoluție pe o traiectorie sau, la rândul său să o moştenească. Modificările atributelor agregat sunt moştenite dinamic de către obiectele componente. Acțiunile din cadrul evoluției agregat se referă direct la agregat, dar au efect indirect asupra obiectelor componente. Acest tip de traiectorie poate fi utilizat pentru modelarea evoluției spațiale a unei mulțimi de obiecte care au evoluție paralelă.

Traiectoria punct. Cu acest tip de traiectorie poate fi modelată evoluția unui element de circuit, care în timp îşi modifică starea grafică, forma sau alte atribute. Evoluția temporală poate fi transpusă spațial printr-o corespondentă biunivocă, între momentele de timp şi poziție. Această transpunere permite controlul evoluției temporale, atât la definirea prin operare directă cât şi la execuție.

Traiectoria polilinie. În acest tip de traiectorie, pozițiile succesive sunt date de un set de puncte P_0 , P_1, ... P_n. Coordonatele unei poziții P_{i+1} sunt definite relativ la poziția P_i. La asocierea comportamentului la un obiect, poziția P_0 a traiectoriei va lua valoarea poziției curente a obiectului asociat. Acest tip de traiectorie poate modela comportamentul unui obiect într-un proces de simulare, cum ar fi, deplasarea unui obiect virtual pe o traiectorie sau deplasarea brațului unui robot.

Traiectoria ciclică. În cazul în care o traiectorie polilinie se parcurge repetat, reprezentând un itinerar poligonal, se defineşte tipul de traiectorie ciclică. Condiția de oprire este specificată de regulile de evoluție pe traiectorie. Condițiile de parcurgere sunt specificate de parametri comportamentului: viteză, staționări etc. Tipul de traiectorie ciclică poate modela comportamentele periodice cum ar fi mişcarea unui pendul, succesiunea de acționare a unor comutatoare.

Traiectorii cu regulă implicită. Un alt tip de traiectorii sunt cele în care se defineşte primul punct şi o regulă implicită de determinare a poziției următoare. Cele mai simple traiectorii de acest tip, sunt traiectoria inerțială şi traiectoria aleatoare.

a) Traiectoria inerțială. Spre deosebire de tipurile de traiectorie prezentate până acum, la care sunt precizate explicit toate pozițiile, traiectoria inerțială defineşte poziții care aparțin unei mişcări inerțiale din fizică. Punctul de start coincide cu poziția inițială a obiectului înainte de activare. Conform mişcării inerțiale, obiectul îşi continuă mişcarea rectilinie şi uniformă, cât timp asupra lui nu acționează nici o forță, deci o regulă de modificare definită de un comportament. Acest tip de traiectorie poate modela majoritatea mişcărilor din natură. De asemenea, se pot modela evoluțiile cu traiectorii simple liniare, cum ar fi deplasările, alinierile, rearanjările de obiecte etc.

b) Traiectoria aleatoare. Asemenea traiectoriei inițiale, poziția următoare se determină după o regulă implicită, în acest caz printr-un deplasament aleator. Componenta deplasare este formată din două elemente şi anume, domeniul pentru generatorul de numere aleatoare, care specifică deplasarea pe axa x, şi respectiv domeniul pentru generatorul deplasării pe axa y, față de poziția curentă.

Traiectoria de tip orientat. Tipul de traiectorie graf orientat defineşte explicit pentru fiecare poziție curentă un set de poziții următoare. Poziția următoare a obiectului se va determina funcție de poziția curentă şi de anumite condiții din scena de obiecte. Condițiile sunt evaluate pe parcursul evoluției comportamentului. Traiectoria de tip graf încearcă să transfere o parte din logica scenariului grafic, într-un comportament încapsulat la nivel de obiect. Prin aceasta, se obține o mai bună specializare a comportamentului obiectelor, în avantajul simplificării programului generat prin programare vizuală. Specializarea comportamentului poate fi realizată pentru anumite clase de obiecte dintr-un domeniu de aplicații. De exemplu, prin acest

tip de comportament se pot construi comenzile inteligente, în care comportamentul comenzii depinde de obiectul operat, de alte obiecte din scenă sau de acțiunile precedente ale utilizatorului.

5.3.4. Modelarea evoluției în cadrul comportamentului

Evoluția definește acțiunile realizate în anumite poziții ale obiectului pe traiectorie, dacă se îndeplinesc anumite condiții. Acțiunile sunt operații realizate automat de către program asupra modelului grafic. Condițiile se referă la starea modelului sau a prezentării sale pentru afișare. Conceptual, în sistemele grafice sunt posibile următoarele modelări ale comportării unui obiect: cod program, sub forma unor proceduri fixe sau parametrizate (orice modificare de comportament necesită modificarea codului) și reguli sub formă cauzală: fapte și acțiuni.

Regulile pentru condiționare pot fi: reguli fixe; reguli interactive (prin cadre, tabele etc.); operare directă prin demonstrație cu exemple sau prin exemple și modelare evolutivă, perfecționabilă prin învățare (o posibilitate de modelare este prin rețele neuronale sau fuzzy). Modelarea prin cod program nu realizează separarea interfeței de aplicație și nu asigură independența dialogului. Regulile sub formă cauzală, din baza de cunoștințe, pot fi definite interactiv de către dezvoltatorul interfeței grafice utilizator sau în programarea vizuală, de către dezvoltatorul aplicației.

Indicatorii statici sau dinamici pot avea sau nu prezentare grafică. Indicatorii sunt entități cu valori booleene (adevărat, fals). Valoarea unui indicator se obține prin evaluarea statică sau dinamică a unei expresii. Variabilele statice sau dinamice iau valori, în domeniul valorilor posibile, pentru anumite entități model: atribute obiecte, parametri comportament. Indicatorii sau variabilele statice sunt evaluate la momentul precizat, iar indicatorii și variabilele dinamice sunt evaluate dinamic în fiecare moment al execuției.

*Structura conceptuală a acțiunilor.*Acțiunile dintr-un model reprezintă operațiile determinate de îndeplinirea anumitor condiții date, pe parcursul evoluției unui obiect în cadrul unui comportament.

Definirea formală a structurii conceptuale a unei acțiuni este următoarea:

acțiune (ACȚIUNE
entități_intrare
entități_ieșire.
operator
)

Entitățile_ieșire reprezintă elementele din model, care vor rezulta prin aplicarea operatorului asupra elementelor *entități_intrare.*

Entități ieșire pot fi următoarele: elementele unui obiect sau agregat (obiectele componente; atribute grafice sau aplicație; comportamentul atașat obiectului), elementele unui comportament (traiectoria și elementele traiectoriei; parametrii), entități program (indicatori statici sau dinamici; variabile statice sau dinamice: culoarea unui pixel; culorile admise pentru dispozitivul grafic din configurația calculatorului; valorile logice „true” și „false”; valoarea „null”).

Setul de operatori este, în general, același cu setul de operatori disponibili la operarea directă, la editarea unui scenariu grafic, prototipizare sau o parte din operatorii din programarea vizuală. Aceste informații sunt mult mai ușor de completat în programarea vizuală, decât în programarea convențională, lingvistică. De exemplu, în programarea convențională, specificarea unui obiect trebuie realizată asemenea

specificării unui fişier, prin precizarea numelui şi a căii de subdirectoare în care se află. În operarea directă se obține acces imediat la obiect, iar completarea informațiilor referitoare la el se face automat. Construirea acțiunilor în operarea directă se realizează prin selectarea obiectelor, agregatelor, comportamentelor, atributelor etc.

Într-o implementare, formatul entităților depinde de particularitățile limbajului şi de structurile de date abstracte din limbaj.

Câteva tipuri de operatori pentru comportament sunt: creare; ştergere; atribuire; adăugare; instanțiere; rotație; translație; scalare; activare; dezactivare.

5.4. Modelul obiect

5.4.1. Managementul proiectării orientate obiect

Modelul obiect are la bază principiile abstractizării, încapsulării, modularității, ierarhizării, clasificării, supradefinirii şi persistenței. Esențială este combinarea lor în modelul-obiect, deoarece acesta este fundamental diferit de modelele structurate şi cere un mod diferit de abordare a principiului de abstractizare.

Mulți programatori au fost formați să lucreze pe principiile de reprezentare structurată şi au dezvoltat nenumărate sisteme software complexe folosind aceste tehnici. Atingerea complexității folosind numai descompunerea algoritmică, cunoaşte numeroase limitări, fapt care impune descompunerea orientată pe obiecte. Programarea orientată pe obiecte, ***OOP*** este o metodă de implementare în care programele sunt organizate în colecții de obiecte ce colaborează între ele, fiecare dintre ele reprezentând o instanță a unei clase şi a căror clase sunt toate membre ale unei ierarhii de clase prin relația de moştenire. Se defineşte un limbaj orientat pe obiect dacă şi numai dacă acesta satisface următoarele cerințe *[4], [7], [17]*:

- suportă obiecte care sunt abstractizări ale datelor, cu o interfață de operații definite şi cu proprietatea de ascundere locală;
- obiectele au un tip (clasă) asociat(ă);
- clasele pot moşteni atribute de la superclase.

Reprezentarea orientată pe obiecte, este o metodă ce întruchipează procesul de descompunere orientată pe obiecte recomandată pentru reprezentarea modelelor logice şi fizice, atât în mod static, cât şi dinamic. Fiecare stil de programare este bazat pe propriul său model conceptual. Alegerea setului corect de abstractizări pentru un domeniu dat, este problema centrală în reprezentarea orientată pe obiecte. Există un întreg spectru de abstractizări, de la obiecte care modelează entitățile domeniului problemei, până la obiecte ce nu au nici o utilizare reală. Aceste tipuri de abstractizări includ următoarele concepte:

- *abstractizarea entității* – un obiect ce reprezintă un model util al unei entități din domeniul problemei;
- *abstractizarea acțiunii* – un obiect care prevede un set generalizat de operații, fiecare dintre ele furnizând acelaşi tip de funcție;
- *abstractizarea maşinii virtuale* – un obiect care grupează împreună operații ce sunt toate folosite de un nivel superior de control sau operații care folosesc toate un nivel inferior care cuprinde un set complet;
- *abstractizarea întâmplătoare* – un obiect care grupează un set de operații ce nu au nici o legătură una cu alta.

S-a constatat că principiile abstractizării şi încapsulării sunt concepte complementare. Abstractizarea se axează pe vederea din afară a unui obiect şi încapsularea împiedică accesul la vederea interioară, unde comportamentul

abstractizării este implementat. În acest fel, încapsularea prevede bariere clare între diferite abstractizări. Se sugerează că "pentru ca abstractizarea să funcţioneze, implementările trebuie să fie încapsulate." *[7]*. În practică, aceasta înseamnă că fiecare clasă trebuie să aibă două părţi: o interfaţă şi o implementare. În esenţă, se defineşte încapsularea ca fiind procesul ce oferă toate detaliile asupra unui obiect care nu contribuie la caracteristicile sale esenţiale.

Un alt principiu al modelului obiect este ierarhia, care interesează problema abstractizării pentru că adesea un set de abstractizări formează o ierarhie. Ierarhia se defineşte şi ca fiind aranjarea abstractizărilor. Două din cele mai importante ierarhii într-un sistem complex sunt: structura de clasă (felul ierarhiei) şi structura obiectului (partea ierarhiei). Moştenirea defineşte o relaţie între clase, acolo unde o clasă împarte structura sau comportamentul definit în una sau mai multe clase, numite moştenire simplă şi respectiv moştenire multiplă.

Modelul obiect se dovedeşte aplicabil la un număr mare de sisteme de ***RV***, acest model având soluţii pentru a administra complexitatea specifică sistemelor complexe.

5.4.2. Relaţii între obiecte grafice

Am definit un obiect grafic ca o entitate tangibilă ce manifestă un comportament bine definit. Din perspectiva cunoaşterii umane, un obiect este un lucru tangibil sau vizibil, ceva ce poate fi conceput intelectual. Un obiect modelează o parte a realităţii şi de aceea este ceva ce există în timp şi spaţiu. Un obiect reprezintă o unitate individuală, identificabilă sau o entitate, fie ea reală sau abstractă, cu un rol bine definit în domeniul problemei. În termeni şi mai generali, un obiect este orice are graniţe strict definite. Dar deşi este util un obiect ce are graniţe strict definite, acest lucru nu este suficient pentru a distinge un obiect de altul. De aceea experienţa sugerează următoarea definire a termenului obiect: un obiect are stare, comportament şi identitate; structura şi comportamentul obiectelor similare sunt definite în clasa lor comună; termenii „instanţă" şi „obiect" nu pot fi schimbaţi între ei.

Starea unui obiect respectă toate proprietăţile unui obiect (de obicei statice) şi valorile curente (de obicei dinamice) ale fiecăreia din proprietăţi. Fiecare proprietate are o anumită valoare. Această valoare poate fi o simplă cantitate sau poate desemna un alt obiect. Faptul că fiecare obiect are stare, implică faptul că fiecare obiect ocupă un loc în spaţiu, în lumea fizică sau în lumea virtuală.

Nici un obiect nu este izolat. Obiectele suportă acţiuni şi la rândul lor, ele acţionează asupra altor obiecte. Se defineşte *comportamentul* unui obiect astfel: comportamentul este felul în care un obiect acţionează şi reacţionează în termenii schimbărilor stării sale. Cu alte cuvinte comportamentul unui obiect este complet definit de acţiunile sale.

Identitatea este acea proprietate a unui obiect care îl distinge de alte obiecte. Multe limbaje de programare şi baze de date folosesc nume variabile pentru a distinge obiectele temporare, îmbinând posibilitatea de adresare şi identificare. Greşeala de necunoaştere a diferenţei dintre numele unui obiect şi obiectul în sine, este sursa a multiple tipuri de erori în programarea orientată pe obiecte. Relaţia dintre două obiecte respectă presupunerile că fiecare acţionează asupra celuilalt, incluzând şi operaţiile ce pot fi îndeplinite şi comportamentul ce rezultă. Două tipuri de ierarhii ale obiectului prezintă un interes particular în reprezentarea orientată pe obiecte şi anume: relaţii de folosire şi relaţii de limitare.

Relaţia de folosire are la bază principala caracteristică şi anume, aceea de a transfera mesaje între două obiecte. De obicei transferul de mesaje între două obiecte

este unidirecțional cu toate că şi comunicarea bidirecțională este posibilă. Având o colecție de obiecte implicate în relațiile de folosire, fiecare dintre obiecte poate avea unul din următoarele trei roluri:

- actor – un obiect ce poate opera asupra altor obiecte, dar care nu poate fi supus niciodată acțiunii altor obiecte;
- server – un obiect care nu operează niciodată asupra altor obiecte; el este întotdeauna apelat de alte obiecte;
- agent – un obiect ce poate opera asupra unor obiecte şi poate fi apelat de altele.

Limitarea unui obiect este uneori mai bună decât folosirea pentru că limitarea reduce numărul de obiecte ce trebuie să fie vizibile la nivelul obiectului închis. Pe de altă parte, folosirea este uneori mai bună decât limitarea, deoarece conduce la o cuplare mai strânsă între obiecte.

Referitor la imaginea unei clase privită din exterior şi din interior, mulți cercetători sunt de acord că programarea este în multe situații o chestiune de "comprimare": funcțiile variate ale unei probleme extinse sunt descompuse în probleme mai mici prin subcomprimarea lor în elemente diferite ale reprezentării. Nicăieri nu este mai evidentă această idee decât în reprezentarea claselor. Oriunde un obiect individual este o entitate concretă ce îndeplineşte un rol în întregul sistem, clasa cuprinde structura şi comportamentul comun tuturor obiectelor înrudite cu el. O clasă este ca un fel de contract de legătură între o abstractizare şi toți clienții săi. Un limbaj puternic de programare poate detecta erorile acestui contract în timpul compilării.

Acest mod de abordare permite detectarea diferenței între imaginea exterioară şi cea interioară a unei clase. Interfața unei clase furnizează imaginea sa externă şi de aceea scoate în evidență abstractizarea în timp ce ascunde structura sa şi secretele comportamentului ei. Prin contrast, implementarea unei clase este imaginea sa interioară, ce nu ascunde secretele comportamentului ei. Implementarea unei clase constă în implementarea tuturor operațiilor definite în interfața unei clase.

Se folosesc trei tipuri de relații între clase: prima este generalizarea ce arată tipul de relație, a doua este reunirea ce arată partea relației şi a treia este asociația care arată o conectare semantică între clase neînrudite. Câteva abordări în acest sens au transformat limbaje de programare spre a exprima generalizarea, reunirea şi asociația. În mod special, limbajele orientate pe obiecte suportă combinația următoarelor relații între clase:

- relații de moştenire;
- relații de folosire;
- relații momentane;
- relații de metaclasă.

Moştenirea este cea mai puternică dintre aceste relații şi poate fi folosită pentru a exprima atât generalizarea, cât şi asociația. Din experiență, am constatat că moştenirea este un mijloc necesar, dar nu suficient pentru a exprima gama largă de relații ce poate exista în abstractizările cheie într-un domeniu al problemei dat. Este nevoie şi de *relațiile de folosire* care suportă reunirea. Mai mult, sunt necesare şi *relațiile momentane*, care, ca şi moştenirea, suportă generalizarea şi asociația, deşi într-un mod cu totul diferit. *Relațiile de metaclasă* nu sunt explicit suportate de orice limbaj de programare orientat pe obiecte. În esență, o metaclasă este clasa unei subclase şi care astfel permite să se trateze clasele ca obiecte.

Moştenirea este o relație între clase, acolo unde o clasă împarte structura sau

comportamentul definit cu una (moştenire simplă) sau mai multe (moştenire multiplă) clase. Se numeşte clasa de la care o altă clasă moşteneşte, superclasă. O clasă dată are tipic două tipuri de clienți tipici: instanțe şi subclase. Este clar că moştenirea înseamnă şi faptul că subclasele moştenesc structura superclaselor lor. Există o foarte reală tensiune între moştenire şi încapsulare pentru că folosirea moştenirii expune nişte informații secrete ale unei clase moştenite. Practic aceasta înseamnă că pentru a înțelege importanța unei clase particulare trebuie studiate adesea toate superclasele sale, ceea ce include uneori şi imaginile lor interioare.

Cele trei tipuri de relații între clase (moştenire, folosire şi momentană) acoperă împreună toate tipurile importante de relații între clase de care cei mai mulți dintre cercetători vor avea vreodată nevoie.În timpul analizei şi studiului reprezentării, proiectantul are două sarcini de bază:

- să identifice clasele şi obiectele din vocabularul domeniului problemei;
- să creeze structurile unde seturile de obiecte lucrează împreună pentru a furniza comportamentele ce satisfac cerințele problemei.

Asemenea clase şi obiecte se numesc abstractizările cheie ale problemei şi structurile cooperante se numesc mecanisme ale implementării.

Pentru aproape toate aplicațiile clasele sunt statice, de aceea, relațiile, semanticile şi existența lor sunt fixe pentru execuția unui program. În mod similar, clasa a mai multe obiecte este statică, însemnând că odată ce un obiect este creat, clasa este fixă, deşi obiectele sunt create tipic şi distruse repede de-a lungul exploatării unei aplicații de ***RV***. Se poate şti dacă o clasă dată sau un obiect sunt bine reprezentate apelând la câteva metode de evaluare:

- cuplarea;
- coeziunea;
- suficiența;
- completivitatea;
- primitivitatea.

Cuplarea este o noțiune împrumutată din reprezentarea structurată dar, printr-o interpretare liberală, poate fi aplicată şi în reprezentarea orientată pe obiecte. Ea este definită ca măsura puterii unei asociații stabilită printr-o conexiune de la un modul la altul. Şi ideea de *coeziune* vine tot din reprezentarea structurată. Ea măsoară gradul de conexiune între elementele unui singur modul (şi pentru reprezentarea orientată pe obiecte, pentru o singură clasă sau obiect). Destul de sugestive sunt şi criteriile că o clasă sau un modul ar trebui să fie suficiente, complete şi primitive. Prin suficient se înțelege că modulul sau clasa cuprinde suficiente caracteristici ale abstractizării pentru a permite o interacțiune deplină şi eficientă. Prin complet, se înțelege că interfața unei clase sau modul cuprinde toate caracteristicile importante ale abstractizării. Conceptul de primitivitate, indică faptul că operațiile primitive sunt acele operații ce pot fi implementate eficient numai dacă au acces la reprezentarea fundamentală a abstractizării.

Identificarea claselor şi obiectelor este cea mai importantă parte a reprezentării orientate pe obiecte. Identificarea cere în acelaşi timp descoperire şi invenție. Prin descoperire se ajunge să se recunoască abstractizările cheie şi mecanismele din domeniul problemei ce trebuie rezolvată. Prin invenție, se imaginează abstractizări generalizate şi noi mecanisme ce reglează modul în care obiectele ar trebui să colaboreze. Descoperirea şi invenția sunt fundamentale în găsirea asemănărilor şi ajută să se identifice generalizarea, specializarea şi reunirea ierarhiilor dintre clase şi

ghidează luarea deciziilor asupra modularizării. Se vor plasa anumite procese împreună în acelaşi procesor sau în procesoare diferite depinzând de felul în care aceste procese sunt înrudite funcţional.

Clasificarea este o muncă grea pentru că este un proces de incrementare şi iterativ. Natura sa incrementală şi iterativă este evidentă în dezvoltarea şi în construcţia ierarhiilor de clase şi obiecte pentru reprezentarea unui sistem software complex. Uneori, unele clasificări sunt mai bune decât altele, neexistând termenul de clasificare ideală sau perfectă. Aceasta pentru că o clasificare inteligentă cere din partea cercetătorului o intensă muncă creativă, de pildă să găsească asemănarea între obiecte ce nu sunt înrudite.

Se cunosc trei abordări ale clasificării:

- împărţirea clasică pe categorii;
- gruparea conceptuală;
- teoria prototipului.

Abordarea clasică a clasificării se bazează pe ideea că toate entităţile ce au o proprietate dată sau o colecţie de proprietăţi în comun formează o categorie. Aceste proprietăţi sunt necesare şi suficiente pentru a defini o categorie. Sintetizând, abordarea clasică foloseşte proprietăţi înrudite drept criteriu pentru asemănarea dintre obiecte. Gruparea conceptuală este o variantă mai modernă a abordării clasice şi, în mare, derivă din încercările de a explica cum este reprezentată cunoaşterea. În această abordare, clasele (grupări de entităţi) sunt generate prin formularea mai întâi a descrierilor conceptuale ale acestor clase şi apoi clasificarea entităţilor corespunzătoare descrierilor. Astfel, gruparea conceptuală se manifestă mai mult ca o grupare probabilistică a obiectelor. Cele două tipuri de abordări ale clasificării sunt suficient de expresive pentru a reprezenta un sistem software complex. Dar se întâlnesc situaţii în care aceste abordări nu sunt adecvate şi de aceea se ajunge la cea mai recentă abordare a clasificării, numită teoria prototipului, care introduce următoarea stipulaţie: o clasă de obiecte este reprezentată de un obiect prototip şi un obiect este considerat a fi membru al acestei clase dacă şi numai dacă se aseamănă cu acest prototip în aspecte importante.

În analiza unui domeniu trebuie respectaţi următorii paşi:

- construirea unui model generic de tip schiţă a domeniului prin consultarea cu experţii domeniului;
- examinarea sistemelor existente în interiorul domeniului şi reprezentarea celor deduse într-o formă comună;
- identificarea asemănărilor şi diferenţelor dintre sisteme prin consultarea experţilor domeniului;
- rafinarea modelului generic pentru a se asemăna cu sistemele existente.

Analiza domeniului poate fi folosită pentru aplicaţii de analiză verticală a domeniului ca şi pentru părţi înrudite ale aceleiaşi aplicaţii (analiză orizontală a domeniului). Clasele şi obiectele ce rezultă reflectă un set de abstractizări cheie şi mecanisme generalizate pentru rezolvarea unui set întreg de acţiuni. Reprezentarea ce rezultă este mai simplă dacă fiecare relatare a fost analizată şi descrisă separat. Analiza domeniului este rareori o activitate monolitică; este mult mai util dacă se analizează câte puţin şi apoi se reprezintă câte puţin. O abstractizare cheie este o clasă sau un obiect ce face parte din vocabularul domeniului problemei. Identificarea abstractizărilor cheie este problema cea mai grea a unui domeniu specific. Alegerea cea mai bună a obiectelor depinde, fireşte, de scopurile urmărite de o aplicaţie şi de

structura granulată a informației ce urmează a fi manipulată.

O dată ce s-a identificat o anumită abstractizare cheie, ea trebuie evaluată în acord cu metrica descrisă anterior. Având o abstractizare nouă, aceasta trebuie să fie plasată în contextul ierarhiei de clasă şi obiect existente. Uneori se poate găsi o subclasă generală şi se transferă problema analizată în structura de clasă, ceea ce duce la creşterea gradului de împărţire. Similar se poate găsi o clasă ca fiind prea generală, ceea ce face dificilă moştenirea datorită unei mari deosebiri semantice. În fiecare caz, trebuie identificate abstractizări cuplate pentru a depăşi aceste situații.

În reprezentarea orientată pe obiecte se reprezintă abstractizări cheie pentru a forma un model al realității. Abia apoi trebuie adăugat comportamentul acestor abstractizări care derivă din comportamentele observabile ale sistemului. Se foloseşte termenul de mecanism pentru a descrie orice structură în care obiectele lucrează împreună pentru a furniza un comportament care să satisfacă cerința problemei. Oriunde reprezentarea unei clase are cunoştință de modul individual în care se comportă un obiect, un mecanism este o decizie de reprezentare. Aceasta conține informații şi despre modul în care o colecție de obiecte cooperează. De asemenea, oriunde abstractizările cheie reflectă vocabularul domeniului problemei, mecanismele sunt esența reprezentării.

La fel ca şi clasele, obiectele şi mesajele dintr-o diagramă de obiect au forme ce pot furniza informații detaliate.

O diagramă de modul este folosită pentru a arăta distribuirea de clase şi module în reprezentarea fizică a unui sistem. O diagramă de modul singulară reprezintă un întreg sau o parte a arhitecturii de modul a sistemului. Cele mai importante elemente ale unei arhitecturi de modul sunt modulele şi vizibilitatea modulului.

Interesează modul prin care se decide organizarea execuției proceselor din interiorul unui procesor. Sunt cinci abordări generale de organizare:

- preemtiv (procesele cu o prioritate mai mare şi care sunt pregătite de execuție întrerup procesele cu o prioritate mai mică ce se află deja în execuție; în cazul proceselor cu aceeaşi prioritate, fiecăruia i se alocă aceeaşi cuantă de timp);
- nonpreemptiv (procesul aflat în desfăşurare continuă să lucreze până ce cedează controlul);
- ciclic (controlul trece de la un proces la altul; fiecărui proces îi este alocat o cuantă de timp);
- executiv (execuția proceselor este controlată algoritmic);
- manual (procesele sunt organizate de către un utilizator din afara sistemului).

6. ARHITECTURI DE BAZE DE DATE DISTRIBUITE

6.1. Orientări generale privind utilizarea bazelor de date distribuite

În acest capitol se definesc teoretic arhitecturi experimentale ale bazelor de date de mari dimensiuni, de tip distribuit. Acestea sunt definite ca fiind structuri informatice care permit culegerea, prelucrarea, analiza şi difuzarea unor colecții de date heterogene. Se definesc şi ca fiind colecții de mai multe baze de date corelate logic, fiecare fiind asociată unui nod al unei rețele şi unui mecanism de acces, care face această colecție transparentă pentru utilizator.

Se prezintă în continuare cele mai importante noțiuni legate de proiectarea bazelor de date distribuite, de mari dimensiuni. Sistemele de baze de date distribuite, de mari dimensiuni se definesc în raport cu următoarele seturi de obiective:

- cele care privesc bazele de date, menite a controla şi utiliza efectiv resursele informaționale asociate, prin disponibilitatea şi integritatea datelor;
- cele care privesc sistemul de comunicații asociat, menit să reducă numărul şi dimensiunea mesajelor, precum şi traseul şi timpul pe parcursul transmisiei.

Pentru proiectarea unui sistem de baze de date pentru aplicații de ***RV*** trebuiesc avute în vedere următoarele aspecte:

1. există baze de date centralizate, care se află într-un singur punct al sistemului;
2. există baze de date replicate sau multiplicate, care se află în două sau mai multe puncte asociate nodurilor sistemului;
3. există baze de date partiționate, în care datele sunt organizate în segmente logice separate, care se află în două sau mai multe puncte fizice / logice ale sistemului.

Independent de tipul de bază de date şi de arhitectura ei, sistemul reclamă cunoaşterea locațiilor înregistrărilor fizice ale datelor. În sistemele de ***RV*** , lista cât şi baza de date se recomandă să se afle în acelaşi punct. Aceste elemente reprezintă factori critici de proiectare, de ele depinzând o serie de caracteristici ale sistemului.

Problema proiectării bazelor de date distribuite, respectiv plasarea bazelor de date şi aplicațiilor în nodurile sistemului, apelează la fragmentarea bazelor de date, adică descompunerea relațiilor inițiale în subrelații şi apoi alocarea acestor fragmente unor noduri ale rețelei, conform următoarelor modele de plasare *[10]:*

a. partiționat (total nereplicat): baza de date este împărțită în partiții disjuncte (fragmente), fiecare dintre aceste partiții fiind plasată apoi în alt punct al sistemului;

b. replicat: acest model are la rândul lui două variante - total replicat, respectiv parțial replicat; în varianta "total replicat" întreaga bază de date este stocată în fiecare punct al sistemului, iar în varianta "parțial replicat" fiecare partiție a bazei de date este stocată în mai multe puncte, dar nu în toate.

S-au studiat două mecanisme: replicare dinamică, respectiv statică.

Definirea fragmentelor în baze de date distribuite se realizează conform unor reguli similare cu cele pentru definirea relațiilor virtuale în bazele de date centralizate. Datele replicate pot fi tratate în acelaşi mod. Argumentul esențial în favoarea fragmentării constă în posibilitatea execuției unor operații în paralel, pe diverse fragmente, crescând concurența. Argumentele împotriva fragmentării se referă la problemele de control semantic şi integritate a datelor.

S-au dezvoltat două strategii fundamentale de partiționare:

a. fragmentarea orizontală - partiționează o relație de-a lungul tuplelor (legăturilor) sale, fiecare fragment conținând un subset din tuplele relației.

b. fragmentarea verticală - produce proiecția relației R în partițiile R1, R2, ... Rn, fiecare conținând un subset al atributelor din R, inclusiv cheia primară.

Fragmentarea verticală a fost utilizată pe larg în cazul bazelor de date centralizate; în contextul bazelor de date distribuite, ceea ce interesează în mod deosebit este nesuprapunerea fragmentelor.

Pentru o aplicație de ***RV***, o simplă fragmentare verticală sau orizontală nu este suficientă; aplicarea ambelor variante, conduce la apariția celui de-al treilea tip de fragmentare, numită hibridă.

Al doilea pas în proiectarea bazelor de date distribuite constă în alocarea fragmentelor la nodurile rețelei. În cazul în care alocarea se realizează conform modelului partiționat, procesarea cererilor este dificilă, dar controlul concurenței este relativ uşor de realizat. Alternativa replicării totale conduce la o procesare uşoară a cererilor.

Dacă se consideră un set de fragmente F={ F1, F2, ... Fn} şi un set de puncte ale sistemului R={ R1, R2, ... Rm} pe care rulează un set de aplicații Q, problema alocării constă în determinarea distribuției optime a lui F pe R.

Un pas important pentru aflarea soluției este definirea "optimului", care poate fi considerat:

a. traseul minimal - funcția de graf reprezintă dimensiunea stocării fiecărui fragment Fi în nodul Ri, durata procesării lui Fi la nodul Ri, durata actualizării lui Fi în toate nodurile unde este stocat şi durata comunicațiilor de date.

b. performanța - strategia de alocare este proiectată astfel încât să mențină o anumită performanță (de exemplu, minimizarea timpului de răspuns, esențială pentru aplicațiile de ***RV***).

Problema alocării poate fi specificată ca o problemă de minimizare a duratei, în care se încearcă găsirea unui set I din S, care specifică unde vor fi stocate copiile fragmentelor.

În proiectarea bazelor de date distribuite pentru o aplicație de ***RV***, trebuie să se decidă următoarele:

- dacă tabelele locale reduc volumul traficului de comunicații (mesaje, tranzacții etc.);
- dacă este necesar pentru toate stațiile ca acestea să actualizeze aceleaşi înregistrări;
- dacă stațiile au nevoie să acceseze înregistrări, trebuie stabilit cât de des trebuie acestea să fie actualizate;
- dacă pot fi utilizate tabelele locale pentru a verifica datele de intrare;
- dacă înregistrările unei tabele fac parte din fragmente care să poată fi păstrate în noduri dispersate;
- responsabilul pentru păstrarea integrității înregistrărilor din bazele de date şi actualizarea acestora;
- cum este păstrată siguranța bazelor de date distribuite.

Procesarea distribuită a cererilor are scopul de a apela la o strategie corespunzătoare care minimizează timpii de comunicație, pentru o anumită interogare.

Strategia este descrisă în termenii operatorilor din algebra relațională, folosind şi primitive de comunicație aplicate bazelor de date locale.

Procesul este compus din următoarele etape:

a. Descompunerea cererilor: realizează transformarea cererilor din calcul relațional (limbaj de nivel înalt) în termenii algebrei relaționale. Deoarece, atât cererile de intrare, cât şi cele de ieşire sunt globale, fără cunoştințe despre distribuția datelor, problema se tratează similar bazelor de date centralizate:

- normalizarea - se transformă în forma conjunctivă normală sau în forma disjunctivă normală;
- analiza semantică - detectează cererile incorecte din punct de vedere semantic şi le elimină;
- eliminarea redundanței - prin aplicarea unei reguli de simplificare asupra predicatelor obținute în prima etapă, se elimină predicate inutile;

- rescriere - se rescrie cererea în termenii algebrei relaționale.

b. Localizarea datelor: se transformă cererea globală exprimată în termenii algebrei relaționale, în cereri fragmentate, aplicate fragmentelor bazei de date.

c. Optimizarea globală şi locală a cererilor: cererea rezultată din primii doi paşi poate fi executată după simpla adăugare a primitivelor de comunicație. Permutările posibile în ordinea operațiilor din cerere pot să conducă la diverse strategii de execuție. Rolul acestui pas constă în găsirea unei ordini "optimale".

Pentru implementarea unui sistem distribuit de baze de date este necesar un limbaj relațional bazat fie pe algebra relațională, fie pe calculul relațional. Pentru scrierea aplicațiilor complexe, SQL ca standard pentru ***SGBD*** nu este suficient şi de aceea, interfațarea cu un limbaj de programare procedural este de obicei necesară.

Limbajele avansate combină operatorii din algebra relațională cu arhitecturi de program; posibilitatea de a utiliza variabile temporare face ca aceste limbaje, numite "limbaje de programare orientate spre baze de date" (Oracle, Sybase, Access etc.) să fie deosebit de utile la scrierea aplicațiilor.

Dezvoltarea bazelor de date distribuite a fost puternic impulsionată de apariția stațiilor de lucru (workstation) puternice şi a calculatoarelor paralele. Un sistem cu prelucrare distribuită poate fi un sistem expert extensibil, adaptabil, fizic distribuit şi logic integrat.

Noile direcții de cercetare cuprind şi aspecte specifice inteligenței artificiale. Bazele de date relaționale distribuite vor fi înlocuite cu baze de cunoştințe distribuite şi cu baze de date distribuite orientate obiect.

Avantajele sistemelor distribuite de baze de date trebuie evaluate într-o ordine care să aibă în vedere, în afara factorilor tehnico-funcționali, în final, şi factorii de ordin economic. Aceste avantaje sunt:

- datele pot fi uşor folosite de mai mulți utilizatori;
- sistemele distribuite sunt amplasate aproape de locul de utilizare, unde se pot implementa şi funcțiuni de control local, ceea ce reprezintă o cerință majoră pentru foarte mulți utilizatori;
- distribuirea accesului la date conduce la reducerea conflictelor în asigurarea accesului şi la creşterea performanțelor de ansamblu ale sistemului de ***RV***;
- copierea datelor şi informațiilor de control şi distribuirea lor în mai multe locuri, conduce la creşterea fiabilității şi disponibilității datelor şi informațiilor la nivel de sistem.

Chiar dacă anumite părți din sistem sunt indisponibile datorită unor cauze diferite (legături de comunicații întrerupte, capacități de calcul indisponibile etc.), sistemul tot poate să ofere un minimum de servicii de acces la date şi informații.

- realizarea modulară a sistemului, în condiții de distribuire geografică, conduce automat la posibilitatea unei extinderi uşoare a acestuia din punct de vedere teritorial;
- avantaje economice pentru sistemele distribuite se pot obține numai în cazul când se concepe utilizarea datelor în raport de strictă dependență de aplicații (legături strânse între aplicație şi datele amplasate la distanță, care ridică mult costurile pentru transmisia de date).

În plus, pot să apară avantaje legate de costurile privind sistemele folosite, instalarea şi exploatarea lor. Pentru sistemele mici aceste costuri sunt mai reduse, comparativ cu sistemele mari, de tip centralizat. Dintre dezavantajele de ordin tehnic şi economic, ale utilizării bazelor de date distribuite se menționează:

- complexitatea sistemelor distribuite este un inconvenient pentru utilizatorii lipsiți de experiență;
- asigurarea protecției sau securității datelor poate fi un dezavantaj dacă sistemul nu este prevăzut cu tehnici şi metode care să asigure aceste facilități;
- migrarea datelor din BD centrala în cele distribuite poate constitui o dificultate prin faptul că nu există încă metodologii şi instrumente asociate care să facă transferul automat;
- costurile sunt un impediment, mai ales pentru comunicații, cu tendința de diminuare pe măsura dezvoltării tehnologiilor de comunicații la distanță.

Problemele de ordin tehnic pe care le ridică implementarea şi exploatarea sistemelor distribuite în aplicații de ***RV***, sunt obiective de cercetare ale căror finalizări vor conduce la soluții particularizate. Câteva dintre acestea sunt:

- proiectarea bazelor de date distribuite strict în raport cu aplicațiile;
- execuția tranzacțiilor pe baza unor algoritmi de analiză care să conducă la o succesiune de instrucțiuni de tratare a datelor apelate din BD centrala;
- realizarea unor mijloace de prezentare neconvențională a unor informații, relativ la conținutul bazelor de date;
- elaborarea unor metode de control eficient al paralelismelor, al blocajelor posibile, al actualizărilor multiple sau multiplicate, al erorilor şi reperării sau semnalării lor;
- elaborarea unor instrumente de conversie automată, în cazul utilizării unor baze de date heterogene.

Prezentând avantajele, dezavantajele şi problemele pe care le ridică utilizarea sistemelor de baze de date distribuite, se conturează noi opțiuni şi direcții de acțiune pe plan conceptual şi ca obiective de cercetare, în ceea ce priveşte utilizarea acestora.

6.2. Elemente definitorii ale unei aplicații de realitate virtuală

Conexiunile specifice aplicațiilor de realitate virtuală sunt atât de diverse şi de complexe, încât orice abordare exhaustivă ar fi sortită eşecului. Pentru a le defini, se utilizează câteva elemente, care coexistă în staturi interțesute, formând subsisteme funcționale:

- indivizi - membri care aparțin unei mulțimi;
- areale - delimitări fizice şi logice ale spațiului ocupat de diverse mulțimi;
- obiecte - factori ai mediului de viață pentru indivizi;
- evenimente - procese naturale, sociale, economice, meteorologice etc. produse în mediul de viață.

Este esențial ca pentru elementele primare ale sistemului să se definească structurile de date în mod corespunzător, care să satisfacă toate subsistemele funcționale, permițând o comunicație eficientă între ele. Subsistemele funcționale pot fi delimitate la nivel micro sau macro constructiv, fixându-le un domeniu corespunzător. De asemenea, se defineşte spațiul teritorial. Acesta nu poate surprinde în totalitate imaginea anumitor obiecte şi evenimente, deoarece le priveşte din punct de vedere strict descriptiv sau strict funcțional. Apare fenomenul de neglijare a unor descriptori esențiali pentru obiecte şi evenimente, dar nesemnificativi pentru decizia de acțiune în contextul aplicației. Spațiile teritoriale sunt stratificate şi întrețesute, de aceea se impune alegerea unei unități elementare de spațiu virtual care să permită agregarea informațiilor.

Obiectele sunt elemente naturale sau imaginare ale mediului virtual: apă, vegetaţie, clădiri, drumuri, poduri, vehicule, mobilier etc. Acestea pot fi caracterizate prin descriptori convenabili.

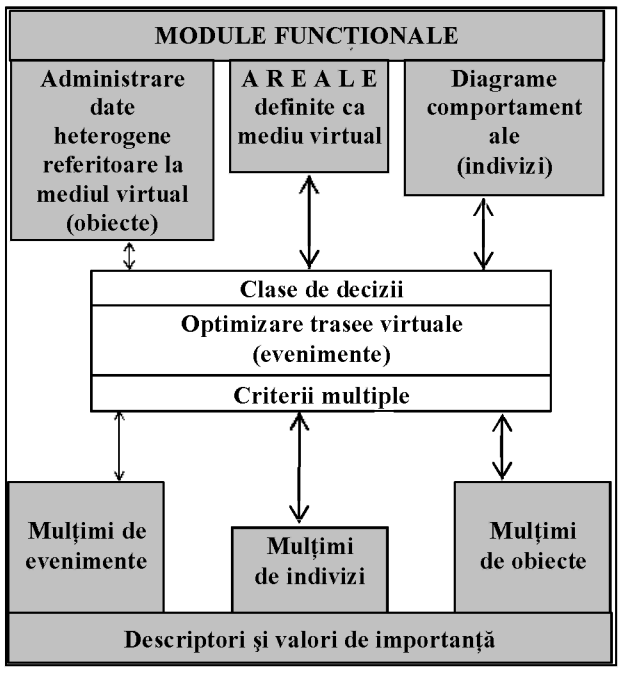

Figura 6.1. Elemente de definiţie pentru problematica aplicaţiilor de RV

Evenimentele sunt procese naturale, sociale, economice, financiare care se produc în mediul virtual. Ele pot fi periodice sau accidentale. În teoria probabilităţilor, evenimentul este o noţiune primară, cadrul structural în care se dezvoltă această teorie fiind cel al algebrei evenimentelor, definit axiomatic. A efectua o experienţă virtuală înseamnă a alege, printr-un procedeu susceptibil de a fi repetat, un element dintr-o mulţime dată. O anume realizare, consumată sau viitoare a experienţei, se numeşte *probă*. Un eveniment se consideră realizat dacă, în urma experienţei, afirmaţia conţinută în propoziţie s-a dovedit adevărată sau informaţia nu s-a realizat, dacă afirmaţia s-a dovedit falsă.

Informaţiile referitoare la personajele (indivizii), obiectele şi evenimentele sistemului de ***RV*** vor fi stocate împreună cu descriptorii lor, iar descriptorilor li se vor atribui valori de importanţă în funcţie de obiectivele urmărite. Descrierea elementelor componente ale unei aplicaţii de ***RV*** va putea fi extinsă pe măsura apariţiei unor noi criterii de decizii astfel încât, sub aspect informaţional, sistemul se va prezenta ca un sistem deschis şi evolutiv, permiţând funcţionarea unor bucle de reglaj eficiente.

6.3. Modelarea evenimentelor virtuale

O etapă importantă în proiectarea unui sistem de ***RV*** o constituie modelarea evenimentelor virtuale. Aceasta se realizează în vederea obţinerii unor funcţii de simulare, a evaluării deciziilor luate sau a execuţiei unor algoritmi de optimizare a elaborării deciziilor.

Obţinerea unui model analitic pentru un sistem de ***RV*** este deosebit de laborioasă datorită influenţelor factorilor care constituie intrări sau variabile de stare ale sistemului. Este aproape imposibil să se cuantifice influenţa tuturor factorilor, în

parte şi din absența datelor despre aceştia, datorată fie lipsei de modele şi de aparate de măsurare, fie chiar necunoaşterii naturii acestor factori. Uneori influența acestora poate fi apreciată dar rezultatele ieşirilor să se situeze în afara clasei de eroare acceptată. De aceea, se recomandă să nu se identifice care este ponderea factorilor neluați în considerare în model, ci să se grupeze observațiile pe "clase de influență" a acestor factori, elaborând pentru fiecare clasă câte un model separat. Rezultă astfel un model general.

Încercările de modelare a fenomenelor şi evenimentelor implică cooperarea interdisciplinară. Complexitatea sistemelor de realitate virtuală obligă deseori în crearea unor modele matematice prin extragerea unei serii de proprietăți caracteristice. Trebuie alese cu multă grijă ipotezele simplificatoare, rezultând uneori, din neglijarea acestui aspect, concluzii teoretice inacceptabile din punct de vedere practic.

Diversitatea şi complexitatea informațiilor care descriu mediile virtuale se caracterizează prin:

- multitudinea de surse ale datelor şi de factori de decizie logică;
- dispersia geografică a surselor de date;
- necesitatea agregării şi/sau descentralizării informațiilor pe diferite nivele şi clase de evenimente;
- dinamica mare a cererilor de informare pe diferite nivele ierarhice şi clase de evenimente şi în diverse areale geografice;
- autonomia cooperativă a surselor şi țintelor.

Aceste caracteristici impun proiectarea unor sisteme de gestiune a bazelor de date distribuite, deschise şi evolutive. Strategia abordării sistemului este prezentată în schema de referință de la *Figura 6.2.*Soluția propusă pentru tratarea problematicii evenimentelor virtuale este abordarea ierarhică, recomandată pentru sistemele mari, complexe. Întrucât nu există o definiție unanim acceptată a sistemelor de ***RV***, se preferă recunoaşterea acestora după un set de caracteristici între care:

- structura modulară interconectată;
- existența mai multor obiective;
- dimensiuni mari, restricții în structura informațională;
- prezența incertitudinii.

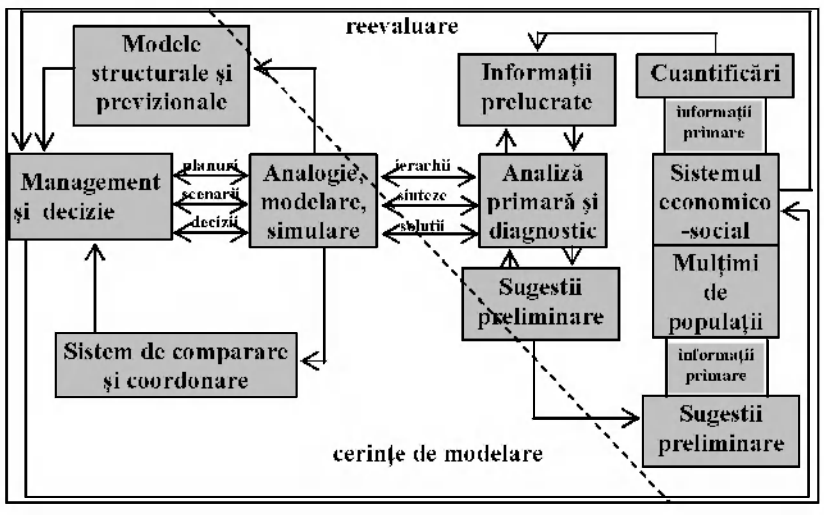

Figura 6.2. Cerințe conceptuale pentru sistemele cu aplicații deRV

În orice problemă de formalizare a mulțimilor de evenimente virtuale se poate observa o piramidă formată din centre de formulare cerințe, pentru rezolvarea unor solicitări care variază în complexitate, acestea fiind mai numeroase, dar mai simple, către bază.

Problemele se rezolvă cu o frecvență din ce în ce mai mare către baza piramidei multistrat, ținând seama de anumiți parametri de "actualizare", care sunt impuşi prin rezolvarea cererilor evolutive aflate către vârful piramidei.

Un grup de probleme de selecție prin care se realizează sarcini asemănătoare, se constituie într-un strat al unei structuri ierarhizate. Ideea abordării ierarhizate constă în transformarea (descompunerea) unei probleme considerate mari (sau a unui sistem mare) într-o mulțime de subprobleme (sau subsisteme) mai mici.

Când structura cu mai multe niveluri se referă la o anumită procedură de rezolvare, eventual iterativă, a unei probleme de acces la baze de date heterogene, se abordează căutarea (de cele mai multe ori optimizată) multicriterială.

Sistemele de baze de date pentru obiecte virtuale au o arhitectură stratificată şi ierarhizată şi sunt prezentate sintetic în *Figura 6.3.*

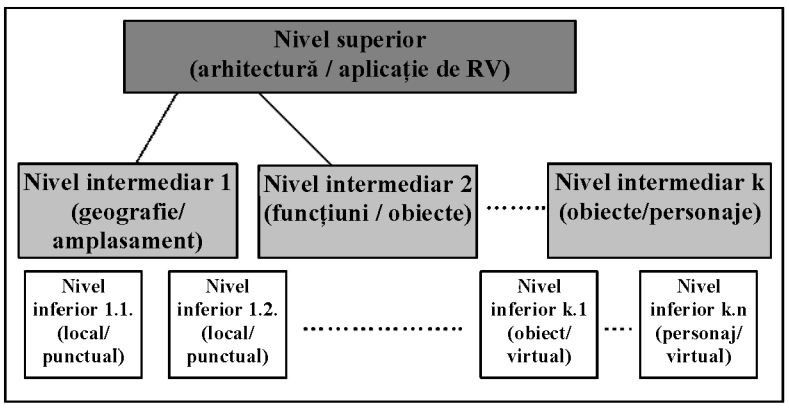

Figura 6.3. Schema arhitecturii bazelor de date cu obiecte virtuale

Subproblemele se aranjează într-o structură cu mai multe straturi, caracterizată prin următoarele:

- stratul de vârf conține, în esență, o singură subproblemă, celelalte niveluri conținând mai multe subprobleme;
- problemele straturilor inferioare sunt "definite" de către straturile superioare prin cereri de furnizare sau servicii;
- în rezolvarea unei cereri plasate pe un strat, se folosesc informațiile de reacție primite numai de la anumite părți din straturile inferioare şi anume, de la cele care îi sunt alocate şi subordonate;
- pe fiecare strat problemele sunt rezolvate în mod independent, de îndată ce se primesc informațiile de reacție şi semnalele de intervenție din straturile inferioare.

În cazul în care organizarea structurii este simultană cu desfăşurarea unor procese, sistemul abordează lucrul ierarhizat.

6.4. Structuri pentru baze de date care conțin obiecte virtuale

Având în vedere cerințele conceptuale pentru sistemele de gestiune a bazelor de date pentru obiecte virtuale şi delimitarea pe nivele ierarhice, se prezintă principalele tipuri de ierarhii, realizate pe baza complexității descrierii, deciziilor şi a organizării

unui sistem de realitate virtuală. Noțiunile generale de nivel şi ierarhie au fost particularizate ținând seama de contextul în care sunt aplicate.

a) Ierarhii bazate pe nivelele de complexitate a descrierii (abstractizării)

Un prim pas în analiza şi proiectarea unor sisteme complexe cum sunt cele de ***RV***, constă în realizarea modelului funcție de factorii, de obicei aflați în conflict:

- simplitatea (un model prea complicat poate avea uneori o valoare practică îndoielnică, în condițiile în care implică un efort şi timp de calcul mare);
- precizia (modelele exagerat de simplificate pentru eficiența în calcul, pot conduce la reacții incorecte).

Pentru a rezolva dilema între necesitatea de a considera multe aspecte (pentru a reprezenta în mod satisfăcător un fenomen) şi nevoia de simplitate (pentru reacție mai rapidă), se poate descrie un eveniment sau un obiect virtual printr-o familie de modele, care îi reflectă comportarea aşa cum este văzută de pe straturi de abstractizare diferite, având caracteristici, variabile, legi şi principii independente. Acelaşi sistem poate fi descris în termenii mărimilor fizico-chimice, ai variabilelor de răspuns, sau în termenii de calitate a graficii de sinteză.

Plasarea către baza ierarhiei permite obținerea mai multor detalii, în timp ce către vârf se poate dobândi o mai bună înțelegere a sensurilor şi implicațiilor globale.

b) Ierarhii bazate pe nivelele de complexitate a deciziilor

În situațiile în care trebuie să se acționeze în timp real, chiar în condițiile incertitudinii asupra datelor disponibile şi a consecințelor aplicării rezultatelor procesului de interogare a ***BD***, este recomandabil să se înlocuiască problema originală cu o familie de probleme mai simple, rezolvabile secvențial.

O aplicație de ***RV*** poate fi reprezentată ca o secvență de acțiuni legate de reglare, optimizare, organizare sau de conducere directă, supervizare, optimizare şi coordonare, pentru următoarele niveluri de complexitate (straturi): stabilizare, coordonare dinamică, optimizare statică şi optimizare dinamică.

Definirea straturilor de complexitate a prelucrărilor, poate fi legată fie de sensibilitatea față de variabilele de comandă, fie de o anumită succesiune temporală a proceselor, fie de frecvența perturbațiilor în variabilele procesului, în condițiile de funcționare, în parametri sau în structură. Mărimile de referință pentru parametrii mediului extern sunt modificate din timp în timp, printr-un proces de optimizare statică, pentru a compensa efectul unor perturbații lent variabile în timp (de exemplu în luminozitatea scenelor, umbrirea materialelor etc.), în timp ce mărimile de execuție trebuie modificate în timp real.

c) Ierarhii bazate pe nivelele de complexitate a organizării

Acest tip de ierarhie se defineşte considerând un sistem sub forma unei familii de subsisteme în interacțiune evolutivă. Dintre aceste subsisteme, unele sunt recunoscute ca unități în care se iau decizii, putând fi aranjate sub forma unei ierarhii în interiorul căreia se exercită anumite influențe.

Caracteristica esențială a ierarhiei constă în aceea că efortul total de prelucrare este împărțit între mai multe unități care au obiective diferite, uneori conflictuale sau necunoscute reciproc, dar şi un număr de grade de libertate variabil. Deşi o structură ierarhizată poate avea foarte multe straturi, ea se poate reduce la una cu două nivele: de coordonare (superior) şi local (inferior).

Pentru a profita de avantajele potențiale ale structurilor ierarhizate, este necesară realizarea prealabilă a unei descompuneri sistematice a problemei. Folosirea descompunerii nu este legată strict de sistemele de ***RV***. Metodele de descompunere-

coordonare se bazează pe o serie de manipulări elementare ale bazelor de date heterogene. Se pot distinge mai multe grupuri de manipulări elementare, aplicabile succesiv:

- transformări (înlocuiri) ale personajelor, obiectelor sau fenomenelor virtuale;
- transformarea de variabile;
- transformarea bazată pe "lagrangean";
- evoluția (modificarea în timp);
- descompuneri pentru împărțirea obiectivului în subobiective:
  - partiționarea;
  - descompunerea parametrică;
  - descompunerea structurală.

Unele manipulări elementare au ca scop pregătirea pentru folosirea manipulărilor din grupul al doilea. Folosirea unor combinații de manipulări permite definirea unor structuri ierarhizate atât în calcul, cât şi în organizarea funcționalităților.

6.5. Definirea sistemelor de gestiune a bazelor mari de date

6.5.1. Gestiunea bazelor mari de date

Sistemele de gestiune a bazelor mari de date reprezintă sisteme software specializate în stocarea şi prelucrarea volumelor mari de date. Termenul de *"bază mare de date"* se referă la volumul, cantitatea datelor de prelucrat şi modul de organizare al acestora pe suportul fizic de memorare, iar gestiunea bazelor mari de date este acțiunea de memorare şi prelucrare a acestor volume mari de date *[3]*.

Funcțiunile unui ***SGBD*** pentru *"baze mari de date"* sunt:

- definirea bazei de date;
- introducerea datelor primare, adăugarea de noi date;
- modificarea unor date existente şi / sau ştergerea lor;
- interogarea bazei de date pentru extragerea informațiilor stocate în aceasta;
- generarea de ieşiri în formatul impus de aplicație;
- organizarea bazelor de date pentru a permite un acces mai rapid (optimizarea accesului);
- lucrul cu interfețe şi periferice specifice aplicațiilor de ***RV***;
- managementul lucrului cu relații între bazele de date, pentru accesarea simultană;
- identificarea legăturilor şi efectelor.

Din punct de vedere al limbajelor pe care le utilizează, sistemele de gestiune pentru baze mari de date care utilizează *limbaje gazdă*, realizează activitățile de creare, actualizare şi interogare bază de date utilizând limbajele de nivel înalt sau extensii ale acestora; limbajele de nivel înalt oferă posibilități complexe, dar au dezavantajul că formularea cerințelor este dificil de implementat.

Sistemele de gestiune baze mari de date cu *limbaj autonom*, folosesc un limbaj independent de limbajul gazdă şi au o independență totală față de platforma pe care rulează; prezintă avantajul portabilității mari şi a unei simplități sporite în formularea cerințelor, motiv pentru care sunt sistemele cu cea mai mare utilizare.

Din punct de vedere al concepției de realizare, se pot utiliza următoarele tipuri de sisteme de gestiune a bazelor mari de date:

- sisteme ce utilizează pointeri logici şi fizici;

- sisteme relaționale, ce aplică teoria ansamblurilor şi relațiilor dintre acestea;
- sisteme orientate obiect;

Ca particularități care recomandă SGBD-urile orientate obiect, pentru aplicații de ***RV,*** se enumeră:

- utilizarea de limbaje de programare orientate obiect;
- crearea şi utilizarea de baze de date de obiecte (datele stocate pot fi imagini, sunete, diverse alte obiecte cuprinse toate în conceptul general de multimedia);
- practicarea de interfețe grafice prietenoase orientate obiect;
- sisteme de gestiune pentru baze de date distribuite;
- sisteme de gestiune baze de date funcționale;
- sisteme de gestiune baze de date deductive (utilizate în sisteme expert).

Înregistrarea fizică este ansamblul de date care face obiectul transferului între suportul tehnic de memorare şi memoria internă, la o operație de intrare-ieşire programul lucrând la nivel de înregistrare logică. Un astfel de ***SGBD*** trebuie să satisfacă următoarele funcții:

- funcția de descriere - asigură definirea structurii datelor, a constituanților, a relațiilor dintre aceştia, a condițiilor de acces la informații;
- funcția de manipulare - asigură crearea bazei de date, inserarea sau suprimarea (ştergerea) de informații, modificarea înregistrărilor deja create; permite căutarea, sortarea (ordonarea) şi prezentarea în ieşire, totală sau parțială a bazei de date;
- funcția de utilizare - asigură comunicarea utilizatorului cu baza de date, prin interfețe avantajoase pentru orice tip de utilizator.

Utilizatorii pot fi clasificați astfel:

- utilizatori liberi sau convenționali, care necesită interfețe prietenoase, cu o formă cât mai apropiată de limbajul natural, mesaje de eroare şi help-uri interactive;
- utilizatori parametrici, care fac uz de limbaje de manipulare prin utilizarea de procedee predefinite activate doar prin specificarea numelui şi a parametrilor;
- utilizatori de aplicații: prin realizarea unei interfețe între baza de date şi utilizator pentru extragerea de informații;
- administratorul bazei de date (are nevoie se un software special care să-i permită să-şi realizeze funcțiile specifice).

Elementele software cu care interacționează un astfel de ***SGBD*** sunt sistemul de operare şi programele de aplicații. Limbajul de descriere defineşte corect elementele componente ale bazei de date, relațiile între ele, legăturile cu programele de aplicație şi tehnica de memorare. Sistemul de operare permite identificarea la nivel fizic a datelor şi transferul acestora în buffer-ele sistemului, ***SGBD***-ul putând funcționa dependent sau independent de sistemul de operare *[3]*.

6.5.2. Baze de date distribuite

Concepția, elaborarea, exploatarea şi întreținerea unui sistem distribuit de baze mari de date, aşa cum sunt sistemele de ***RV***, impun concentrarea unor resurse umane şi materiale importante pentru o perioadă îndelungată de timp. Aceste greutăți sunt generate atât de dimensiunea şi complexitatea problemelor de rezolvat, cât şi de faptul că instrumentele tehnologice specifice sunt încă limitate. În acelaşi timp, cerințele

aplicațiilor de ***RV*** privind criteriile de performanță, fiabilitate, interfață de utilizare şi raportul performanță/cost, reprezintă tot atâtea elemente de dificultate puse în fața celor care proiectează o aplicație de ***RV*** *[3].*

Centralizarea şi descentralizarea sunt două tendințe opuse care au marcat alternativ evoluția sistemelor informatice. Bazele de date distribuite reprezintă unul dintre compromisurile cele mai profitabile, păstrând avantajele ambelor abordări. În 1988, Stonebraker prognoza că în următorii 10 ani bazele de date centralizate vor deveni doar o "curiozitate antică". Interesul de care se bucură bazele de date distribuite, atât în comunitatea ştiințifică cât şi în cercurile comerciale, demonstrează importanța acestui subiect.

Există câteva elemente cheie care au motivat în cazul aplicațiilor de ***RV*** alegerea unor sisteme distribuite de baze de date. Vitezele de comunicație sunt mai mari în sistemele distribuite, cât timp accesul la un computer local este mai rapid decât accesul la distanță.

Chris Date a formulat un set de obiective relative la bazele de date distribuite. Cu toate că Date însuşi avertizează că aceste obiective nu sunt cu totul independente unele de altele, ele reprezintă un cadru foarte bun pentru a caracteriza într-o manieră sistematică funcționarea sistemelor de baze de date distribuite. Celor 12 obiective li se adaugă un "principiu fundamental", numit "Regula Zero" *[3]*:

Pentru utilizator , un sistem distribuit trebuie să arate exact ca un sistem nondistribuit – „REGULA ZERO"

Într-o aplicație de ***RV*** distribuția este transparentă pentru utilizator, toate aspectele legate de distribuție presupunându-se a fi rezolvate intern sau la nivelul implementării.

Cele 12 obiective privind utilizarea bazelor de date distribuite pentru aplicații de ***RV*** se prezintă în continuare:

1. ***Autonomia locală*** - fiecare locație corespunzătoare bazei de date este autonomă. Baza de date locală poate să funcționeze independent de starea altor locații, cu toate că aplicațiile software implică o anumită interdependență. Datele locale aparțin, din punct de vedere administrativ, organizației care le gestionează, le întreține, le asigură securitatea şi integritatea.

2. Nu există dependență de o bază de date centrală, ***toate locațiile sunt egale ca statut***, nu există deci o locație "master" pentru anumite servicii, astfel încât celelalte să depindă de aceasta. Dependența de o locație centrală s-a evitat din două motive majore: poate conduce la gâtuirea sistemului şi creează un punct unic de vulnerabilitate; căderea locației centrale ar conduce la căderea întregului sistem. Acesta este motivul pentru care subsistemul de interfațare trebuie să aibă o concepție multistrat în sensul existenței de module de interfațare.

3. ***Operare continuă*** - sistemele distribuite asigură un nivel superior de fiabilitate; sistemul distribuit poate să funcționeze chiar în condițiile căderii unor componente individuale şi oferă un nivel sporit de disponibilitate ca urmare a fiabilității sporite şi a posibilității de replicare a datelor. La modul practic sistemul de ***RV*** nu necesită opriri planificate (pentru upgrade sau adăugarea unei locații) efectul opririlor neplanificate (pene) fiind limitat.

4. ***Independență de localizare*** - ideea acestui tip de independență, numit "transparența localizării datelor" este că utilizatorul nu are nevoie să ştie unde se află stocate fizic datele, pentru a putea opera cu ele. Sistemul se comportă ca şi cum toate

datele ar fi stocate local. Aceasta simplifică activitățile utilizatorului final, permițând migrarea datelor între locații precum şi mutarea lor în rețea, pentru a optimiza performanțele sau pentru a reflecta dinamica activității, fără a afecta aplicațiile existente sau activitățile uzuale.

5. ***Independența de fragmentare*** - un sistem care suportă fragmentarea datelor prin operațiile relaționale de restricție şi protecție, trebuie să suporte şi cerința ca fragmentarea să fie transparentă pentru utilizatorul final şi pentru programele de aplicație. Aplicațiile de ***RV*** se comportă ca şi cum datele nu ar fi fragmentate.

6. ***Independență de replicare*** - replicarea este transparentă pentru utilizator, care nu are nevoie să ştie dacă lucrează cu replicări locale ale datelor. Este responsabilitatea sistemului să propage eventualele actualizări în restul sistemului, păstrând integritatea şi coerența datelor.

7. ***Procesare distribuită a interogărilor*** - spre deosebire de un sistem (să presupunem relațional) nedistribuit, în cazul sistemelor distribuite, optimizarea procesării unei cereri este globală şi evaluează în mod corespunzător activitățile specifice, legate de transportul datelor între locațiile implicate în cerere.

Practic, aceasta înseamnă minimizarea numărului de mesaje şi minimizarea dimensiunii rezultatelor intermediare care se cer transmise între sistem şi utilizator.

8. ***Administrarea distribuită a tranzacțiilor*** - există două aspecte majore legate de administrarea tranzacțiilor: controlul recuperării şi controlul concurenței. Deoarece bazele de date locale sunt autonome, o tranzacție constă din analiza mai mulți agenți software, fiecare dintre aceştia desemnând un proces desfăşurat într-o anumită locație, în virtutea respectivei tranzacții. Controlul recuperării trebuie să asigure execuția tranzacției după principiul „totul sau nimic", ceea ce, într-un mediu distribuit, înseamnă că toți agenții finalizează la unison modificările (commit) sau anulează la unison modificările în caz de eşec (roll back). Acest efect poate fi obținut prin utilizarea protocolului "two phase commit". Controlul concurenței se bazează pe mecanisme de blocare (locking) analoge celor din sistemele nedistribuite.

9. ***Independența de hardware*** - practic, aceasta înseamnă că, indiferent de platformele hardware din care este compus sistemul, pentru utilizator apare imaginea unui sistem unic, cu toate maşinile participând ca parteneri egali.

10. ***Independența de sistemul de operare*** - acest obiectiv este strâns legat de cel precedent, deoarece în principiu, platforme hardware diferite implică sisteme de operare diferite.

11. ***Independența de rețea*** - din moment ce sistemul distribuit este independent de platforma hardware şi software, este de la sine înțeles că poate utiliza o varietate de rețele de comunicații, locale sau de arie largă.

12. ***Independența de SGBD*** - pe fiecare post de aplicație este instalată o copie a sistemului de gestiune a bazelor de date deci, există un sistem distribuit omogen.

În luarea deciziei de adoptare a unui sistem de baze de date distribuite, trebuiesc analizate şi dezavantajele prezentate de sistemele distribuite. Acestea pot ridica atât probleme de sincronizare, de consistență şi integritate a datelor, cât şi probleme legate de controlul accesului.

De-a lungul anilor, s-a trecut de la bazele de date ierarhice la cele relaționale, de la centralizare la distribuire, fiecare dintre aceste etape implicând dezvoltarea şi implementarea unor concepte suplimentare de descriere, stocare şi regăsire a datelor. Dacă structurile de date pot să pară naturale unui proiectant sau elaborator de programe software, nu acelaşi lucru se întâmplă cu utilizatorul obişnuit. În realizarea

aplicațiilor de ***RV*** trebuie pus accent pe caracterul "prietenos" pe care acestea să îl prezinte utilizatorului final.

Realizarea practică a prototipului de aplicație, implică luarea de decizii în ceea ce priveşte plasarea datelor şi a programelor în nodurile rețelei şi la perifericele specializate. Distribuția aplicației presupune atât distribuția programelor software de bază, cât şi distribuția programelor de aplicație care vor rula. Dacă rețeaua de periferice este deja funcțională, procesarea aplicației de ***RV*** revine la rezolvarea a două etape: stabilirea nodului optimal pentru consultarea bazelor de date potrivit cererii beneficiarului şi procesarea distribuită a cererilor *[3]*.

6.5.3. Cerințe tehnice şi de standardizare pentru aplicațiile de realitate virtuală

O problemă importantă pentru sistemele de ***RV*** este eliminarea paralelismelor în actualizarea datelor şi redondanței în înmagazinarea acestora, iar prin multiplicarea criteriilor de regăsire şi micşorarea complexității operației de identificare, se poate scurta timpul de răspuns la cererile de regăsire.

O primă accepțiune este aceea că, fiind sisteme deschise, sistemele de baze de date pentru medii virtuale necesită un mediu de dezvoltare de aplicații bazat pe standarde de interfață ce permit portabilitatea aplicațiilor, portabilitatea utilizatorului şi inter-operabilitatea.

Conform Comitetului IEEE-POSIX, un sistem deschis este un sistem care permite: realizarea de module software şi aplicații portabile într-o gamă largă de platforme hardware; interacțiunea cu alte aplicații din sisteme locale sau distributive; interacțiunea cu utilizatorii, care să asigure şi portabilitatea.

Un element important al acestei cerințe îl constituie termenul de *specificație deschisă*, care reprezintă o specificație publică actualizată printr-un proces public, permițând adaptarea unei tehnologii şi în conformitate cu standardele deja existente.

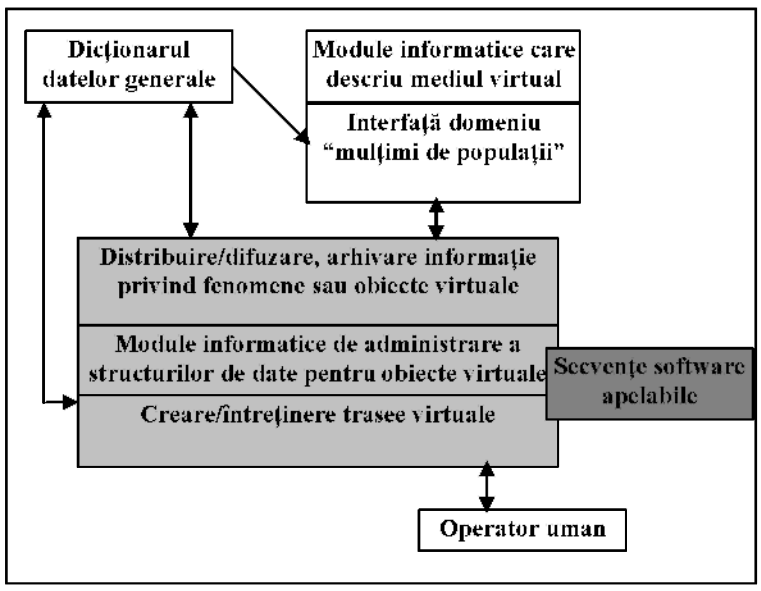

Figura 6.4. Arhitecturi de aplicații de RV cu acces la baze de date distribuite

Necesitatea adoptării conceptului de sisteme deschise pentru produse de ***RV***, poate fi sintetizată astfel:

- asigură portabilitatea aplicațiilor, datelor şi resurselor informatice între diferite sisteme hardware şi periferice inter-operabile;
- permit inter-operabilitatea aplicațiilor şi modulelor;
- asigură independența față de un mediu particular software sau hardware;
- asigură flexibilitatea în efectuarea de adaptări sau dezvoltări ale mediului (sistemului) şi de alegere a modelului de aplicație pentru fiecare din problemele specifice;
- integrează module de aplicații, informații şi sisteme provenind din surse diferite, într-un mediu coerent şi productiv.

Pentru elaboratorii de software pentru ***RV***, sistemele deschise oferă posibilitatea de a utiliza mai multe platforme hardware sau diferite generații software, de la furnizori diferiți.

Corporation for Open Systems ***(COS)*** a dezvoltat un model de mediu integrat pentru sisteme deschise. Acest model se concentrează pe acele zone de interacțiune şi interfață care sunt critice pentru o aplicație ce funcționează într-un sistem deschis. Modelul COS utilizează acronimul ***MUSIC*** pentru a clasifica elementele de bază ale unui sistem deschis şi standardele corespondente acestora. Acestea sunt *[3], [15]*:

M = management - componente care realizează funcțiuni specifice de administrare sistem, securitate, control al rețelelor şi al resurselor, evidența şi controlul configurației sistemului. Unele componente, cum ar fi securitatea şi clasarea, trebuie să asigure compatibilitatea pentru o întreagă mulțime de platforme hardware, astfel încât toate resursele să poată fi accesate şi administrate în totalitate, în cadrul mediului integrat. Componentele acestui element sunt destinate unei clase speciale de utilizatori: administratori de sistem, administratori de rețea şi operatori u drepturi speciale.

U = utilizator - elementul de interfață cu utilizatorul este format din două componente de bază: un set de interacțiuni (conexiuni) ce apar în general între utilizator şi platforma aplicațiilor (mai puțin cu aplicația activă în acel moment). Exemple de astfel de interacțiuni sunt modul particular în care un utilizator startează o aplicație sau salvează un traseu virtual. A doua componentă este constituită din interfața curentă între utilizator şi aplicație. U - lucrează cu termeni ca: ergonomie, interacțiunea structurilor, uşurința în mişcare (acel atribut al interfeței utilizator ce stă la baza abilității utilizatorului de a se "mişca" de la o platformă de aplicații la alta, fără dificultăți).

S = sistem de operare - element ce include interfețele pentru aplicații şi programe de sistem, precum şi servicii pentru platforma de aplicații. Aceste programe includ partea tipică a sistemului de operare, în plus aşa numitele API, interfețe pentru programe de aplicație, ce furnizează servicii grafice şi standarde pentru limbaje. Componentele acestui element afectează puternic portabilitatea aplicațiilor şi sunt de un mare interes pentru programatorii de aplicații de ***RV.***

I = interfața - acest element include funcțiuni necesare accesului şi transferului de date. El se referă la următoarele domenii: accesul aplicațiilor la date, formate de transfer şi codificări ale datelor. Serviciile orientate pe obiecte sunt, de asemenea, incluse în acest concept. Serviciile oferite au impact asupra interoperabilității.

C = comunicație - conține componentele de comunicație reclamate de următorii termeni generici: rețele, interconectări şi interoperabilitate. Se adresează atât rețelelor locale, cât şi celor distribuite. Administratorii sunt cei ce resimt impactul componentei.

Diferitele componente ale unui sistem de ***RV*** reprezintă suportul fizic pentru diferite straturi conceptuale *[6], [13]*. De aceea, ca sistem deschis acesta trebuie să acopere o arie largă de domenii, între care cele mai importante sunt:

- schimbul de informații şi sincronizarea activităților (comunicație, interpunere);
- reprezentarea datelor (crearea şi menținerea descrierilor, transformărilor şi translatărilor de date);
- memorarea datelor (medii de memorare, gestiune date heterogene);
- gestiunea proceselor şi resurselor necesare;
- integritatea şi securitatea sistemului;
- suportul de programare;
- definirea, testarea, memorarea, transferul şi accesul la programe.

Cu privire la arhitectura multistrat a sistemului de ***RV***, fiecare componentă trebuie să ofere puncte de conexiune cu celelalte nivele şi rutine informatice. Având în vedere necesitatea de agregare şi dispersie a informației pe diferite nivele şi pe diverse areale, comunicația dintre componentele sistemelor trebuie să asigure:

- stabilirea conexiunii;
- transferul informațiilor (primare → prelucrate);
- eliberarea conexiunii;

În definirea straturilor unei aplicații de ***RV*** pentru prototipizare virtuală s-au avut în vedere următoarele probleme particulare:

- crearea prea multor straturi duce la dificultăți în înțelegere şi prelucrarea lor;
- o limită de strat poate fi marcată acolo unde numărul interacțiunilor peste această limită este minim;
- funcțiunile evident diferite, vor fi tratate diferit;
- funcțiunile similare trebuie să facă parte din acelaşi strat;
- o limită poate fi stabilită şi acolo unde o indică experiența proiectantului;
- un strat trebuie să faciliteze satisfacerea nevoilor legate de nivele diferite de tratare a datelor;
- delimitarea unui strat trebuie să permită schimbări ale funcțiilor şi protocoalelor din acel strat, fără a fi afectate alte nivele;
- interfețele între straturile adiacente sunt obligatorii.

6.6. Algoritmi pentru prezentarea sub formă multidimensională a datelor

Realizarea subsistemului de interfațare la sisteme cu aplicații de ***RV*** trebuie să asigure posibilități multiple de prezentare a datelor. În urma analizei informațiilor obținute din bazele de date, modelul multidimensional de prezentare a prelucrărilor realizate pe acestea trebuie să reprezintă alternativa cea mai atractivă. Pentru acesata, se vor apela principiile de realizare a unei afişări multidimensionale pentru prelucrările efectuate. În acest context se dezvoltă câteva noțiuni teoretice legate de forma de prezentare multidimensională a datelor.

Orice analiză vizează cu precădere anumite variabile, numite metrici, care de obicei au o natură grafică. Metricile nu există în mod izolat. O valoare a metricii naşterilor nu exprimă în sine nimic. Valorile metricilor capătă sens doar însoțite de anumiți calificatori. Calificatorii conceptuali ai metricilor se numesc dimensiuni, în cazul de față: datele de stare. Alături de timp şi structura ambientală, acestea sunt exemple tipice de dimensiuni.

Pot fi imaginate şi alte dimensiuni, în funcție de activitatea modelată sau metrica într-o situație concretă. Dimensiunile sunt însă abstracțiuni ("calificatori conceptuali"), care nu se memorează. Ele se concretizează prin intermediul atributelor, care reprezintă calificatorii specifici ai metricilor. Orice atribut se asociază unei singure dimensiuni. O dimensiune poate fi exprimată prin mai multe atribute. Exemple tipice de atribute pentru dimensiunea timp: ziua, săptămâna, luna, trimestrul, anul, iar pentru aşezarea geografică: țara, regiunea, localitatea, incinta, încăperea etc.

Adeseori, atributele asociate unei dimensiuni au un caracter ierarhic, formând ierarhii de atribute. De exemplu, atributele asociate dimensiunii geografice formează de regulă ierarhii de tipul: țară, diviziune administrativă, localitate. Se pot da şi exemple practice de atribute multiple asociate unei dimensiuni şi care să nu aibă caracter ierarhic. De pildă, pentru dimensiunea "personaj", pot exista atributele locuință şi meserie, care nu formează o ierarhie.

Este posibil ca numai o parte dintre atribute să formeze o ierarhie, ca în situația în care există şi atributul îmbrăcăminte (exemplu: în condițiile în care fiecare personaj locuieşte într-o anumită încăpere şi fiecare încăpere aparține cetății medievale).

Datele prezentate după dimensiuni multiple sunt numite la modul generic "cuburi de date" (chiar dacă pot avea mai mult sau mai puțin de trei dimensiuni). Viziunea multidimensională a datelor este utilă în analiza performanțelor aplicației, deoarece oferă posibilitatea de a privi datele din diverse perspective.

Secțiunile bidimensionale în aceste cuburi de date se numesc *tablouri*. Nu întotdeauna reprezentările multidimensionale sunt potrivite nevoilor analizei.

Dacă se consideră o bază de date de „personaje" în care, pentru fiecare individ virtual, sunt înregistrate date personale (de exemplu: numele, marca, vârsta şi locația) o reprezentare multidimensională nu este relevantă. Sau, dacă sexul personajului este metrica luată în considerare, vârsta nu poate fi o dimensiune relevantă, iar tabloul având numele ca o dimensiune şi vârsta ca o altă dimensiune ar fi ineficient. Dacă însă se doreşte o analiză privind încăperile şi ocupațiile / meseriile pentru femeile adulte, statistica acestora analizată în funcție de traseele de vizitare sau în funcție de ambientul geografică are o semnificație evidentă).

Ca regulă generală, cu cât există un număr mai mare de interdependențe semnificative între elementele unui set de date, cu atât este mai plauzibil ca studiul acestor interdependențe într-o formă multidimensională să furnizeze informații valoroase.

Rotațiile reprezintă operațiile cele mai uzuale în interfețele multidimensionale. Ele oferă utilizatorului posibilitatea de a alege perspectiva asupra datelor, necesară în funcție de analiza efectuată.

În cazul reprezentării tridimensionale, există 6 rotații posibile, care oferă orice perspectivă asupra datelor dorite. Fiecare rotație pune în evidență o nouă perspectivă, aducând în prim plan o structură bidimensională, o "fațetă" a datelor, motiv pentru care rotația se mai numeşte "data sliceing". Faptul că rotația aduce în prim plan o nouă "fațetă" bidimensională nu este suficient, pentru anumite funcțiuni care au nevoie de viziuni specifice ale datelor.

Prin combinarea de rotații şi secțiuni se obțin viziuni semnificative şi structuri particulare. Pornind de la cubul de date din *Figura 6.5.*, se poate obține o viziune a datelor care reflectă funcțiunea activă a aplicației, în secvența de timp considerată.

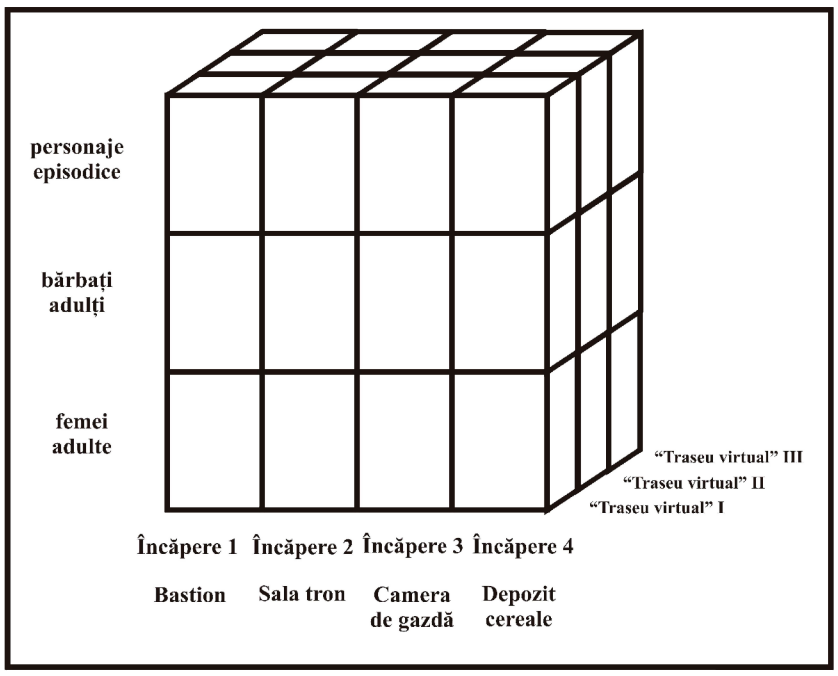

Figura 6.5. Cubul de date prezentate multidimensional (exemplificare pentru aplicația de RV „Cetatea de Scaun Suceava")

Diferite cerințe pot conduce la selectarea unor anumite valori ale atributelor unor dimensiuni pentru a obține viziuni specifice. Rezultatul constă în definirea unor "cubulețe" de date.

Un aspect foarte important este faptul că sistemele de ***RV*** lucrează în mod inerent cu date la nivel de detaliu şi în acelaşi timp, este foarte adesea interesat de imagini globale asupra anumitor aspecte. Din acest motiv, exemplul prezentat în *Figura 6.5.* nu se referă la datele de detaliu, ci porneşte de la premiza că datele primare au fost în prealabil „pregătite" într-o manieră care să le facă mai utile pentru prezentare.

Cu toate că multe instrumente pot realiza dinamic toate calculele necesare, această operațiune necesită timp şi resurse, asfel încât pentru acele seturi de date care sunt în mod frecvent cerute într-o astfel de formă, se preferă precalcularea unor valori. Această operație este numită *consolidare* (atunci când se referă la acceptul conceptual), *sumare* (din perspectivă procedurală), sau *agregare* (când este vizat aspectul structural). Termenii pot fi însă considerați sinonimi, în absența unor precizări explicite.

Întotdeauna aceste agregări se fac după dimensiunile corespunzătoare ale acesteia. Pentru atributele organizate ierarhic, consolidarea se face nivel cu nivel. De cele mai multe ori, sumarea implică doar calcularea unor valori, dar există situații în care se cer aplicate formule mai complicate sau anumite procedee speciale. Nivelul la care se face sumarea în cazul în care sunt implicate ierarhii de atribute se numeşte glanularitate. Interfața multidimensională oferă ca opțiuni fundamentale operațiile evocate în exemplele de mai sus, împreună cu altele, care permit reducerea numărului de dimensiuni sau calculații la nivelul metricilor. Mai mult chiar, posibilitățile de vizualizare prin Web şi integrarea unor vizualizări bazate pe ***VRML*** conduc la rezultate spectaculoase.

6.7. Asigurarea integrității și a actualizării performante a datelor

Integritatea datelor într-o bază mare de date relațională trebuie menținută mai ales în timpul accesului multiplu la date. Concurența reprezintă procesul de distribuire a resurselor între mai mulți utilizatori sau programe şi module de aplicație, ce rulează în acelaşi timp. Pentru a asigura integritatea datelor, se impune ca modificările să se facă cât mai rapid, dacă se poate chiar simultan. În acest sens, se recomandă ca un subsistem de interfațare să asigure modificarea datelor numai în versiunea activă.

Evenimentele nedorite pot fi:

1. Pierderea actualizărilor: două aplicații A şi B, pot să citească aceeaşi linie din tabelă şi să calculeze noi valori pentru una din coloane. Dacă A reuşeşte actualizarea şi apoi B face propria actualizare, valoarea coloanei actualizată de A este pierdută.

2. Acces la date neprocesate: dacă aplicația A actualizează valoarea unei coloane, pe o linie, aplicația B citeşte acea valoare înainte de sfârşitul sesiunii active A, dar aplicația A renunță ulterior la executarea activării, atunci B efectuează calcule cu o valoare invalidă.

3. Citiri nerepetabile: aplicația A citeşte o linie din tabelă, ale cărei date le prelucrează, apoi procesează alte linii. Intre timp, aplicația B modifică valorile coloanelor de pe acea linie sau chiar o şterge. Când aplicația A revine pe acea înregistrare, o găseşte modificată sau chiar ştearsă, deşi nu a finalizat propria activare.

4. Fenomenul de citire fantomă: aplicația A execută o cerere de citire linii dintr-o tabelă, după anumite criterii. Aplicația B inserează date noi, sau actualizează datele selectate de A. Aplicația A repetă cererea de selecție date după criteriul anterior, constatând apariția unor linii adiționale “fantomă”.

Problemele de mai sus se rezolvă prin introducerea procedurii de “izolare” sau “încuiere” a datelor selectate, procesul având efect pe durata “sesiunii de lucru” a cererii utilizator.

Referințe Bibliografice

[1]Andoni I.	*Sisteme expert: Principii şi dezvoltarea aplicațiilor de gestiune, Iaşi, editia 2006*
[2]Bergounioux, M.[Maïtine]	*Mathematical Image Processing, Springer 2011, ISBN: 978-3-642-19603-4*
[3]Bodea Constanța, Lungu I.C., Bădescu Georgeta	*Baze de date: organizare, proiectare şi implementare, Editura ALL Educational, Bucureşti, editia 2006*
[4]Cecal Liana	*Programe Object Windows, Editura Agni, 2004*
[5]Chaillou C., Froumentin M.	*La Synthese d'images, École Universitaire d'ingénier de Lille, France, publicație electronică, 2010*
[6]Chaillou C.	*Architectures des Systèmes pour la Synthèse d'Images, Dunod Informatique, France, 2002*
[7]Coad P., Yourdon E.	*Object Oriented Design, Ed. Prentice Hall International Inc., 1999*
[8]Coster M., Chermant J.	*Précis d`analyse d`images, Presses du CNRS, Paris, 1999*
[9]Duşa S.	*Microsoft ODBC (Open DataBase Connectivity), PC-Report, octombrie, 2004*
[10] Fieguth, P.	*Statistical Image Processing and Multidimensional Modeling, Springer 2010, ISBN: 978-1-4419-7293-4*
[11]Grueber M.	*Understanding SQL, 2009*
[12]Kalay Y.E.	*Redefining the role of computers in architecture: from drafting/modelling to knowledge – based assistants, Comput.-Aided Des., vol17, no.3, 2002, p.319-328*
[13]Milosescu M.	*Baze de datein Visual FoxPro, Editura Teora, 2005*
[14]Mocanu Aura-Mihaela	*Baze de date spațiale. Analiza, proiectarea şi dezvoltarea unui GIS. (publicație electronică, 2009)*
[15] Mukhopadhyay, J.	*Image and Video Processing in the Compressed Domain, CRC Press 2011. ISBN: 9781439829356*

[16]Robison L . — *Programarea bazelor de date cu Visual C++6, Editura Teora, 2009,*

[17]Salomon D. — *Computer Graphics Manual Springer London Ltd ISBN: 978-0-857-29885-0 ISBN-10: 0-857-29885-2, 2011 United Kingdom*

[18]Solomon C., Breckon, T. — *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab, Wiley 2011. ISBN: 978-0-470-84472-4*

[19]Stoica L. — *Desenul digital in arhitectura, Bucuresti 2011, ISBN 978-973-0-10574-2*

[20]Tâmbulea L. — *Access pentru programatori, Editura Promedia Plus, Cluj-Napoca, 2006*

[21]Ulman J. — *Principle of Database and Knowledge-Base Systems, 2007*

[22]Weber P., Chapman D. — *Investing in geography. A GIS to support inward investment, Computers, Evironmental and Urban Systems, no. 33, 2009*

*[23]**** — *office.microsoft.com/ro-ro/access-help/despre-migrarea-unei-baze...*

*[24]**** — *http://www.scritube.com/stiinta/informatica/autocad/Baze-de-date-si-AutoCAD.php*

*[25]**** — *Improved pogramming techologies, IBM, 2009*

*[26]**** — *Microsoft Developer Network, 2010*

*[27]**** — *Microsoft internet Information Server, Microsoft Corporation, 2012*