UNIVERSITATEA "POLITEHNICĂ"
TIMIŞOARA

ing. Daniela-Carmen REIMELT

TEZĂ DE DOCTORAT

Conducător ştiinţific
Prof. dr.ing. Ioan Nicoară

TIMIŞOARA 2003

UNIVERSITATEA "POLITEHNICĂ"
TIMIŞOARA

ing. Daniela-Carmen REIMELT

TEZĂ DE DOCTORAT

APLICAREA TEHNICILOR DE PROIECTARE ORIENTATE PE OBIECTE ÎN CONSTRUCŢIA APARATELOR ŞI ECHIPAMENTELOR

Conducător ştiinţific
Prof. dr.ing. Ioan Nicoară

TIMIŞOARA 2003

Cuprins

1 Introducere

Întreprinderile stau la ora actuală în faţa unei probleme dificile. Libertatea pieţei, concurenţa ofensivă şi modificarea permanentă a cerinţelor beneficiarilor impun o orientare mai puternică asupra pieţei, precum şi flexibilitate maximă în domeniul proceselor şi al formelor de organizare. Evoluţia rapidă a tehnologiilor şi a mijloacelor de comunicaţie oferă noi modalităţi de soluţionare a problemelor. Introducerea de noi tehnologii implică însa o transformare revoluţionară a modului de gândire, a metodelor de proiectare şi construcţie, a sistemelor de fabricaţie si de prelucrare a informaţiilor: ele trebuie să fie flexibile şi să servească realizării de noi produse, a unor forme de organizare şi a unor procese noi. Doar întreprinderile care reuşesc să facă fată acestor cerinţe, avand o arhitectura de specialitate bazata pe procese optimizate si flexibile, la care se asociaza o arhitectura tehnica stabila si adaptabila, vor exista o perioadă îndelungată pe piaţă.

In acelasi timp, revoluţia mondială în domeniul calităţii conduce la extinderea sensului pe care îl are termenul "calitate". Dacă în trecut, acest termen, s-a referit mai mult la aspectele tehnice ale unui produs, în prezent s-a impus în toate domeniile referindu-se la un proces dinamic de îmbunătăţire (Total Quality Management).

Managementul calităţii are ca scop creşterea valorii calitative ale unui anumit produs pentru beneficiari şi utilizatori. Orientarea pe calitate în procesele de proiectare conduce la diminuarea costurilor cauzate de erori, precum şi la diminuarea timpului de proiectare.

Procesele de transformare îşi pun amprenta şi asupra activităţii de proiectare. Dacă în trecut proiectarea s-a desfăşurat izolat, în cadrul unor grupuri specializate strict pe domenii, astăzi, conlucrează la rezolvarea unei probleme specialişti din domenii variate. Odată cu stăpânirea complexităţii problemei şi a elaborării unor soluţii optime, trebuie tinut cont din ce în ce mai mult de aspecte particulare de management, precum şi de aspecte psihologice si sociale în activitatea de proiectare.

O atenţie deosebită se acordă în această lucrare metodelor de dezvoltare software. Dezvoltarea de software capătă în prezent o importanţă din ce în ce mai mare în domeniul construcţiilor de maşini. Aceasta se explică prin creştera ponderii componentelor software în produse mecatronice şi în linii de producţie automatizate. Metodele de lucru orientate pe obiect, reprezentând standardul actual in dezvoltarea de software, se recomandă a fi utilizate în activităţi de modelare şi construcţie din domeniul construcţiilor de maşini şi echipamente, pentru a mări eficienţa de lucru în proiectare.

Unul dintre cele mai importante aspecte este comunicarea în cadrul unor echipe mixte. Cele mai multe proiecte eşueaza din cauza absenţei unei comunicări adecvate. Este absolut necesar într-o echipă de proiectare formată din specialişti din domenii diferite (de exemlu, ingineri mecanici, informaticieni, graficieni, economişti ş. a.) utilizarea unui limbaj de dialog comun in toate fazele de proiectare.

Legile economiei de piaţă dictează flexibilitate si rapiditate în realizarea de noi produse. Aceasta presupune utilizarea unor sisteme flexibile şi stabile în proiectarea asistată de calculator. Utilizarea uneltelor de proiectare trebuie îmbinată optim cu efortul manual de proiectare. Astfel, trebuie găsite mijloace optime pentru descrierea variantelor, pentru conducerea proceselor de fabricaţie, automatizarea unor procese. Solutiile PLM (Product Lifecycle Management) confera unei intreprinderi avantaje strategice in cadrul concurentei de piata prin posibilitatea inovatiilor si a reactiilor flexibile la cerintele consumatorilor. Sistemele informatice destinate managementului ciclului de viata al unui produs au ca scop utilizarea optima a cunostintelor de specialitate ale inginerilor, informaticienilor si a celorlalti parteneri din lantul de productie. Structurarea corespunzatoare a datelor reprezinta baza unei modularizari rationale ale proiectelor si ale produselor de seie. Prin aceasta se creeaza premisele unor produse flexibile, adaptabile cerintelor consumatorilor, precum si posibilitatea realizarii acestor produse oriunde pe lume prin utilizarea tehnologiilor de internet.

Importanţa şi actualitatea temelor abordate in aceasta lucrare este justificată pe de o parte prin necesitatea utilizării calculatoarelor în procesele de proiectare şi de fabricaţie, pentru a mări eficienţa proiectarii şi calitatea produselor, precum şi

prin absenţa, pe de altă parte, a unor solutii PLM standard care integrează mulţimea de operaţii specifice construcţiei asistate de calculator.

Lucrarea de faţă este pe de o parte o sinteză a experienţei autoarei în domeniul activitatii de proiecte şi o încercare de apropiere a informaticii de domenii, cum ar fi proiectarea asistata de calculator in construcţia de maşini şi echipamente, precum şi de aplicare a unor metode din domeniul dezvoltării software, în proiecte tehnice, cu scopul îmbunătăţirii calitative a procesului de proiectare.

Experienta autoarei in domeniul proiectelor tehnice cu aplicatii pe calculator s-a acumulat de-a lungul anilor de studiu in cadrul Facultatii de Calculatoare si Automatica din Universitaea „Politehnica" Timisoara si de-a lungul anilor petrecuti ca inginera stagiara in centrul de calcul al firmei AEM-SA din Timisoara, apoi ca asistenta universitara la Catedra de informatica economica a Universitatii de Vest din Timisoara, iar din 1994 ca arhitecta in domeniul sistemelor si aplicatiilor informatice precum si conducatoare a numeroase proiecte in cadrul concernului german HDI Haftpflichtverband der Deutschen Industrie cu sediul in Hanovra, Germania. Prin colaborarea cu cercetatori si cu cadre didactice universitare din Romania si Germania, precum si cu specialistii unor firme recunoscute pe plan mondial (IBM, CSC, msg systems s.a.), autoarea a avut nu doar sansa de a patrunde in problematica celor mai moderne tehnici de proiectare, ci si de a-si verifica si consolida ideile originale atat in ce priveste analiza, design-ul cat si metodologia conducerii de proiecte. Autoarea conduce in prezent in cadrul concernului HDI un proiect avand ca tema introducerea unui nou sistem de proiectare asistata de calculator pentru configurarea produselor, avand la baza metodologia orientata pe obiecte si o arhitectura centrata pe componente.

Experienţa practică a autoarei în domeniile de concepţie şi de realizare se regăseşte in aceasta teza în propuneri vizând arhitectura realizării pe calculator a unor operaţii specifice construcţiei asistate de calculator, precum şi îmbinarea proceselor de proiectare asistate de calculator cu cele de automatizare şi de conducerea proceselor. O parte din temele prezentate in aceasta lucrare sunt contributii originale si au fost transpuse in practica in cadrul proiectelor axate pe tema proiectarii asistate de calculator, respectiv managementul ciclului de viata

al unui produs (PLM) la care a participat autoarea in Germania, incepând din anul 1996.

Autoarea îşi exprimă recunoştinţa faţă de toate persoanele care au sprijinit-o în elaboarea tezei de doctorat, adresând în primul rând un gând plin de stimă domnului profesor Nicoară Ioan pentru profesionalitatea şi răbdarea cu care a îndrumat realizarea acestei lucrări.

2 Scopul lucrării

Scopul acestei lucrări este aplicarea tehnicilor de proiectare orientate pe obiecte in constructia aparatelor si echipamentelor urmarindu-se prin aceasta imbunatatirea calitativa a procesului de proiectare in proiecte tehnice cu aplicatii pe calculator.

Obtinerea unor solutii de o calitate superioara se urmareste pe doua planuri:

- aplicarea unei metodologii moderne de proiectare concretizata prin:
  - aplicarea conceptelor orientate pe obiecte in activitatea de analiza, design si implementare,
  - aplicarea metodei de proiectare iterativ-incrementuale utilizând technicile de notaţie oferite de Unified Modelling Language (UML),
  - realizarea unei arhitecturi flexibile, deschise, centrata pe componente,
  - utilizarea limbajelor de programare orientate pe obiecte.
- gasirea unor solutii originale, avand radacinile in modelarea orientata pe obiecte, pentru proiectarea asistata de calculator in domeniul construcţiei de maşini şi echipamente:
  - optimizarea procesului de proiectare,
  - realizarea de sisteme inteligente destinate proiectarii asistate de calculator,
  - elaborarea unei solutii generalizate cu aplicatii in gestionarea ciclului de viata al unui produs,
  - proiectarea variantelor,
  - cuplajul componentelor in cadrul unei arhitecturi flexibile,
  - elaborarea unor programe pentru proiectarea sistemelor optice.

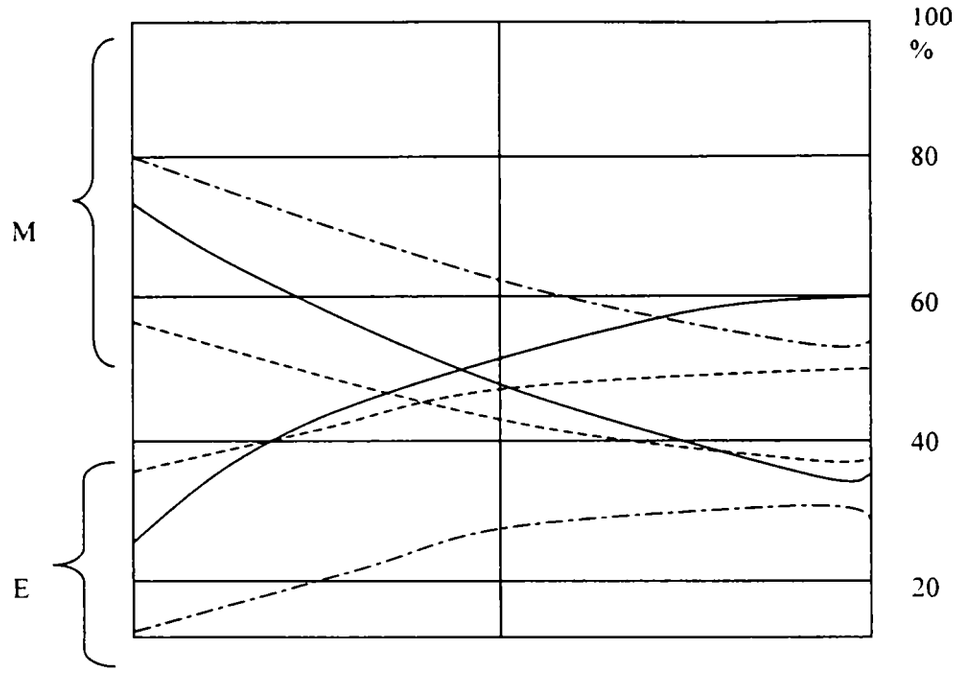
Pe parcursul lucrarii se acorda o atentie deosebita definirii unor solutii generalizate (pattern) cu un inalt potential de reutilizare.

3 Evoluţia tehnologiei informaţiilor şi efectele ei asupra întreprinderilor industriale

3.1 Tendinţe actuale

Tehnica de calcul pătrunde din ce în ce mai masiv în diferite domenii ale industriei. Tendinţa realizării unor funcţionalităţi prin soluţii electronice este clară şi conduce la reducerea soluţiilor tradiţionale neelectronice de construcţie a echipamentelor. Soluţiile tradiţionale (mecanice sau electromecanice) în cazul echipamentelor automatizate se concentrează asupra periferiei echipamentelor, vizând comunicaţia cu utilizatorii sau cu interfeţe ale diferitelor componente destinate de exemplu unor măsurări, transmiterea sau afişarea unor semnale mecanice, optice sau acustice ş. a. În această ordine de idei, când vorbim despre componentele unui sistem automatizat, trebuie acordată importanţa cuvenită componentelor destinate prelucrării de informaţii în cadrul unei arhitecturi generale.

Figura 3.1 [K1] vizualizează tendinţele actuale în ce priveşte proporţia valorică a componentelor mecanice şi electrice/ electronice pentru anumite categorii de echipamente din domeniul mecanicii fine.



Prelucrări de date pe calculator

Tehnica automatizărilor

Tehnici de măsurare mecanică

Fig.3.1

Tehnologia informaţiilor se dezvoltă în prezent în direcţia "network computing". Această direcţie de dezvoltare promovată de tehnologia internet face posibilă participarea tuturor persoanelor implicate într-o afacere în procesele operative ale unei înterprinderi.

Direcţia actuală spre care se îndreapta tehnologia informaţiei se reflectă în următoarele domenii:

- **Costuri**: competiţia puternică în cadrul pieţei şi orientarea spre necesităţile beneficiarilor forţează reducerea costurilor în toate procesele. Acesta este o parte integrantă a capacităţii de concurare şi a reacţiei la cerea de pe piată în condiţiile unui preţ acceptabil.

- **Timp**: întreprinderile trebuie să poată reacţiona într-un timp cât mai scurt la modificari în cadrul pieţei. Un sistem de prelucrare a datelor care face posibilă conducerii întreprinderii construirea rapidă şi eficientă a unor soluţii alternative este ţelul dezvoltării aplicaţiilor software.
- **Complexitate**: introducerea tehnologiilor noi, precum şi individualizarea din ce în ce mai puternică a pieţelor complică dezvoltarea de noi sisteme, care trebuie să îmbine din ce în ce mai multe reguli complexe în medii tehnice dificile.
- **Globalizare**: prin introducerea treptată a internetului în toate domeniile dispar graniţele de odinioară şi este posibilă procurarea directă a informaţiilor şi a bunurilor.
- **Importanţa cunoştinţelor despre mediu**: întreprinderile trebuie să completeze orientarea puternică asupra datelor operative cu prelucrarea informaţiilor bazată pe cunoaşterea funcţionalităţilor şi a mediului. O condiţie esenţială în acest sens este dialogul eficient între întreprinderi prin utilizarea posibilităţilor tehnice (reţele, sisteme de gestiune a documentelor ş.a.).
- **Resurse umane**: în domeniul prelucrării informaţiilor devine din ce în ce mai dificilă găsirea persoanelor cu o calificare adecvată. Este absolut necesară calificarea personalului pentru a putea aplica şi stapâni tehnologiile noi.

Inovatiile tehnice de bază se află în următoarele domenii:

- **Comunicaţii**: creşterea treptată a importanţei reţelelor, schimbările determinate de acestea în domeniul comunicaţiilor şi participarea tuturor celor implicaţi în diverse procese, conduce la modificarea scopurilor urmărite şi a structurilor sistemelor.
- **Orientarea pe obiecte**: realizarea unor aplicaţii bazate pe componente, distribuite şi independente de platforme într-un proces de imbunătăţire continuă a calităţii este sprijinită în mod adecvat prin tehnologiile orientate pe obiecte.

Aceste tehnologii fac posibilă realizarea unor sisteme cu grad ridicat de comunicare, ale căror parţi componente pot fi distribuite conform necesităţilor participanţilor la procese. Condiţia esenţială a unui astfel de sistem este

existenţa unei arhitecturi potrivite, care să garanteze capacitatea de integrare a părtilor componente pe baza unor standarde internaţionale.

3.2 Ce inseamnă "orientat pe obiect"?

Ideea "orientării pe obiecte" există de aproape 30 de ani. Limbaje de programare orientate pe obiecte există tot de aproape 30 de ani. Totuşi, abia în anii '90 apar primele publicaţii privind metodele de analiză şi design orientate pe obiecte, tehnologiile orientate pe obiecte devenind un standard pentru dezvoltarea de software în prezent. Succesul teoriei orientate pe obiect rezidă în câteva trăsături ale acesteia, oferind o serie de avataje în comparaţie cu alte metode.

În teoria orientată pe obiecte se modelează lumea reală sub formă de obiecte. Obiectele posedă anumite trăsături, au un anumit comportament şi interacţionează între ele. Un produs software este de asemenea o reprezentare a realităţii bazată pe un model. Aplicarea metodelor de analiză, design şi implementare orientate pe obiecte contribuie la ancorarea în realitate a software-ului produs, acesta fiind mai uşor de întreţinut, de modificat, de extins şi reutilizabil. Metodele de analiză şi de design uşurează comunicarea cu specialiştii dintre diferite domenii prin faptul că modelele de specialitate şi cel tehnic sunt profund ancorate în realitate.

În modelarea obiectelor reale se va renunţa adeseori la detalii neinteresante. Un obiect complex nu poate fi modelat în toată complexitatea lui într-un produs software. Din acest motiv, modelele vor fi simplificate după nevoie, descriind obiectele doar prin trăsăturile lor semnificante pentru produsul software respectiv.

În modelarea obiectelor nu se descrie fiecare obiect în parte, ci se defineşte un plan de construcţie pentru obiecte asemănătoare ca trăsături şi comportament. Un astfel de şablon se numeşte clasă. O clasă defineşte comportamentul şi atributele de obiectelor. Pentru a obţine un obiect, se creează o instanţa a unei clase. Instanţa este un exemplar definit pe baza şablonului descris de clasă, mai precis, ea va primi atributele şi comportamentul ei de la clasa instanţiata. Relaţia dintre clasă şi obiect este asemănătoare cu relaţia dintre un tip şi o entitate de tipul respectiv.

O clasă încapsulează o mulţime de proprietăţi ale obiectelor. Astfel, clasa conţine atribute. Un atribut conţine informaţii şi date ale obiectului. O altă proprietate o reprezintă operaţiile. Ele definesc comportamentul obiectului. În literatura de specialitate, operaţiile sunt numite şi servicii, metode, proceduri sau funcţii. În plus, obiectele pot defini şi anumite restricţii pe care trebuie să le indeplinească atributele (de exemplu, diferite domenii de valori pentru atribute).

Comunicaţia dintre obiecte se realizează pe bază de mesaje. Astfel, un obiect poate transmite un mesaj unui alt obiect, cu conditia ca acesta să îi fie cunoscut (să existe o relaţie între cele două obiecte). Mesajele conduc la operaţiile implementate într-o clasă. Astfel, un obiect poate înţelege numai acele mesaje, pentru care are operaţii definite.

Obiecte diferite care recepţionează acelaşi mesaj pot reacţiona în mod diferit. Aceasta este o proprietate a obiectelor (numită polimorfie) şi se bazează pe faptul că operaţiile invocate de mesajul respectiv pot avea implementări diferite în diferite clase.

Relaţiile dintre obiecte se pot clasifica astfel:

- Specializare/ generalizare

În cadrul acestei relaţii vorbim întotdeauna despre o clasă cu trăsături mai generale şi una cu trăsături speciale care se adaugă la trăsăturile generale. Obiectele clasei cu trăsături speciale pot substitui obiecte ale clasei mai generale. Vom spune că obiectele cu trăsături speciale moştenesc proprietăţile clasei cu trăsături mai generale. Clasele se pot organiza pe baza acestor relaţii în ierarhii de moştenire.

- Asociaţii

O asociaţie este o relaţie orientată, bidirecţională sau neorientată între obiectele unei sau mai multor clase.

- Agregaţii

Agregaţia este o variantă specială a unei asociaţii. Ea exprimă o relaţie de tip "întreg- parte".

3.3 Limbaje de programare orientate pe obiecte [O1]

Evoluţia limbajelor de programare este reprezentată în figura 3.2. Pătratele din figură corespund limbajelor de programare neorientate pe obiecte, ovalele corespunzând limbajelor orientate pe obiecte.

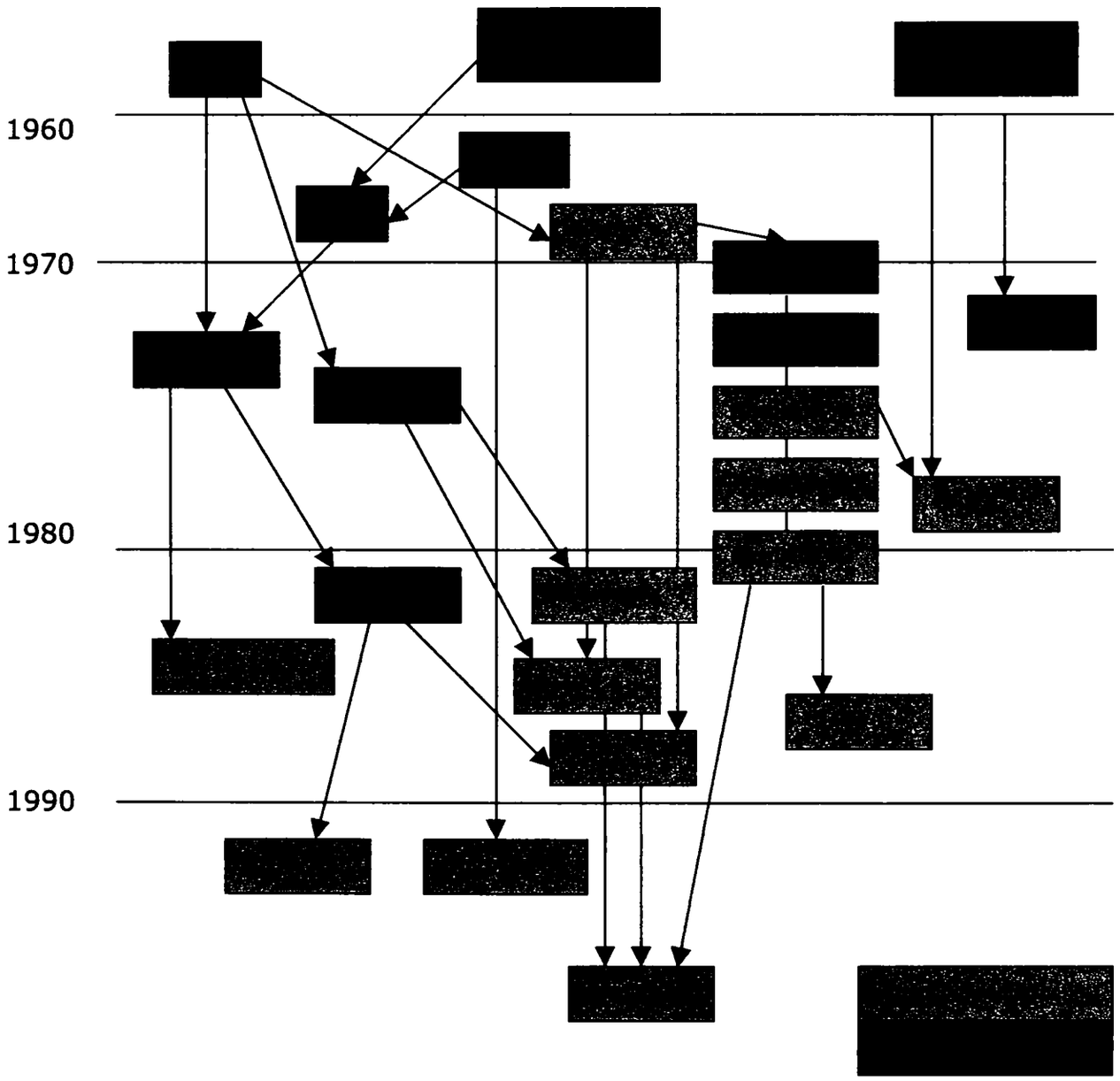


Fig.3.3

În continuare vom descrie pe scurt limbajele de programare Smalltalk şi Java, utilizate în prezent pe scară largă.

Limbajul Smalltalk a fost conceput la începutul anilor '70 în laboratoarele firmei Xerox („Palo Alto Research Center", pe scurt: PARC). Limbajul Smalltalk reprezintă cea mai riguroasă aplicare a paradigmei orientării pe obiecte. Pâna şi structurile de control sunt realizate orientat pe obiecte. Sistemul Smalltalk este format dintr-o mulţime de clase predefinite, organizate într-o bibliotecă puternică de clase. Chiar şi uneltele de programare sunt extensii ale acestei biblioteci de clase. Obiectele (clasele sunt considerate a fi obiecte, pe baza conceptului de metaclasă) există într-o aşa numită imagine (image). Imaginea realizează legătura cu sistemul de operare printr-o maşina virtuală. Acest mecanism asigură independenţa de o anumită platformă a imaginii. Aceeaşi imagine funcţionează pe toate platformele hardware pentru care există o maşină virtuală corespunzătoare.

Limbajul Java a fost conceput de către firma Sun şi a apărut pe piaţă la sfârşitul anului 1995 şi s-a impus cu mare rapiditate. Firma Sun caracterizează acest limbaj prin termenii: simplu, orientat pe obiecte, distribuit, interpretat, robust, sigur, neutru faţă de arhitectură, portabil, performant, capabil de multithred şi dinamic. Sintaxa limbajului Java se bazează pe structurile cunoscute din limbajele C, respectiv C++. O proprietate importantă a acestui limbaj este posibilitatea realizării unor aplicaţii de dimensiuni reduse, numite applets, care respectă anumite restricţii. Aplicaţiile Java se execută, de asemenea, printr-o maşina virtuală care garantează independenţa de o anumită platformă. Multe browsere pentru internet au integrată o maşină virtuală pentru Java, permiţând executarea de applets în cadrul lor.

La ora actuala se asociaza cu Java nu numai un limbaj de programare, ci si o tehnologie complexa devenita standard, cunoscuta in literatura de specialitate sub numele prescurtat J2EE (Java 2 Platform Enterprise Edition). Aceasta platforma asigura dezvoltarea de aplicatii sigure, robuste si interoperabile, punand la dispozitie infrastructura si serviciile Web necesare.

Deşi limbajul Java câştigă la ora actuală din ce în ce mai mulţi adepţi în primul rând din mediul programatorilor în C/ C++, mulţi specialişti sunt înca de părere că cel mai bun limbaj de programare orientat pe obiecte este limbajul Smalltalk.

3.4 Metode de modelare orientate pe obiecte [O1]

Într-un proiect modern, orientat pe obiecte se aplică pe toate nivelele (analiză, design, realizarea, vizualizarea şi documentarea aplicaţiilor) metode bazate pe principiul orientării pe obiecte. Aplicaţiile vor fi privite sub două aspecte:

- Sub aspectul static se va modela structura de date, respectiv de obiecte.
- Aspectul dinamic descrie interacţiunile şi comportamentul obiectelor.

Ambele aspecte contribuie la înţelegerea mai bună a problemelor de rezolvat şi reprezintă punctul de plecare în dezvoltarea aplicaţiei software.

Cele mai cunoscute metode de dezvoltare software orientate pe obiect sunt: "Object-Oriented Analysis and Design" (OOAD) de Booch, "Object Modeling Technique" (OMT) de Rumbaugh şi "Object-Oriented Software Engineering" (OOSE) de Jacobson. Din aceste metode s-au cristalizat standardele actuale: "Unified Modeling Language" (UML) de Booch ş.a. pentru notaţii, precum şi "Unified Software Development Process" (USDP) de Jacobson ş.a. pentru conducerea de proiecte. Figura 3.3 ilustrează evoluţia istorică a acestor metode.

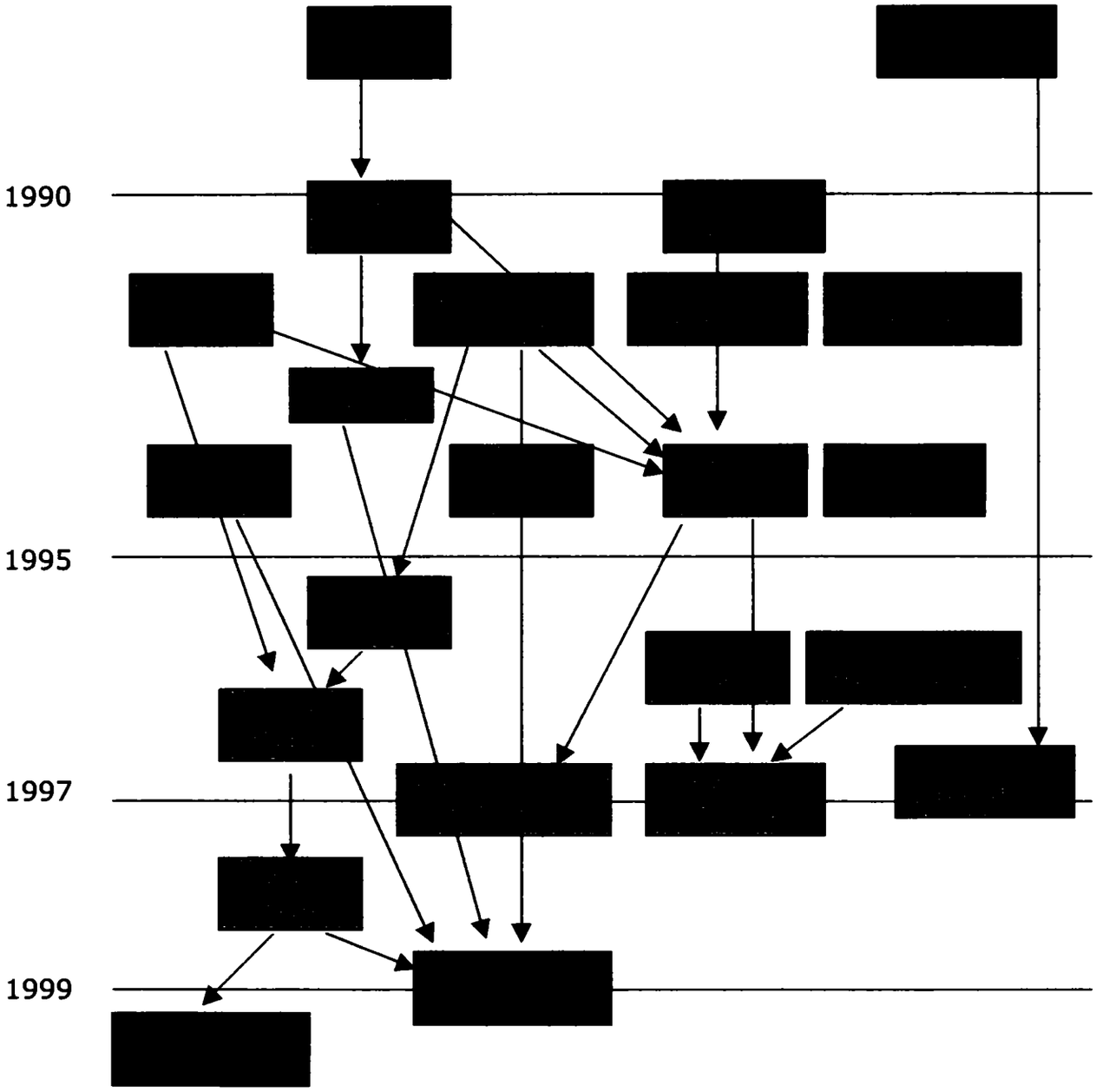


Fig.3.3

Dintre tehnologiile care se aplică în faza de realizare amintim: "ActiveX" realizată de firma Microsoft, "JavaBeans/ Enterprise JavaBeans" realizată de firma Sun, precum şi CORBA realizată de Object Management Group (OMG).

Dezvoltarea aplicaţiilor software pe baza unui model de lucru orientat pe obiecte se caracterizează prin urmatoarele proprietăţi:

- Se bazează pe cazuri de aplicaţie: pentru specificarea cerinţelor se utilizează cazuri de aplicaţie (use cases) care descriu procesele de bază din punctul de vedere al utilizatorilor.
- Centrat pe arhitectură şi componente: dezvoltarea de software ţine seama de condiţiile impuse de arhitectură.

- Iterativ: împărţirea procesului de dezvoltare a software-ului în mai mulţi paşi, având aceleaşi dimensiuni. Fiecare iteraţiune conduce la un rezultat parţial.
- Incrementual: funcţionalitatea sistemului de creat se întregeste cu fiecare pas de dezvoltare.

Procesele iterative prezintă avantaje suplimentare în urmatoarele domenii:

- Managementul riscurilor,
- Integrarea continuă,
- Strategiile de test,
- Flexibilitatea livrării,
- Planificare continuă,
- Evaluarea stadiului proiectului,
- Optimizarea procesului de implementare.

În prezent se aplică pe scară largă în realizarea proiectelor software metodele iterative-incrementuale utilizând technicile de notaţie oferite de Unified Modelling Language (UML). Figura 3.4 ilustreaza domeniile de aplicare a acestor metode.

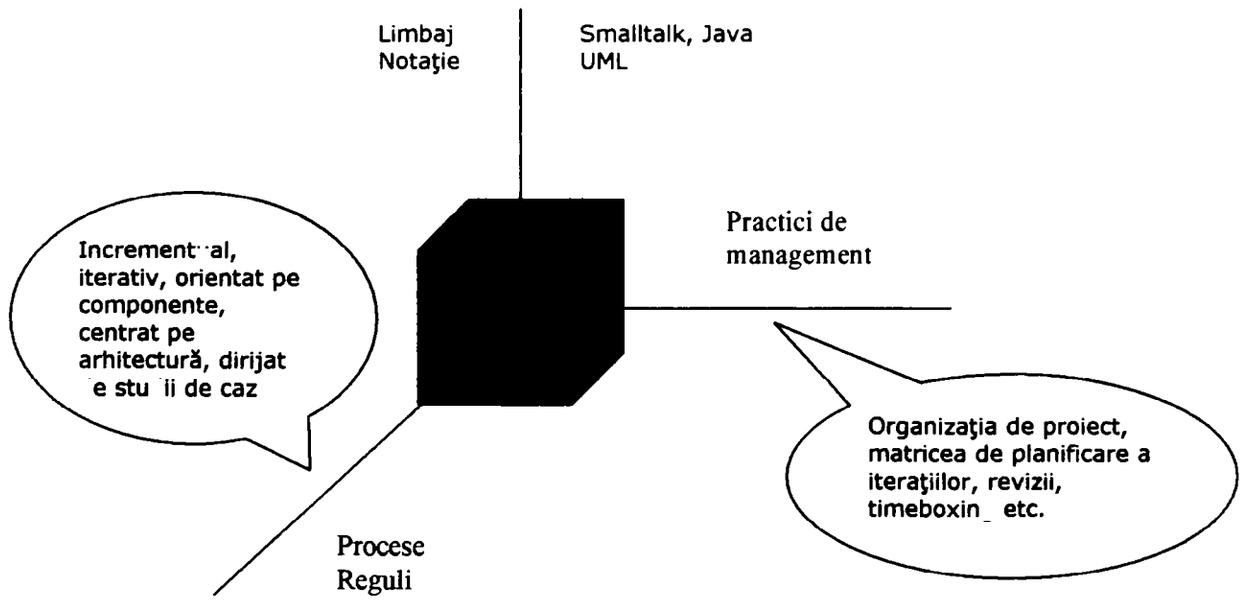


Fig.3.4.

În cele ce urmează vom prezenta limbajul de modelare "Unified Modelling Language" şi doua metode actuale de conducere a proceselor de proiectare software, şi anume: "Unified Software Development Process", denumită pe scurt

si "Unified Process", un quasistandard actual pentru procesele de dezvoltare a aplicaţiilor software, precum şi metoda eXtreme Programming.

3.5 Aplicarea unor practici din domeniul proiectării software în proiecte tehnice

În cele ce urmează ne vom referi la proiecte tehnice (de exemplu din domeniul construcţiei echipamentelor).

Complexitatea unui proiect tehnic poate rezulta din caracteristicile tehnice ale echipamentelor de realizat, precum şi din caracteristicile tehnice ale uneltelor de lucru (hardware în sens larg şi software) şi ale mediului.

Într-un astfel de proiect trebuie îmbinate diverse metode şi moduri de reprezentare a informaţiilor, a unei părţi anume din lumea reală, precum şi a unor moduri de gândire şi de lucru diferite, caracteristice specialiştilor din domenii neînrudite.

Experienţa autoarei în diverse proiecte demonstrează că o parte din practicile din domeniul proiectării software pot fi aplicate cu succes şi în proiecte tehnice.

3.6 Metoda de lucru orientată pe obiect [O1, O3]

Metodele de lucru orientate pe obiect, reprezentând standardul actual în dezvoltarea de software, se recomandă a fi utilizate în activităţi de modelare şi construcţie din domeniul construcţiilor de maşini şi echipamente, pentru a mări eficienţa de lucru în proiectare. Reprezentarea şi prelucrarea elemetelor geometrice, a elementelor din domeniul transportului de materiale sau al unui sistem de comandă pentru linii de producţie automatizate se modelează optimal îmbinând formele de modelare orientate pe obiect (modele "use-case", modele de clasă, obiect ş. a.) cu cele specifice domeniului ingineresc.

Limbajul de modelare UML se recomandă a fi extins asupra întregii activităţi de analiză şi modelare dintr-un proiect tehnic. Aceasta oferă pe de o parte avantajul existenţei unui limbaj comun de analiză şi modelare într-o echipă de proiectare formată din specialişti din diverse domenii. Pe de altă parte, există unelte de modelare care fac posibilă modelarea pe calculator pe baza limbajului UML,

precum şi importul şi exportul de informaţii către alte sisteme de prelucrare a datelor. Din modelele UML se pot genera automat părţi de program, reducând astfel, efortul de implementare.

3.7 Arhitectura [V4]

O serie de principii din domeniul arhitecturii sistemelor informatice pot fi extinse asupra unor sisteme tehnice, în special în domeniul construcţiei de maşini şi echipamente, al automatizărilor şi al sistemelor de comandă.

În domeniul mecanicii fine, stadiul actual al posibilităţilor de automatizare constă în utilizarea unor nuclee de circuite bazate pe microprocesoare, cu grad înalt de integrare, componente electronice programabile, precum şi o structură internă permiţând cuplajul şi schimbul de informaţii simple şi eficiente cu unităţile funcţionale periferice [K1].

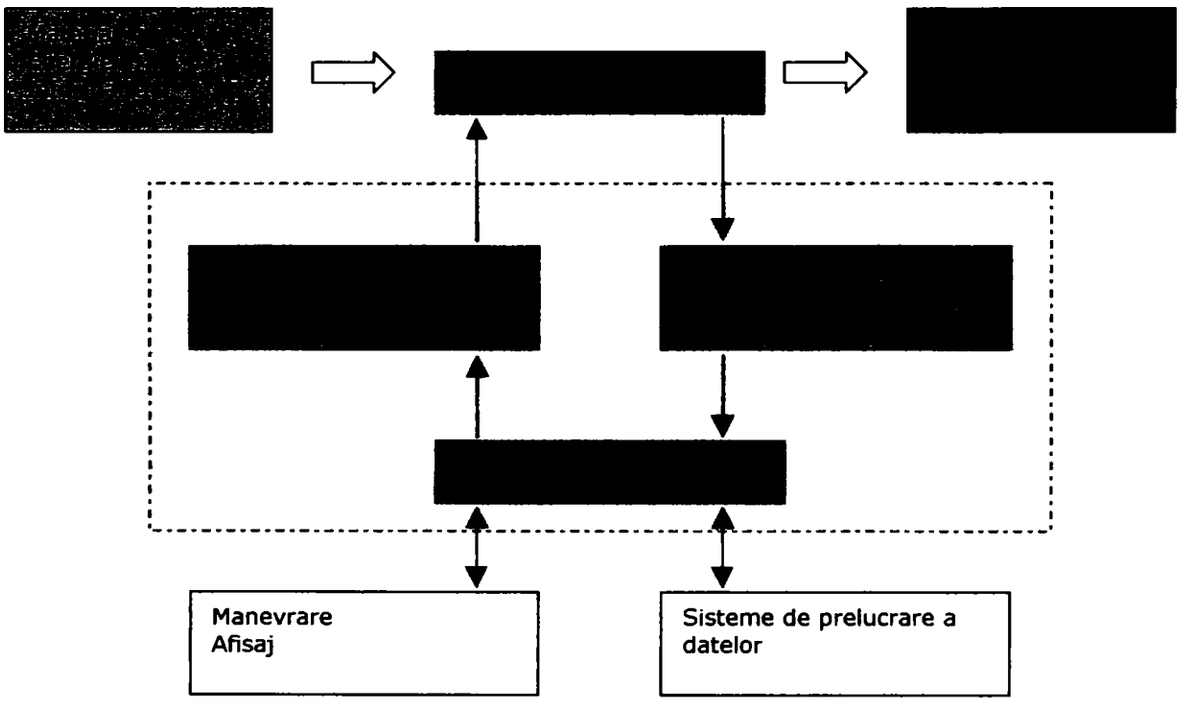


Fig.3.5

Particularităţile unei arhitecturi specifice unui sistem automatizat de prelucrare constau într-un grad mult mai înalt de heterogenitate, decât în cazul unei arhitecturi destinate numai prelucrării de informaţii. Pe lângă respectarea principiilor generale enumerate mai jos, trebuie acoradată o atenţie mai ridicată standardizării interfeţelor de comunicaţie cu dispozitive mecanice, optice, electrice, electronice ş. a. Figura 3.5 ilustreaza o astfel de structura heterogena.

3.7.1 Arhitectura sistemelor

Arhitectura unui sistem descrie structurile de bază ale sistemelor de aplicaţii care sunt în măsură să realizeze anumite cerinţe şi proprietăţi. O astfel de arhitectură (figura 3.6) include cel puţin următoarele componente:

- **Arhitectura de specialitate (business architecture),** care defineşte proprietăţile de structură ale proceselor de specialitate, a mecanismelor de dirijare, a componentelor de specialitate şi a unor servicii;
- **Arhitectura de componente,** care defineşte şi structurează elementele de construcţie hardware (echipamente) şi software, precum şi cuplajul/ decuplajul şi interacţiunile dintre componente;
- **Arhitectura de sistem,** care descrie mediul în care funcţionează elementele de construcţie hardware şi software, modul în care li se comunică datele, modul în care se realizează comunicarea tehnică dintre componente şi modificarea acestora.

Arhitectura unei aplicaţii trebuie privită ca o definiţie a proprietăţilor structurale ale soluţiei de realizat (de exemplu, un sistem de proiectare asistată de calculator sau o linie de fabricaţie automatizată pentru construcţia unor echipamente). Ea face posibilă realizarea unor sisteme asemănătoare pe baza unui plan de construcţie generalizat.

Procesul de dezvoltare a soluţiei, de la ideile de structurare şi până la sistemul funcţional se poate ghida după diferite principii de construcţie sau paradigme, mai mult sau mai puţin potrivite pentru realizarea proprietăţilor dorite. O metodă care se recomandă este construcţia orientată pe obiecte.

La principiul de construcţie ales se adaugă în procesul dezvoltării soluţiei aşa-numite principii de elaborare care precizează procedeele, metodele şi uneltele de lucru.

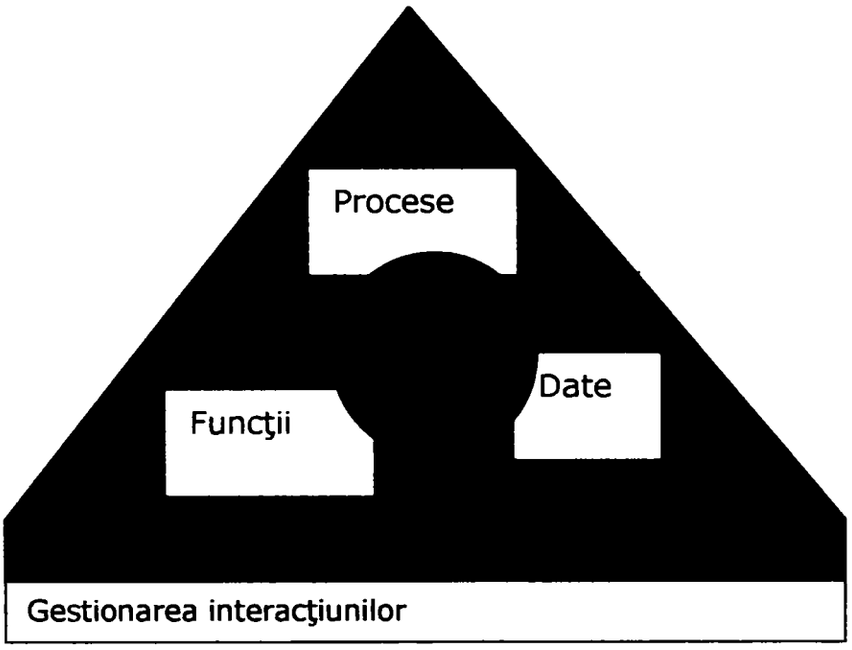


Fig.3.6

3.7.2 Cerinţele unui model de arhitectură

În condiţiile unui mediu tehnic şi organizatoric supus permanent unor transformări din ce în ce mai rapide, întreprinderile au nevoie de aplicaţii care fac faţă unui număr ridicat de cerinţe. Astfel:

- Deoarece durata de realizare a unei aplicaţii incluzând părţi de software, cât şi de hardware este îndelungată, aplicaţiile noi trebuie să se poată utiliza în medii şi constelaţii necunoscute în timpul proiectării.
- În unele cazuri, aplicaţiile noi trebuie să includă unele funcţionalităţi ale aplicaţiilor din trecut, deoarece o substituire completă a unei aplicaţii printr-una nou realizată necesită un timp considerabil (probleme de compatibilitate, migrare a datelor în noile structuri).
- Deseori apar grupuri neprevăzute de utilizatori, care trebuie integraţi în procese.
- Forme diferite de organizare şi infrastructuri trebuie să fie servite efectiv.
- Soluţii standard independente trebuie să fie integrabile rapid şi simplu în sistemele existente.
- Sisteme complexe, distribuite trebuie să poată fi bine stăpânite în ce priveşte funcţionalitatea şi administrarea lor.

Ţelul unei arhitecturi de aplicaţie este asigurarea unei flexibilităţi optime şi a unei înalte stabilităţi ale sistemelor de aplicaţii. Baza unei astfel de arhitecturi o reprezintă următoarele proprietăţi:

- Formularea potrivită a scopurilor,
- Orientarea spre procese de specialitate,
- Adaptabilitate,
- Reutilizabilitate,
- Distributivitate,
- Deschidere.

În conceperea arhitecturii trebuie ţinut cont de aceste principii şi de posibilitatea relizării economice a acestora.

3.7.3 Formularea potrivită a scopurilor

Prin formularea potrivită a scopurilor unui sistem se urmăreşte rezolvarea unor probleme în condiţii date şi cu o anumită calitate, fără a rezolva mai mult decât a fost solicitat. În cazul unei întreprinderi, aceasta se realizează prin respectarea următoarelor principii:

- Limitarea la necesităţile întreprinderii,
- Reprezentarea completă a funcţionalităţii întreprinderii,
- Gruparea neredundantă a funcţionalităţilor,
- Asigurarea atingerii unei calităţi propuse şi suficiente.

Cerinţele calitative ale unei aplicaţii operative de mari dimensiuni se pot concretiza astfel:

- Siguranţa prelucrării,
- Posibilitatea de a prelucra cantităti mari,
- Funcţionarea performantă în cazul unui număr ridicat de utilizatori,
- Posibilitatea administrării in cazul unei complexităţi ridicate,
- Robusteţe în ce priveşte erorile,
- Ergonomie şi un mod simplu de utilizare.

O arhitectură care nu ţine cont de principiile enumerate mai sus nu poate fi considerată ca acceptabilă.

3.7.4 Orientarea pe procese dintr-un anumit domeniu (business process)

Prin orientarea pe procese se înţelege cerinţa ca o aplicaţie să servească modului de lucru şi mecanismelor dintr-o întreprindere sau dintr-un domeniu de activitate.

Prin "business process" înţelegem o succesiune definită de subprocese. Aceasta va fi declanşata de un eveniment şi are ca scop efectuarea integrală a comenzii legate de evenimentul respectiv.

Orientarea arhitecturii pe prelucrarea proceselor de domeniu este o decizie esenţială de design. Ea permite elaborarea independentă în cadrul unei întreprinderi a elementelor de construcţie a soluţiei prin integrarea lor într-o ierarhie superioară de dirijare a proceselor. Orientarea pe procese aduce o noutate în ce priveste sistemele de aplicaţii, noutatea ei constând în determinarea unor părţi de proces refolosibile care se pot combina flexibil în procese noi.

Un proces este declanşat de un eveniment extern sau intern (figura 2.7).

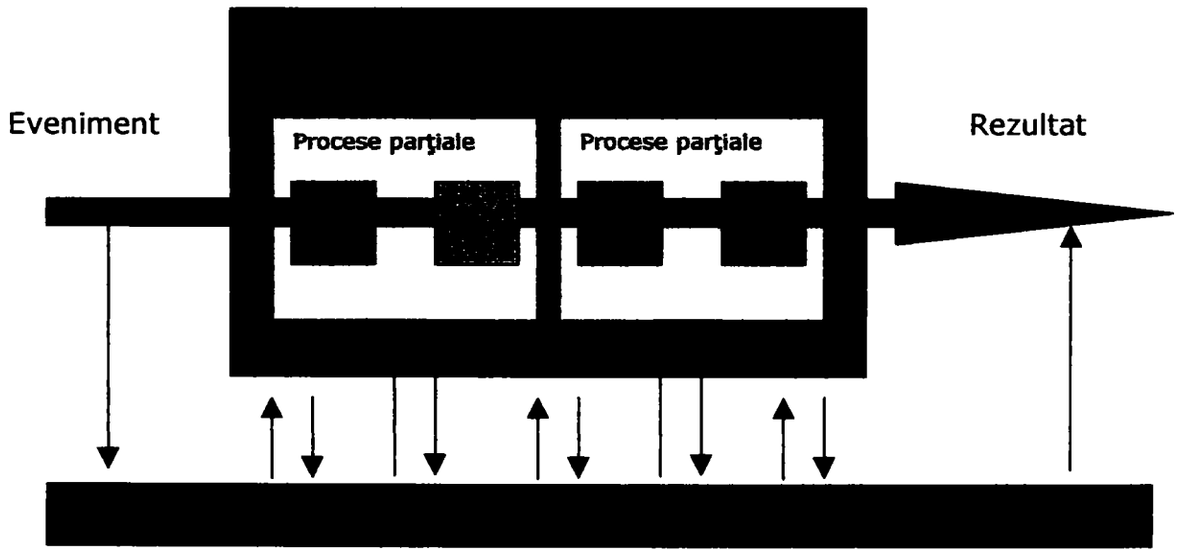


Fig.3.7

Procesul se defineşte prin descrierea proceselor sale parţiale. Pentru execuţia unui proces parţial, unitatea organizatorică implicată necesită informaţii sub

forma de date. Acestea se prelucrează de către persoane fizice sau programe care necesită informaţii şi reguli. Procesele parţiale se pot prelucra la momente de timp diferite, în locuri diferite şi de către persoane sau programe diferite.

Fiecare proces poate declanşa la rândul său alte procese.

3.7.5 Adaptabilitate

Dinamica crescândă a pieţei accentuează necesitatea unei adaptabilităţi eficiente şi rapide a aplicaţiilor.

- Aplicaţiile trebuie să poată profita de către dezvoltările în domeniile hardware-ului şi a software-ului.
- Aplicaţiile trebuie să poată transpune în viaţă cât mai repede şi mai eficient transformările funcţionale.
- Aplicaţiile trebuie să poată prelua rapid şi flexibil strategii modificate sau noi.

De aceea este necesară concentrarea atenţiei asupra adaptabilităţii în cadrul conceperii arhitecturii. Adaptabilitatea se poate defini astfel:

Prin adaptabilitate înţelegem capacitatea unui sistem de a reacţiona eficient şi cu precizie la modificarea cerinţelor care stau la baza sistemului.

Adaptabilitatea unui sistem este garantată prin următoarele proprietăţi:

- Capacitatea sistemului de a fi modificat (întreţinut): prin "întreţinere" se întelege "procesul de adaptare în dimensiuni reduse", când cerinţele care stau la baza sistemului nu s-au modificat deloc sau modificările sunt de dimensiuni reduse. Din acest motiv, procesul de adaptare se va desfăşura local, în domenii înguste ale sistemului. Exemplu: corectarea unor erori, efectuarea unor reparaţii.

In condiţiile unei arhitecturi orientate pe componente, capacitatea sistemului de a fi modificat poate avea ca scop efectuarea modificărilor într-o singură componentă, fără a fi afectată interfaţa acesteia.

- Capacitatea sistemului de a fi configurat: prin aceasta înţelegem posibilitatea de a adapta aplicaţia la un anumit mediu sau context prin luarea în consideraţie a unor parametri în timpul instalaţiei sau al exploatării. Pentru a

atinge acest scop, în procesul de realizare se iau în considerare diferite scenarii şi funcţionalităţi care ţin cont de unii parametri externi sau de activarea/ deactivarea sau adăugarea unor părti ale sistemului. Exemple: schimbarea sistemului de baze de date, schimbarea sistemelor de tranzacţii, schimbarea unor module sau ansamble de fabricaţie etc.

Capacitatea sistemului de a fi configurat se bazează pe arhitecturi structurate corespunzător şi pe modele în nivele, care permit obţinerea unor mulţimi de soluţii în locul unei soluţii unice. Mărimea mulţimii soluţiilor influenţează complexitatea şi dificultatea realizării unui sistem flexibil.

Mijloacele prin care se poate realiza configurarea sistemului sunt următoarele:

- Parametrizarea: modificarea comportamentului, al rezultatelor sau ale unei funcţiuni a sistemului fără intervenţie directă în construcţie, prin componente de comandă, variabile de program şi informaţii de dirijare memorate în sisteme care pot fi editate separat.
- Programarea generică: interpretarea semanticii transmise din exterior şi traducerea ei într-un comportament determinat sau punerea la dispoziţie a unor informaţii semantice privind comportamentul propriu.

- Capacitatea de extindere: sistemele de aplicaţii trebuie să poată fi extinse pe baza unor noi cerinţe cu noi funcţionalităţi sau chiar modificarea masivă a structurilor de bază.

În cazul unei arhitecturi orientate pe componente, capacitatea de extindere poate fi interpretată în sensul că modificările necesare atingerii scopului propus vor fi efectuate în mai multe componente, precum şi în interacţiunea dintre componente.

3.7.6 Reutilizarea

Scopul general al reutilizării este creşterea eficienţei procesului de realizare a unui sistem. Ca reutilizabile sunt considerate nu numai componente fizice, ci şi programe, concepte generale, metode (de exemplu: tipare, şabloane de program generalizate, rezultatele unor generatoare etc.).

Reutilizabilitatea în cadrul procesului de realizare a unui sistem este strâns legată de capacitatea de combinare şi separare.

- Capacitatea de combinare: prin aceasta se înţelege posibilitatea îmbinării părţilor unui sistem în diferite contexte. Capacitatea de combinare presupune unificarea interfeţelor în cadrul unei arhitecturi unice.
- Capacitatea de separare: un sistem întreg poate fi doar parţial reutilizabil. Împărtirea potrivită a unui sistem de aplicaţii în părţi componente face posibilă reutilizarea anumitor părţi ale sistemului.

Capacitatea de reutilizare a unui sistem depinde esenţial de măsura în care este realizată adaptabilitatea sistemului, depinzând de o separare corespunzătoare a funcţiunilor.

3.7.7 Distributivitate

În ultimii ani a crescut drastic numărul sistemelor hardware şi software din întreprinderi în paralel cu integrarea şi interacţiunea dintre aceste aplicaţii. Rezultatul îl reprezintă un peisaj integrator, existând ca un întreg. În paralel, dezvoltarea din domeniul hardware-ului şi a reţelelor s-a îndreptat în directia unei tehnologii client/ server heterogene, cu o infrastructură distribuită.

Prin distributivitate întelegem proprietatea unui sistem prin care componentele sale funcţionează în medii distincte atât din punct de vedere logic, cât şi fizic.

Decizia asupra proprietăţii unui sistem de a fi distribuit şi posibilităţile rezultate din această proprietate se adoptă în cursul definitivării arhitecturii de aplicaţie.

Împărţirea unui sistem în părţi componente conduce în medii în reţea la aspectul distribuţiei. Doar o împărţire adecvată poate permite realizarea distributivităţii în reţele heterogene, componentele fiind cele mai adecvate candidate pentru o distribuţie. Un aspect important al distribuţiei este partiţionarea: prin aceasta se precizează modul în care se împarte un sistem, precum şi părţile rezultante. Există diferite modele de partiţionare:

- Modelul client/ server

- Partiţionare orientată pe funcţiuni: sistemul se împarte în blocuri de funcţiuni care conţin părţi componente dedicate prezentaţiei, prelucrării şi gestionării de date.
- Partiţionare bazată pe fluxul de prelucrare: sistemul se împarte în părţi componente în funcţie de fluxul intern de prelucrare şi de deciziile necesare pentru dirijare.
- Partiţionare orientată pe încărcare: sistemul va fi astfel descompus încât sarcinile de încărcare să se poată distribui adecvat.

3.7.8 Deschidere

Sistemele deschise permit integrarea funcţiunilor lor în alte sisteme pe de o parte, precum şi înglobarea unor funcţiuni străine pe de altă parte. Aceasta trebuie realizată printr-un procedeu bine definit şi transparent. Necesitatea care decurge de aici este asigurarea interoperabilităţii. Un alt aspect important al sistemelor deschise este transparenţa arhitecturii lor tehnice şi funcţionale, a mecanismelor şi conceptelor lor şi, în mod special, o arhitectură bine definită şi standardizată a interfeţelor. Specificaţia sistemelor deschise devine astfel publică şi adaptată standardelor internaţionale.

Vom enumera în continuare câteva aspecte ale deschiderii din punctele de vedere al specialităţii, tehnicii şi al structurării:

- Integrabilitate: în general, sistemele nou realizate trebuie să permită utilizarea unor funcţiuni existente deja în peisajul integrator.
- Interoperabilitate: sisteme diferite, heterogene, cu scopuri diferite acţionează ca un tot unitar.
- Posibilitatea migrării: pe lânga definirea proprietăţilor structurale ale sistemelor deschise este necesară precizarea modului în care aplicaţiile deja existente pot fi adaptate arhitecturii şi principiilor de bază ale sistemelor deschise.

3.8 Principiile de bază ale arhitecturii de aplicaţie [V4]

Arhitectura de aplicaţie trebuie să garanteze realizarea cerinţelor formulate pentru sistemul de proiectat, în condiţiile în care dezvoltarea de software se realizează după regulile stabilite de această arhitectură.

O arhitectură de aplicaţie reprezintă definiţia proprietăţilor structurale ale sistemului de realizat. Ea face posibilă realizarea unor sisteme asemănătoare pe baza unui plan de construcţie generalizat.

Principiile de bază sunt afirmaţii elementare despre proprietăţile structurale ale aplicaţiilor. Ele definesc principii călăuzitoare şi idei. Aceste principii de bază sunt:

- Formarea de componente,
- Stabilitatea,
- Structurarea.

3.8.1 Principiul formării de componente

Complexitatea sistemelor din diferite domenii poate atinge în prezent un grad foarte ridicat. Pentru a putea stapâni această complexitate, sistemele se împart în părţi componente. Părţile componente asigură prin interacţiunea lor funcţionalităţile necesare.

Un sistem privit din exterior reprezintă o unitate funcţionala, integrala, corespunzând unui scop bine definit.

Un sistem contine de regulă mai multe părţi autonome, care cooperează în scopul atingerii unui obiectiv propus. Niciuna dintre părţile componente nu are control asupra sistemului întreg. Părţile de sistem sunt cuplate lejer şi interacţionează prin intermediul interfeţelor.

O parte de sistem este o componentă dezvoltată izolat şi care comunică prin intermediul unor interfeţe bine definite cu celelalte părţi ale sistemului.

O parte de sistem nu poate fi privită automat ca o componentă. Pentru aceasta trebuie să fie satisfacută o proprietate suplimentară, şi anume: independenţa de context.

O componentă este o parte de sistem capabilă să fie integrată în diferite contexte de aplicaţie neprevizibile, fără a-şi pierde funcţionalitatea.

Componentele sunt candidatele ideale pentru distribuţie. Totuşi, poate avea sens şi realizarea unei grupări suplimentare a componentelor după criterii ca: prezentaţie, prelucrare, control şi achiziţie de date.

În cele ce urmează, enumerăm caracteristicile principiului formării de componente:

- Decuplare: doar ceea ce se poate separa (decupla) de celelalte părti ale sistemului, se poate distribui. Criterii importante în acest sens sunt: legătura funcţională şi interfeţele înguste. Prin legătura funcţională se asigură încorporarea într-o componentă a funcţionalităţilor de specialitate strâns legate între ele. Interfeţele înguste trasează graniţele dintre componentele decuplate şi fac posibilă stăpânirea tuturor părţilor unui sistem. Legătura funcţională se poate realiza prin diferite mecanisme. De exemplu:
  - Legătura orientată pe date, în sensul tipurilor abstracte de date şi a metodelor care operează asupra acestora,
  - Legătura orientată pe funcţiuni în sensul utilizării clasice sub forma de module.
- Interacţiune: aceasta descrie necesitatea posibilităţii de utilizare a unei funcţiuni ale unei părţi de sistem decuplate, în scopul soluţionării unei probleme globale. Aceasta se realizează prin mecanisme adecvate de comunicare. De exemplu:
  - Mecanisme de apel,
  - Legături bazate pe mesaje,
  - Cozi de aşteptare pentru realizarea comunicaţiei asincrone,
  - Remote Procedure Call (RPC),
  - Request Broker.

Decuplarea fără interacţiune (sau invers) are prea puţin sens. În timp ce în sistemele clasice aceste mecanisme au putut fi separate relativ clar, în sistemele orientate pe obiecte pălesc graniţele dintre ele. De exemplu: mecanismele de moştenire şi polimorfie. Ambele mecanisme reunesc atât aspecte ale decuplării, cât şi ale interacţiunii.

Din aspectele de interacţiune şi decuplare se pot deduce alte două aspecte importante ale formării de componente. Împărţirea unui sistem în părţi componente într-un mediu în retea (în sens larg) conduce la aspectul de distribuţie. Doar o împărţire corectă poate garanta realizarea distribuţiei în reţele heterogene. Componentele reprezintă baza pentru distribuţie. Sunt necesare însă reguli de navigaţie pentru a preciza modurile de interacţiune în cazul în care pentru soluţionarea unei probleme trebuie apelat la interacţiunea dintre componente:

- Distributivitate: aceasta reprezinta o proprietate a unui sistem prin care părţi ale acestui sistem pot funcţiona în medii diferite din punct de vedere logic sau fizic.

Se poate distribui doar ceea ce se poate separa (decupla) pe baza unor criterii bine stabilite, de celelalte părţi ale sistemului.

Interacţiunea privită în legătură cu părţi de sistem care pot fi separate şi distribuite necesită neapărat mecanisme de comunicaţie, care pot depăşi graniţele unui anumit mediu de exploatare al sistemului distribuit. O condiţie esenţială în acest sens este transparenţa mediului de execuţie: o parte nu are dreptul să cunoască detalii privind locul şi mediul de execuţie ale unei alte părţi, altfel nu se poate vorbi de decuplare.

Câteva exemple din domeniul informaticii ar fi:

- Arhitecturi in 3 nivele,
- Client/ server,
- Thin clients.

Exemplu din domeniul liniilor de productie automatizate:

- Structuri de tip master/ slave.

- Navigaţie şi control: mecanismele de interacţiune asigură utilizabilitatea în principiu a părţilor de sistem şi a componentelor. Fără reguli referitoare la procesele sistemului complet, ar rămâne părţile de sistem pur şi simplu aşezate una lângă alta, fără un scop comun. Regulile rezultate din scopurile

propuse şi necesităţi funcţionale asigură conlucrarea orientată pe rezolvarea unei anumite probleme de către părţile de sistem.

Exemple:

- Procesoare de activităţi,
- Sisteme workflow.

3.8.2 Principiul stabilităţii

Realizarea de noi sisteme trebuie să ţină cont de sistemele existente şi trebuie să decurgă in paşi controlabili. Dezvoltarea treptată a noilor sisteme pe baza unei structuri de bază adecvate garantează adaptările rapide şi extinderile fără contradicţii structurale în diferite cicluri de dezvoltare.

- Reutilizare: când se vorbeşte despre reutilizare, se uită adesea că înaintea scopului reutilizării se află scopul utilizării. Reutilizarea înseamna folosirea repetată a unei funcţionalităţi a sistemului şi a conceptelor fără a efectua muncă repetată.
- Integrare: părţile de sistem existente trebuie să interacţioneze în diferite contexte pentru a oferi utilizatorului funcţionalitatea dorită, fără contradicţii structurale. Cel mai important aspect în acest sens este integrarea unor sisteme străine, dezvoltate pe baza unor principii străine, precum şi integrarea funcţionalităţii proprii în sisteme străine.
- Constanţa: o arhitectură prezintă doar atunci interes, dacă ea poate fi utilizată timp îndelungat. Aceasta implică îndeplinirea unor proprietăţi structurale care au valabilitate în viitor. Chiar dacă soluţiile tehnice sau de specialitate se modifică, mecanismele de bază care le realizează trebuie să rămână constante.
- Adaptare: o arhitectură trebuie să ofere mecanisme de bază prin care funcţionalitatea să poată fi adaptată pentru diferite medii, încărcări şi cerinţe. Exemple:
  - Mecanisme de parametrizare,
  - Procedee de scalare şi configurare,
  - Mecanisme de flexibilizare.

- Migrare: acest principiu descrie posibilităţile şi mecanismele pentru dezvoltarea structurilor existente pentru a putea fi adaptate structurilor şi cerinţelor unei anumite arhitecturi.

Principiul stabilităţii conduce la necesitatea subordonării interfeţelor şi a grupării de funcţiuni unui proces evolutiv şi a le stabiliza astfel în paşi controlabili.

3.8.3 Principiul structurării

În pofida împărţirii sistemelor în părţi, respectiv componente, complexitatea rezultată pentru sistemele de realizat este deosebit de ridicată. Aceasta se face simţită prin numărul componentelor individuale din cadrul unui sistem. Complexitatea unui astfel de sistem este diminuată prin mecanisme suplimentare. Sistemele necesită mecanisme structurate de control. Aspectele de bază ale principiului de structurare sunt organizarea pe nivele şi gruparea unor părţi de sistem sau componente în partiţii orizontale sau verticale.

- Gruparea: părţi de sistem sau componente pot fi grupate în părţi complexe de sistem. Şi în acest caz sunt valabile afirmaţiile referitoare la principiul decuplării pe baza legăturilor funcţionale maxime şi a interfeţelor minime. Toate părţile componente posedă aceleaşi drepruri. Exemple:
  - Formarea unor subsisteme de specialitate,
  - Gruparea unor adaptoare într-un nivel de acces de date.

- Organizarea pe nivele completează gruparea printr-o supra- sau subordonare. Ea ordonează fiecare grupă după un criteriu bine definit într-o dependenţă ierarhică (de exemplu, ierarhia formată din nivelele: dialog, prelucrare şi gestiunea datelor).

Nivele sunt ierarhice în sensul că elementele unui nivel pot cunoaşte doar elementele unui nivel învecinat.

Luarea în considerare a mecanismelor de structurare conduce la conceptul de "model în nivele".

Un model în nivele al unui sistem este o clasificare (divizare) a părţilor componente ale sistemului în grupe/ nivele unice, pe baza unui criteriu de clasificare, care se completează printr-o supra- sau subordonare a nivelelor respective.

Modelele în nivele conduc la o ordonare pe bază de grupare. În timp ce nivelele individuale se determină pe baza mecanismelor de decuplare, supra-/ subordonarea în cadrul modelului este determinată în primul rând de mecanismele de interacţionare.

3.8.4 Exemplu de arhitectură deschisa [H2]

Ideea arhitecturilor deschise provenită din domeniul dezvoltării de software se pretează în mod deosebit la domeniul sistemelor de comandă.

Un sistem de comandă deschis presupune definiţia unei arhitecturi de comandă modulare, flexibile, care face posibilă adaptarea comenzii la cerinţele utilizatorilor. Astfel, o comandă (complexă) se poate reconfigura de catre utilizator pe baza unor module software care fac posibilă adaugarea, înlocuirea sau eliminarea unor module de comandă.

La fel ca în teoria componentelor software, este de o mare importanţă standardizarea interfeţelor.

Iniţiativa europeană OSACA (Open System Architecture for Controls within Automation Systems) a definit standardele sistemelor de comandă deschise. La definiţia acestor standarde au contribuit firme ca: Siemens, Bosch, Daimler-Chrysler şi BMW.

Figura 3.8 ilustrează o structură deschisă după principiul OSACA.

Sistemul de comunicaţie oferă posibilitatea combinării flexibile a modulelor software, precum şi distribuţia lor pe diferite platforme hardware. Dacă două module provenind de la firme producătoare diferite, doresc să comunice între ele, comunicaţia trebuie asigurată prin intermediul unor interfeţe standardizate (API). Astfel de interfeţe sunt formulate de către OSACA şi implementate sub numele de software de platformă.

Pentru a putea combina module de provenienţă diferită nu este suficientă doar standardizarea comunicaţiei dintre ele. Trebuie definită funcţionalitatea modulelor şi specificarea detaliată a interfeţelor din punct de vedere sintactic şi semantic. Arhitectura de referinţă OSACA defineşte domeniile de funcţiuni şi interfeţele unui sistem de comandă numerică. Prin aceasta se garantează posibilitatea cooperării dintre doua module de provenienţă diferită, precum şi interschimbabilitatea a două module cu funcţionalitate şi interfaţă identice.

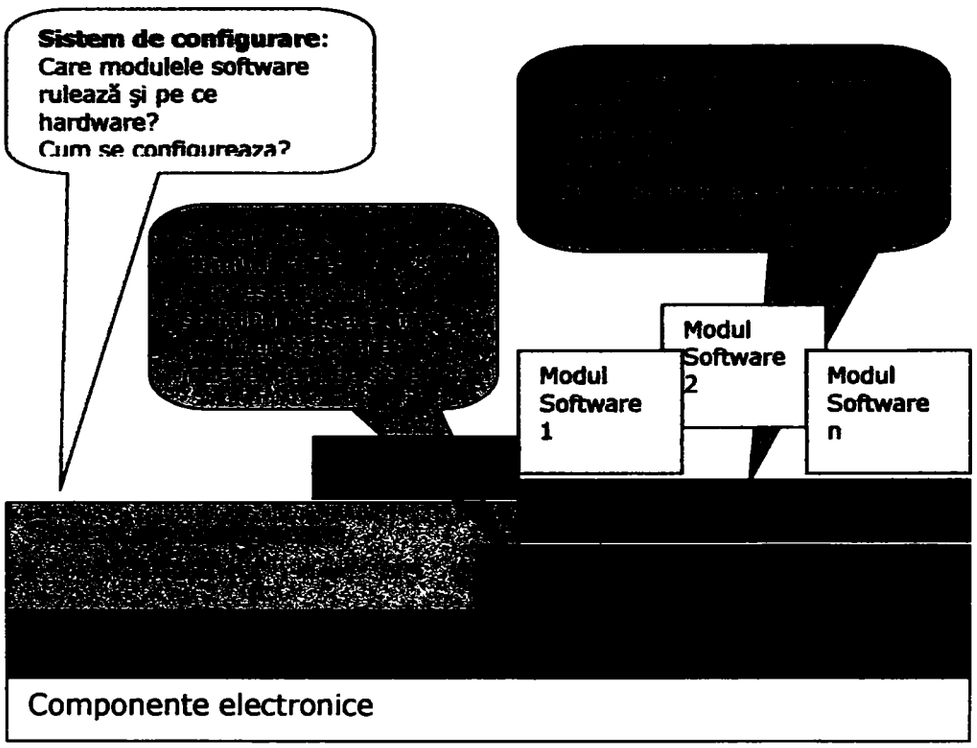


Fig.3.8

Ideea asamblării unui sistem de comandă din mai multe module, pe baza principiului "Baukasten", justifică necesitatea unui sistem de configurare. Acest sistem oferă pe de o parte posibilitatea descrierii configuraţiei dorite, pe de altă parte ofera facilităţile unui sistem confortabil de configurare care preia o parte din activităţile de rutină, uşurând munca utilizatorilor.

3.9 Arhitectura componentelor

Un sistem complex poate fi privit ca o grupare de componente (subansamble) hardware şi software, în scopul realizării unui anumit serviciu sau set de servicii într-un anumit domeniu, de exemplu construcţia de maşini, prelucrarea de date, conducerea unor procese. În literatura de specialitate există definiţii diferite

pentru termenul "componentă". Din mulţimea de definiţii rezultă câteva trâsâturi esenţiale pentru o componentă:

- Componentele încapsulează date şi procese şi îşi oferă serviciile prin intermediul unor interfeţe şi aşa-numite plugs (de exemplu, parametri, porturi, mesaje).
- Componentele se realizează pentru a fi integrate în diverse sisteme.
- Componentele se stochează într-un repository pentru a putea fi reutilizate în alte sisteme.
- Componentele vor fi elaborate utilizând metode orientate pe obiecte.
- O grupare de obiecte reprezintă o unitate ideală pentru o componentă.

Componentele software prezintă în plus particularitatea că ele se vor elabora astfel încât să întreţină relaţia dintre procesele de specialitate şi implementarea de software.

Punctul de plecare în procesul de construcţie al unui sistem complex, orientat pe componente este elaborarea uneia sau mai multor componente centrale (numite şi componente interne) specifice domeniului specific sistemului. În jurul componentelor interne se vor grupa alte componente (componente externe) care contribuie la realizarea scopului propus de către sistem.

O arhitectura bazată pe componente prezintă o serie de avantaje:

- Efect "Lego" (componentele se pot asambla într-o componentă mai complexă),
- Mai uşor de înteles (structură mai clară),
- Mai uşor de întreţinut,
- Localizarea şi corectarea defecţiunilor sau erorilor într-un spaţiu mai restrâns,
- Anumite componente dintr-un produs pot fi substituite, fără a afecta componentele rămase.

3.9.1 Componente reutilizabile

Componentele se pot combina în diferite moduri cu scopul de a obţine diferite produse. De aici rezultă că o componentă trebuie să fie reutilizabilă, ceea ce se poate realiza numai prin respectarea anumitor reguli: o componentă este reutilizabilă dacă este închisă sau dacă utilizează un singur protocol pentru

referinţele externe, valabil în toate contextele în care se poate îngloba componenta respectivă.

3.9.2 Componentă închisă

O componentă este închisă dacă utilizează numai informaţii din interior.

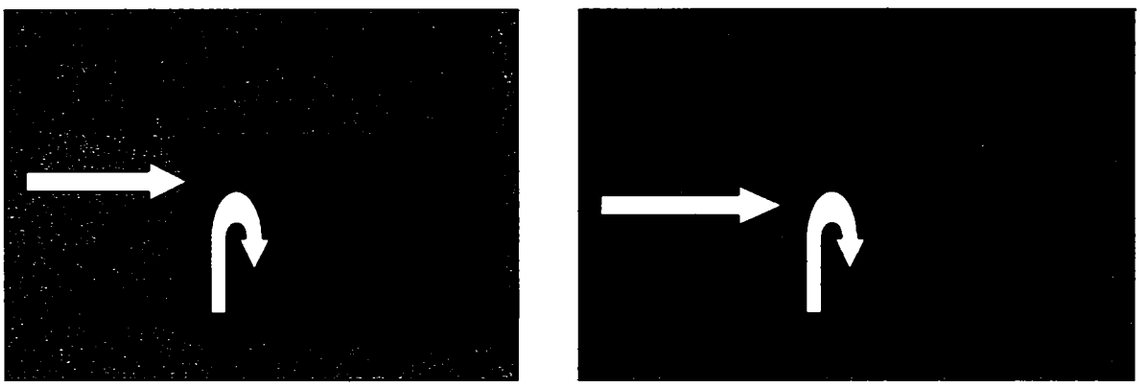


Fig.3.9

Dupa cum se poate observa din figura 3.9, componenta A este închisă. Componentele B şi C sunt şi ele închise deoarece utilizează doar atribute interne (Componenta A este inclusă atât în B, cât şi în C). Definirea unei dependenţe între componentele B şi C (de exemplu: B.e = 10% C.i) ar leza principiul de mai sus.

3.9.3 Protocol unic

Daca o componentă va fi inclusă în diferite contexte (componente) care utilizează un protocol comun, componenta este reutilizabilă şi în cazul în care ea conţine referinţe în exterior. Aceste referinţe vor utiliza numai un protocol unic, comun mai multor contexte.

După cum rezulta din figura 3.10, componenta A utilizează informaţia furnizată de context (parent).

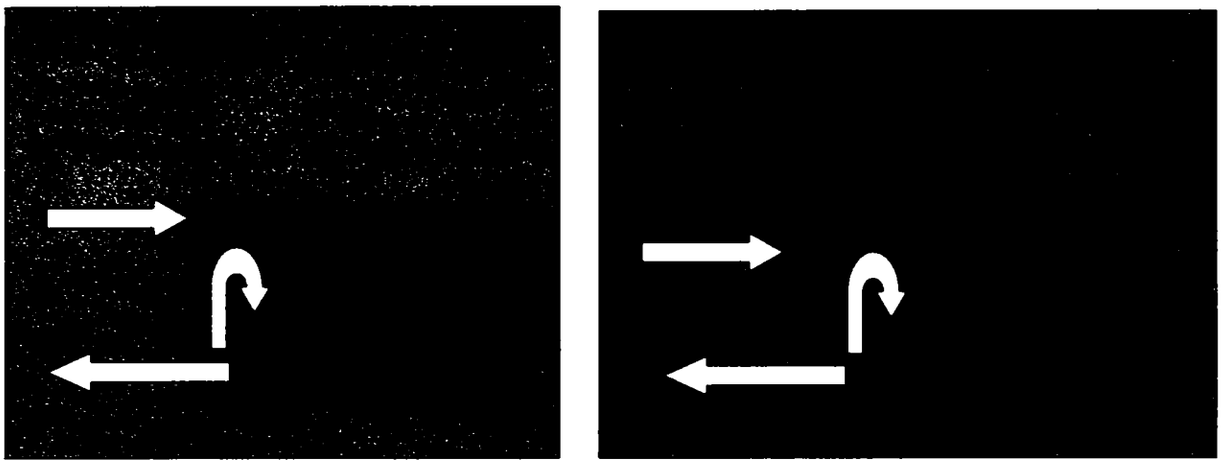


Fig.3.10

3.9.4 Modele de arhitecturi de componente

Întelegerea şi întreţinerea sistemelor complexe, bazate pe componente este imposibilă fără o descriere detaliată a structurilor interne - arhitectura componentelor. Deşi termenii "componentă", respectiv "arhitectură de componente" nu sunt standardizaţi, putem afirma că prin arhitectura de componente se înţelege un model de interdependenţă dintre componente.

3.9.5 Tipuri de componente si interdependenţe [Z1]

Daca o componentă este vazută ca "o parte separabilă de restul sistemului", ea se poate numi "componentă identificată". Componenta îşi poate păstra această identitate sau se poate transforma mai târziu într-o componentă orientată complet pe sevicii, respectiv un pachet de elemente de model.

Componentele orientate pe servicii se caracterizează prin integritate, interdependenţele dintre astfel de componente concretizându-se prin apelul unor operaţii din protocolul extern al componentelor.

Componentele care nu prezintă cele doua trăsături de mai sus, vor prezenta şi alte interdependenţe. Un exemplu în acest sens ar fi elementele de construcţie sub formă de pachet de elemente de model. În acest caz, interdependenţele se extind asupra codului sursă, elementelor grafice (bitmaps) sau a elementelor de model în cazul componentelor software, sau asupra interdependenţelor

referitoare la mediu ş. a. în cazul unor componente fizice. Şi în acest caz, cele mai interesante interdependenţe rezultă din apelul unor operaţii.

În continuare ne vom ocupa de interdependenţele pe baza apelului de operaţii.

În cazul componentelor orientate pe servicii există o separare clară între specificare şi implementare. Această trasatură permite adoptarea unei metodologii structurate pe toată durata de viaţă a elaborării componentei.

Componentele orientate pe servicii parcurg următoarele faze:

- Identificare: componenta este considerată ca fiind acceptabilă, lipseşte însă specificaţia.
- Specificare: protocolul extern şi operaţiile împreună cu toate caracteristicile tehnice ale componentei sunt complet definite, dar nerealizate.
- Realizare: caracteristicile tehnice şi întregul comportament al componentei sunt puse la dispoziţie utilizând diferite mijloace de construcţie (echipamente tehnice, un limbaj de programare, un script ş. a.).

În aceasta ordine de idei trebuie amintit că doar realizarea unei componente poate utiliza o altă componentă. Deşi se poate defini că o operaţie din protocolul extern al unei componente are la bază o operaţie din protocolul unei alte componente, pentru faza de realizare va fi necesară adoptarea unei decizii privind restricţiile asupra unei anumite operaţii. Prin restricţii se înteleg cerinţe care garantează corectitudinea operaţiei din punct de vedere tehnic şi semantic. Astfel, echipa de lucru care va realiza componenta nu va avea de ales între soluţii alternative, restricţiile de design făcând parte din arhitectură şi fiind astfel bine precizate (documentate).

Din punctul de vedere al ciclului de viaţă al unei componente, în metodica orientată pe componente se pune cel mai mare accent pe faza de realizare. Concepte pentru fazele de analiză, design, test şi întretinere sunt slab reprezentate în literatura de specialitate.

Tehnologiile utilizate în faza de realizare nu sunt însa suficiente pentru succesul dezvoltării produselor bazate pe componente. De o mare importanţă sunt

concepte şi metode orientate pe componente, care însotesc întregul ciclu de viaţă al unui produs, fiind capabile să răspundă la următoarele întrebări:

- Prin ce se distinge designul unui sistem orientat pe componente de cel al unui sistem orientat pe obiecte?
- Cum se pot genera specificaţiile pentru componente din descrierea cerinţelor de specialitate?
- Cum se poate transforma un model de obiect dintr-un domeniu într-un model de componente?

3.9.6 Identificarea componentelor

În identificarea componentelor, respectiv a subansamblelor, se recurge la informaţii conţinute în modelele de clasă şi de proces. Figura 3.11 exprimă principiul identificării componentelor. Pe baza modelelor de clasă şi a modelelor de procese existente se obţin componente, aplicând o anumita metoda de analiza.

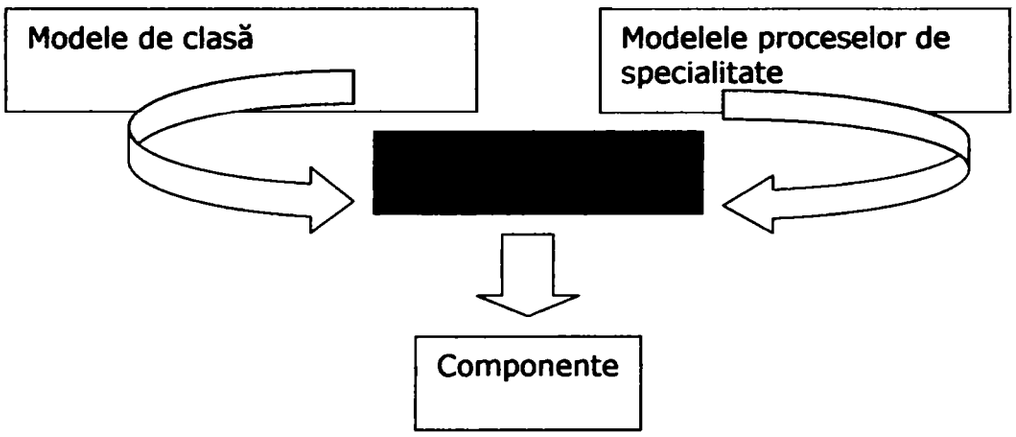


Fig.3.11

Literatura de specialitate indica in general metode de modelare care au la baza modele de clasa.

O astfel de metoda [Z1] este analiza bazată pe clustere, prin care se pot obţine componente, care corespund caracteristicilor mai sus enumerate. Această metodă se bazează pe gruparea elementelor unei mulţimi în clustere, pe baza unor proprietăţi asemănătoare. Se încearcă o astfel de grupare încât elementele unui cluster într-un anumit mediu semantic să prezinte însuşiri cât mai

asemănătoare, în timp ce elemente aparţinătoare unor clustere diferite să se asemene cât mai puţin între ele.

O alta metoda [O1] care recurge la analiza modelelor de clasa, conduce la identificarea componentelor pe baza relatiilor de specialitate dintre clase.

Experienta rezultata din proiectele realizate de autoarea lucrarii de fata demonstreaza ca accentul pe modelele de proces conduce la un rezultat calitativ superior in ce priveste identificarea componentelor. Intr-un proces iterativ, se alege o multime de candidati de componente pe baza unor legaturi de specialitate si se realizeaza diagrame se secventa care descriu comunicarea dintre candidati. Procesul se reia, optimizand de fiecare data multimea candidatilor si granularitatea componentelor. Avantajul acestei metode consta in faptul ca la sfarsitul procesului iterativ de analiza se obtin ca rezultate modelele componentelor impreuna cu interfetele acestora.

3.9.7 Modelarea structurii statice [Z1]

O arhitectură care se bazează în definiţia ei pe componente, interfeţe şi interedependenţe, descrie de fapt structura statică a sistemului.

În cazul utilizării designului "top down", în elaborarea arhitecturii de componente se pleacă de la identificarea componentelor. Interdependenţele dintre elementele de construcţie sunt definite într-o manieră generală. Se va menţiona doar că o componentă A va utiliza una sau mai multe operaţii apartinând protocolului extern al componentei B (a se observa figura 3.12).

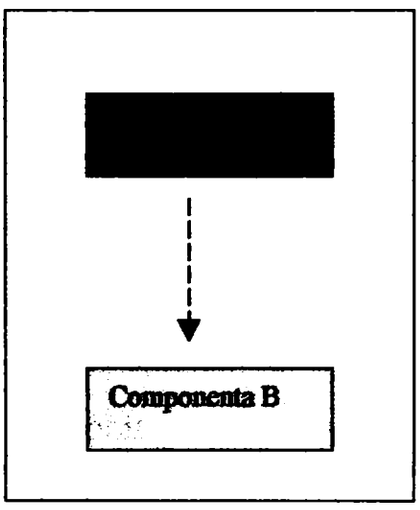


Fig.3.12

Pasul următor în descrierea exactă a interdependenţei conţine identificarea şi denumirea protocolului extern care oferă operaţiile solicitate. În cazul nostru, putem face următoarea afirmaţie: componenta A utilizează una sau mai multe operaţii aparţinând unui protocol al componentei B, având o denumire fixată (a se observa figura 3.13).

Ultimul pas în detalierea descrierii interdependenţei constă în identificarea şi denumirea apelurilor de operaţii şi a caracteristicilor tehnice (a se observa figura 3.14).

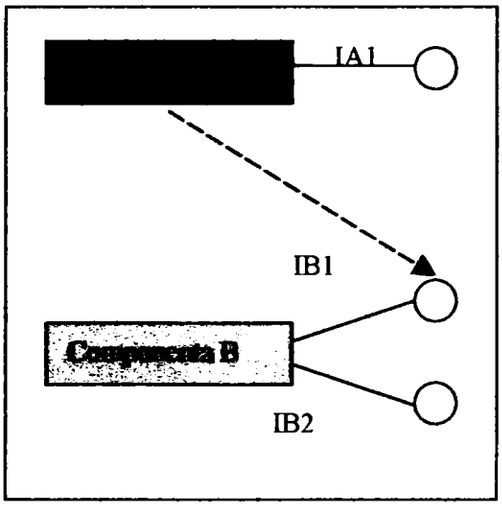


Fig.3.13

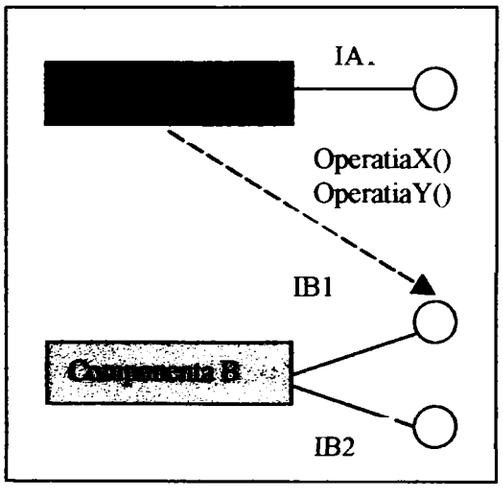


Fig.3.14

Paşii de mai sus se recomandă atât pentru etapa de specificare, cât şi pentru cea de realizare.

Etapa de specificare contribuie la modelarea restricţiilor de design pentru toate realizările posibile ale componentei. Modelul astfel obţinut se va numi arhitectura specificaţiei de componente.

Interdependenţele se utilizează în faza de realizare în modelarea deciziilor privind realizarea, cât şi a restricţiilor de design cunoscute. Componentele pot fi asamblate doar când toate caracteristicile tehnice şi interdependenţele din faza de realizare dintre toate elementele de construcţie implicate sunt definite cu un grad maxim de detaliere. Această arhitectură se numeşte arhitectura de design de componente.

Arhitecturile de componente se utilizează in următoarele domenii de activitate:

- Documentaţia deciziilor şi a restricţiilor de design,
- Descrierea şi analiza diferitelor alternative de design,
- Analiza efectelor modificărilor,
- Ajutor în alegerea şi înglobarea componentelor existente,
- Dirijarea procesului de îmbinare a componentelor.

În aplicarea acestei metodologii de construcţie bazată pe componente, designerul necesită în legatură cu fiecare componentă următoarele informaţii:

- Protocoalele externe şi operaţiile oferite de componenta respectivă (aceste informaţii fac parte din specificaţia standard a componentei)

- Repartizarea operaţiilor între diferitele protocoale (aceasta se va realiza prin implementarea componentei)

- Caracteristicile tehnice ale componentei.

În realitate, nu toate proiectele evoluează după o metodică "top down" liniară. În cazul asamblării componentelor se preferă adesea maniera "bottom up". În acest caz va trebui găsit un compromis între structura teoretică (dorită) a componentelor interdependente şi a protocoalelor, precum şi structura şi protocoalele elementelor de construcţie existente. În timpul asamblării componentelor se va putea observa, de exemplu, că mai multe componente sunt asemănătoare din punctul de vedere al caracteristicilor tehnice, oferă unul şi acelaşi protocol sau că operaţii identice aparţin unor protocoale diferite.

În ultimă instanţă, operaţia de asamblare a componentelor se reduce la adaptarea din punct de vedere al conformităţii de tipuri a caracteristicilor tehnice sau a protocoalelor existente sau dorite. O componentă dependentă de protocoale externe va funcţiona în orice mediu, în care există componente cu protocoale compatibile fără a fi necesară o identitate perfectă a componentelor şi protocoalelor. De multe ori este de ajuns un protocol care permite doar o parte din operaţiile necesare. Această interfaţă poate fi privită ca o specializare (moştenire) a interfeţei dorite.

În cazul în care lipseşte o interfaţă sau una dintre specializările ei, dar operaţiile necesare sunt oferite de alte interfeţe, se pot crea componente adaptoare. Aceste componente activează interfaţa care lipseşte prin apelul operaţiilor puse la dispoziţie de interfeţele existente.

Din cele prezentate rezultă că prezenţa unei descrieri complete a componentelor, precum şi a operaţiilor utilizate este absolut necesară pentru asamblarea de componente. Pentru reprezentarea clară şi uşor de înteles a informaţiilor despre componente s-ar putea utiliza, de exemplu, o matrice în care se vor vizualiza:

- Interdependenţele încă nespecificate între componente
- Interdependenţele rezultate din utilizarea interfeţelor
- Interdependenţe detaliate pe baza apelurilor de operaţii.

3.9.8 Modelarea comportamentului dinamic [Z1]

Modelarea structurii statice trebuie completată cu descrierea comportamentului dinamic al sistemului format din componente. Pentru modelarea comportamentului dinamic se recomandă una din modalităţile oferite de Unified Modelling Language (diagrame de colaborare, diagrame "use case", diagrame de secvenţă şi de stare).

3.9.9 Tipare şi cadre [Z1]

Un tipar (pattern) descrie o anumită problemă, frecventă în activitatea de proiectare, în contexte de design specifice şi oferă o schemă de rezolvare acceptabilă, descrisă generic. Deseori se utilizează tipare pentru a conferi sistemelor un anumit stil arhitectonic.

Un cadru (framework) defineşte un design reutilizabil şi care poate fi extins, fiind alcătuit din componente prefabricate care prin interacţiunea lor formează o aplicaţie sau o infrastructură a unei aplicaţii. Un cadru defineşte arhitectura sau infrastructura unei aplicaţii.

În literatura de specialitate, marcată de teoria orientării pe obiecte, un cadru este privit adesea ca o mulţime de clase abstracte de obiecte, care sunt legate print-un grup de interacţiuni dintre instanţele lor. Aplicarea acestei tehnologii se restrânge de multe ori la realizarea unui framework orientat pe obiect, neacoperind întregul domeniu sau întreaga infrastructură.

Deoarece cadrele oferă cea mai bună bază pentru un concept de design arhitectonic, este de recomandat, utilizarea lor la un grad mai ridicat de abstractizare.

Ca şi în cazul componentelor, putem distinge între cadre de infrastructură şi cadre de specialitate (business-frameworks). Deoarece în cazul unui cadru ne referim la o mulţime de componente integratoare, etapele dezvoltării unui cadru se aseamănă cu cele ale componentelor. În acest sens, distingem următoarele faze:

- Identificare: se precizează componentele implicate, împreună cu interacţiunile dintre ele, fără definirea formală a interfeţelor
- Specificare: se descriu toate interacţiunile şi interfeţele, fără a implementa componentele
- Realizare: se construiesc toate componentele implicate.

Pentru a asigura succesul asamblării unui sistem bazat pe componente, este importantă cunoaşterea arhitecturii care stă la baza sistemului. Arhitecturile de componente fac parte din dezvoltarea de sisteme bazate pe componente. Totuşi, izolate, nu prezintă nici un interes. Succesul realizării unui sistem depinde în aceeaşi măsură şi de alte arhitecturi, cum ar fi: arhitectura de aplicaţie şi cea de sistem.

3.9.10 Platforme de componente [Z1]

O platformă de componente reprezintă un mediu standardizat în care funcţionează componentele şi prezintă o importanţă deosebită în alegerea unei tehnologii adecvate produsului care urmează a fi construit. O astfel de platformă integrează diferite servicii care permit eliminarea unor funcţiuni standard din caracteristicile produsului (de exemplu, dacă platforma conţine deja un dispozitiv de răcire, se poate renunţa la un astfel de dispozitiv în modelarea produsului).

3.10 Utilizarea tehnicii de calcul în proiectarea şi construcţia de echipamente [K1, H2]

Tehnica de calcul cunoaşte o paletă largă de utilizare în proiectarea şi construcţia de echipamente. În vocabularul actual există câţiva termeni (prescurtări) care oglindesc diferitele domenii de aplicare a tehnicii de calcul, şi anume:

- CAD (Computer Aided Design): utilizarea tehnicii de calcul în procesele de construcţie şi de pregătire a execuţiei, rezolvând probleme de prelucrări de date geometrice, calcule, reprezentare şi realizare de liste de piese componente, planuri de lucru, informaţii pentru comenzi numerice şi oferte,
- CAE (Computer Aided Engineering): utilizarea tehnicii de calcul în procese specifice întregii infrastructuri,
- CAM (Computer Aided Manufacturing): utilizarea tehnicii de calcul în planificarea producţiei
- CAQ (Computer Aided Quality Assurance): utilizarea tehnicii de calcul în asigurarea calităţii
- CIM (Computer Integrated Manufacturing): integrarea tehnicii de calcul în procesul de execuţie.

Utilizarea sistemelor CAD în mecanica fină se caracterizează prin următoarele trăsături:

- Utilizarea şi integrarea într-un aparat a unor elemente cu proprietăţi fizice diferite (mecanice, optice, electronice, electromecanice şi optoelectronice)
- Profitarea de anumite principii până la atingerea graniţelor fizice
- Inovaţii rapide în domeniul soluţiilor tehnice (înlocuirea soluţiilor mecanice prin cele electronice, a celor electronice prin cele optice, a celor de tip analog cu cele de tip digital, a hardware-ului prin software)

- Automatizarea numeroaselor funcţii interne specifice echipamentelor
- Atingerea unei înalte precizii prin structurarea specială a produselor
- Mare varietate de variante constructive şi materiale
- Mare varietate de procedee tehnologice.

Din proprietăţile de mai sus se poate deduce că gradul de repetiţie a unor soluţii constructive este destul de scăzut şi se conturează clar cerinţa utilizării unui sistem CAD flexibil care permite formalizarea unor activităţi de rutină. Domeniile de bază pentru aplicarea sistemelor CAD sunt cele de concepţie/ design şi calculaţie pentru grupe de construcţie şi elemente. Se conturează tendinţa unei modelări complete a unui produs de la principiul constructiv până la realizarea propriu-zisă.

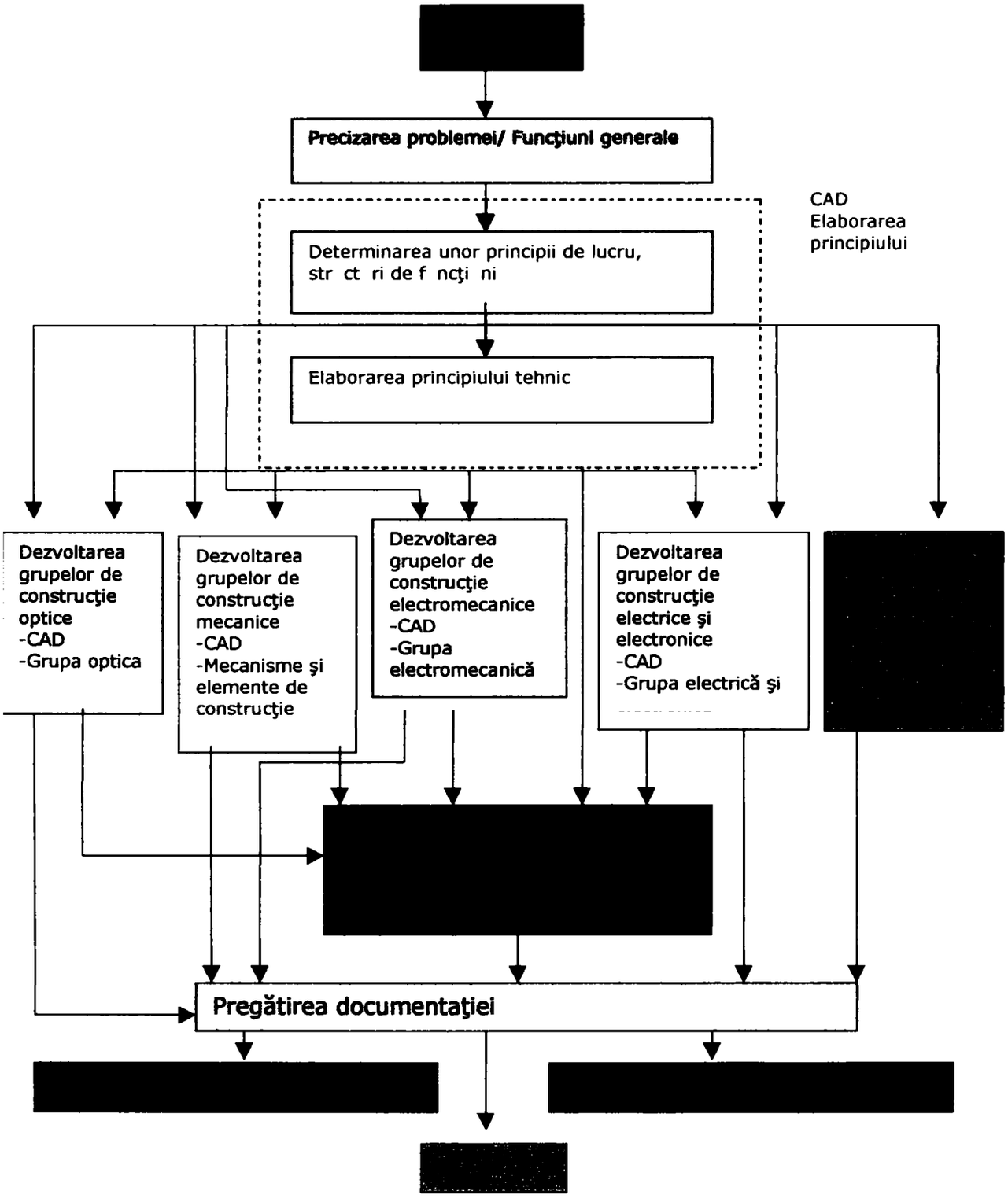


Fig.3.15

Figura 3.15 ilustrează utilizarea sistemelor CAD în procesele de construcţie.

3.10.1 Structura unui sistem CAD

Un sistem CAD realizează prelucrarea informaţiilor în scopul rezolvării unei probleme de construcţie şi cuprinde resurse hardware, software, precum şi resurse umane.

Figura 3.16 ilustrează următoarele aspecte referitoare la utilizarea unui sistem CAD:

- Dialogul om-maşină (alfanumeric şi grafic)
- Reprezentarea internă a modelului tehnic
- Construirea modelelor asociate orientate pe funcţiuni
- Cuplajul dintre diferite activităţi de construcţie (calcule, optimizări, reprezentări, informaţii, aprecieri, modificări şi documentări)
- Utilizarea bazelor de date.

În cazul în care sistemul se cuplează cu procese de planificarea şi dirijarea producţiei, se poate vorbi de un sistem CAD/ CAM sau CIM.

În procesul de construcţie sunt implicate calculatoare de diferite ordini de mărime, care corespund cerinţelor privind dialogul, posibilităţile grafice şi de disponibilitate. Locurile de muncă CAD pot funcţiona ca sisteme de-sine-stătătoare (stand-alone) sau cuplate cu alte calculatoare din reţele cu configuraţii diferite (LAN, WAN). Interfeţele fac posibilă realizarea configuraţiilor diferite şi adaptarea hardware-ului la cerinţele problemelor de rezolvat în domeniul construcţiei.

Tabelul 3.1 conţine resursele hardware implicate în construcţia asistată de calculator.

Resursele software necesare unui sistem CAD cuprind toate programele necesare rezolvării unei probleme cu ajutorul calculatorului, de la software-ul de sistem şi până la software-ul de aplicaţie.

Tabelul 3.2 ilustrează gruparea resurselor software.

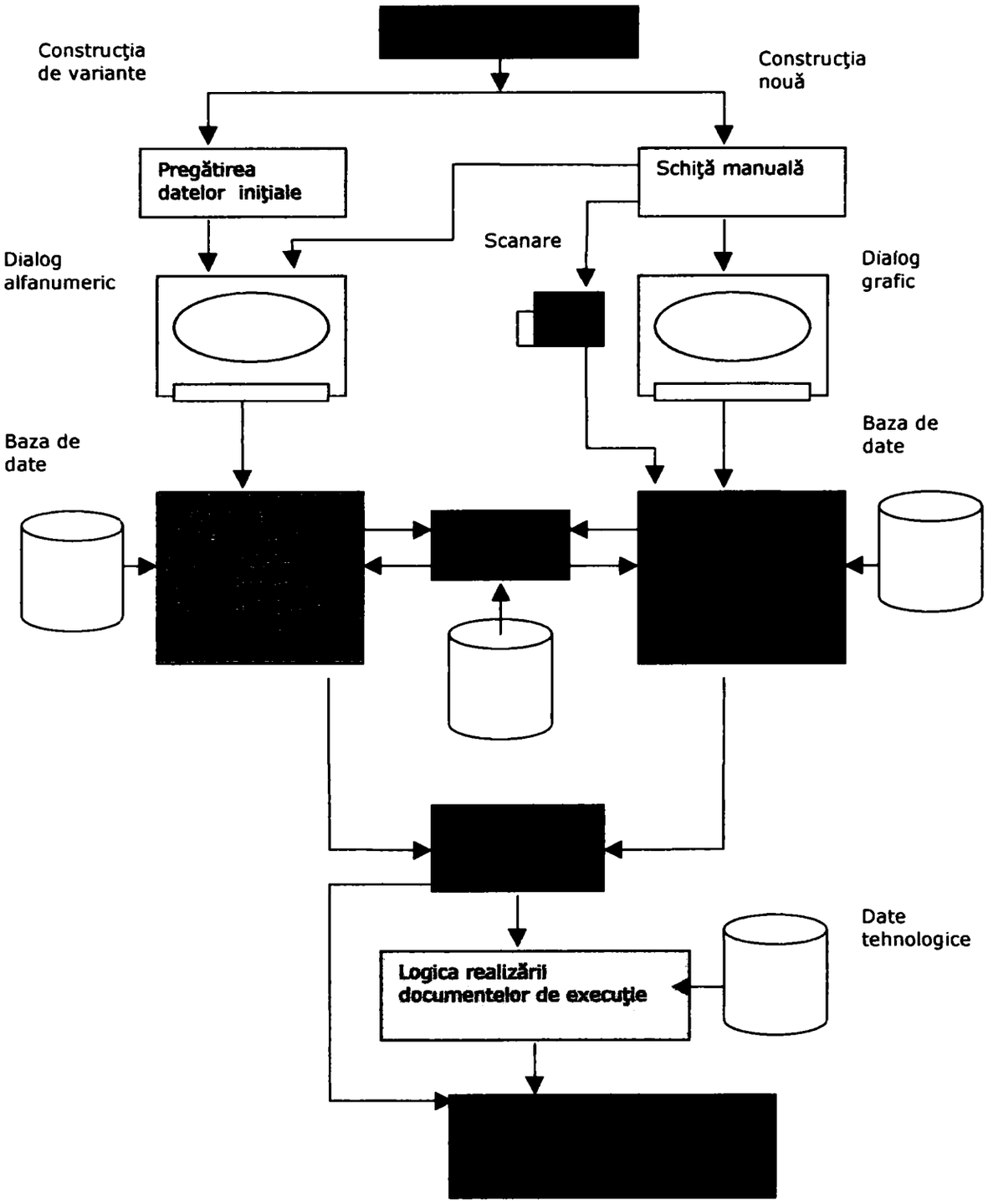

Construcţia
de variante
Construcţia
nouă
Pregătirea
datelor iniţiale
Schiţă manuală
Scanare
Dialog
alfanumeric
Dialog
grafic
Baza de
date
Baza de
date
Date
tehnologice
Logica realizării
documentelor de execuţie

Fig. 3.16

Tabelul 3.1

Tipul calculatorului	Calculator personal (PC)	Workstation (WS)	Mainframe
Procesor	80486 Pentium Pentium2	VAX, RISC, SPARC Sisteme multiprocesor	Sisteme multiprocesor
Lungimea cuvântului (Bit)	32 32 32-64	16-64	>=64
Memoria internă	16-64MB 16-196MB 32-512MB	64MB - 1 GB	> 1GB
Harddisc (Bytes)	120M - 4G - 34G	1,3 - 47G	> 16 G
Monitor grafic	800x600 - 1280x1024	1024x864 - 1664x1248	Terminale până la 16000x 16000
Sistem de operare	MS-DOS, OS/2, UNIX, Windows 95/98, Windows NT	VMS, UNIX, Windows NT	BS2000, UNIX
Aplicaţii	• Construcţie 2D • Construcţie 3D simplă • Dimensionare • Metoda elementelor finite pentru obiecte simple • Gestionare de fişiere • Liste de piese • Planificarea execuţiei • Multimedia	• Construcţii 2D/ 3D • Reprezentări conforme realităţii • Simularea dinamicii şi animaţie • Metoda elementelor finite pentru obiecte complexe • Multimedia • Realitate virtuală	• Gestionare a centralizată a datelor • Sisteme CIM complexe • Simularea dinamicii şi animaţia sistemelor complexe • Realitate virtuală

Tabelul 3.2

Software			
Software de sistem		**Software de aplicaţie**	
Sistem de operare	Servicii	Sisteme de programe comerciale	Sisteme de programe speciale
• Controlul operaţiilor de intrare/ ieşire • Jobmanager • Sistem de acces asupra fişierelor • Gestionarea timpului • Gestionarea resurselor • Gestionarea erorilor • Sistem de exploatare a reţelei • Gestionarea drepturilor de acces • ...	• translatoare (asambloare, compilatoare, inerpretoare) • limbaje de programare de nivel înalt (Smalltalk, Java, Prolog, Pascal, C,...) • Editoare • Debugger • Programe de dialog	• Matematică numerică • Statistică • Simulare, optimizare • Metoda elementului finit • Prelucrarea de texte • Baze de date • CAD (2D, 3D) • CAM, CAQ • PPS • Animaţie • Shell pentru sisteme expert	• Programe orientate pe metode (combinaţie, variaţie, analiză, diagnoza erorilor, ...) • Programe orientate pe obiecte (construcţia specială de variante, dimensionare, simulare)

Software-ul de sistem asigură operaţiile de bază pentru prelucrarea informaţiilor.

În ce priveşte software-ul de aplicaţie, o importanţă deosebită revine componentelor grafice în aprecierea performanţei sistemului CAD. Problema principală în prelucrarea informaţiilor grafice constă în necesitatea transformării informaţiilor de tip analog în informaţii digitale spre a fi prelucrate, precum şi în transformarea în sens invers, de la rezultatele prelucrării datelor de către calculator spre vizualizarea sub formă de imagine. Figura 3.17 ilustrează procesul descris.

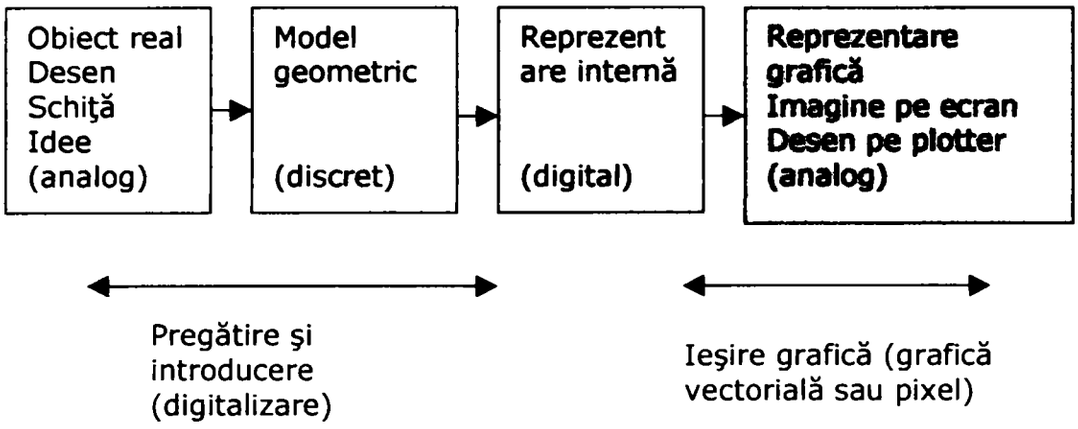


Fig. 3.17

Un alt aspect important în aprecierea performanţei software-ului CAD îl reprezintă posibilitatea prelucrării grupate a unor procese pentru a mări randamentul construcţiei asistate de calculator. În acest sens, se recomandă concentrarea atenţiei asupra unui obiect de construcţie sau a unei grupe de produse de acelaşi tip, care au la bază modele şi algoritmi de construcţie identici. O altă cale de extindere a posibilităţilor unui sistem CAD ar fi îmbinarea cunoştinţelor inginereşti şi a soluţiilor CAD cu procedee specifice inteligenţei artificiale, obţinând astfel sisteme experte.

3.10.2 Operaţii fundamentale în cadrul unui sistem CAD

În procesul de construcţie apar probleme pentru a căror rezolvare trebuie aplicate anumite operaţii specifice sistemelor CAD sau chiar sisteme speciale de programe. Astfel de operaţii sunt:

- Calcule
- Desenare
- Construcţie
- Simulare
- Sinteză.

Calculele se pot grupa în următoarele categorii:

- Calcule ulterioare: calculul unor anumiţi parametri pentru o soluţie constructivă cu structura calitativă şi cantitativă în prealabil stabilită. Calculele ulterioare servesc la verificarea cerinţelor funcţionale şi de altă natură.
- Calcule de structură: calculul unor parametri structurali pentru o structură calitativ determinată pe baza unor parametri funcţionali (sinteza mărimilor).

- Optimizare: determinarea parametrilor de structură a unei soluţii astfel, încât un anumit scop funcţional să poată fi îndeplinit în condiţii restrictive.
- Simulare: reprezentarea comportamentului unei structuri create, folosind un sistem de substitut (model).

Programele de desenare ale sistemelor CAD servesc reprezentării normate a unor piese sau ansamble sub forma unor desene, diagrame, schiţe de principiu, imagini bloc, reprezentări în perspectivă ş. a.

Operatiile de construcţie sunt următoarele:

- Constructie nouă, caracterizată prin:
  - Structura necunoscută
  - Descrierea problemei adesea foarte generală
  - Lipsesc propunerile de soluţionare

- Construcţie de adaptare, caracterizată prin:
  - Structura cunoscută
  - Problema de rezolvat se referă la modificarea sau extinderea soluţiei existente
  - Există exemple

- Construcţia de variante, caracterizată prin:
  - Pentru probleme de construcţie avind o repetabilitate ridicată există un principiu al soluţiei sau o soluţie standard
  - Problema de rezolvat conţine toate datele necesare unei construcţii concrete.

Metoda simulării reprezintă cercetarea comportamentului unui obiect prin experimente bazate pe un anumit model, utilizând sisteme de simulare. Acestea se pot clasifica în:

- Sisteme de simulare cu caracter general, conţinând blocuri destinate unor funcţiuni matematice de bază, generatoare de semnale ş. a.
- Sisteme de simulare orientate pe domenii de specialitate, conţinând elemente speciale de construcţie.

În domeniul mecanicii fine, obiectele posedă de cele mai multe ori aspecte specifice domeniilor: electronică, magnetism, caldură, mecanică, frecare ş. a.

În cazul sintezei unei structuri dorite, calculatorul trebuie să efectueze următoarele operaţii:

- Furnizarea elementelor de construcţie (sau de sinteză)
- Legarea elementelor de sinteză sub formă de structură
- Evaluarea soluţiei şi alegerea alternativei optime.

În procesele de sinteză se aplică următoarele metode:

- Formarea de lanţuri şi reţele: programul îmbină elemente ale structurii de elaborat într-o structură sub formă de lanţ sau reţea.
- Combinarea asistată de calculator: elementele de construcţie a soluţiei se grupează sub forma unor tabele de combinaţii sau grafuri (noduri "ŞI" în grafuri) spre a fi legate conjunctiv (noduri "ŞI" în grafuri) pentru a forma nouă structură. Grupele de elemente de constructie se includ disjunctiv (noduri "SAU" în grafuri) în structura cea nouă. Mulţimea de soluţii posibile se poate reprezenta tabelar sau sub forma unor grafuri.
- Variaţia asistată de calculator: o structură dată se modifică pentru a obţine noi proprietăţi în următoarele domenii:
  - realizarea unei funcţii identice sau
  - realizarea unei noi funcţii.

Determinarea unor soluţii constructive cu ajutorul calculatorului se află în prezent în dezvoltarea ei într-o fază de început. Pentru unele operaţii sau metode încă nu există sisteme de program, care să le integreze. Experienţa de până acum demonstrează că şi fazele creative ale căutarii de principii şi a concepţiei trebuie să recurgă la utilizarea mai intensă a calculatoarelor pentru a imbunătăţi eficienţa şi calitatea proceselor.

3.10.3 Soluţii CAD bazate pe obiect

În mecanica fina se utilizează cu succes sisteme integrate destinate unor elemente precizate sau unor grupe de construcţie (vezi figura ...). Produsele complexe pot fi abordate doar în cadrul grupei din care fac parte sau se proiecteaza pe baza unor soluţii pregătite în prealabil. Un astfel de sistem

integrat îmbină componentele: CAE, CAD şi CAM, fiind condiţionat de posibilitatea descrierii normate a soluţiei constructive şi existenţa unei fabricaţii automatizate şi flexibile. În cazul în care pentru o grupă de elemente de construcţie se cere realizarea optimă a unei anumite funcţionalităţi, sistemul CAD va determina automat proprietăţile (magnetice, electrice, mecanice, geometrice, termice ş. a.) ale produsului de realizat.

Figura 3.18 ilustrează principiul bazat pe obiect pentru un sistem de proiectare de plăci conductoare. Din planul de comutare şi sortimentul elementelor de construcţie se determină printr-un efort minimal de dialog structura plăcii conductoare. Această structură trebuie să corespundă unei construcţii normate. Presupunând existenţa unei linii de fabricaţie automatizate şi flexibile, sistemul CAD va furniza datele necesare pregătirii şi dirijării producţiei.

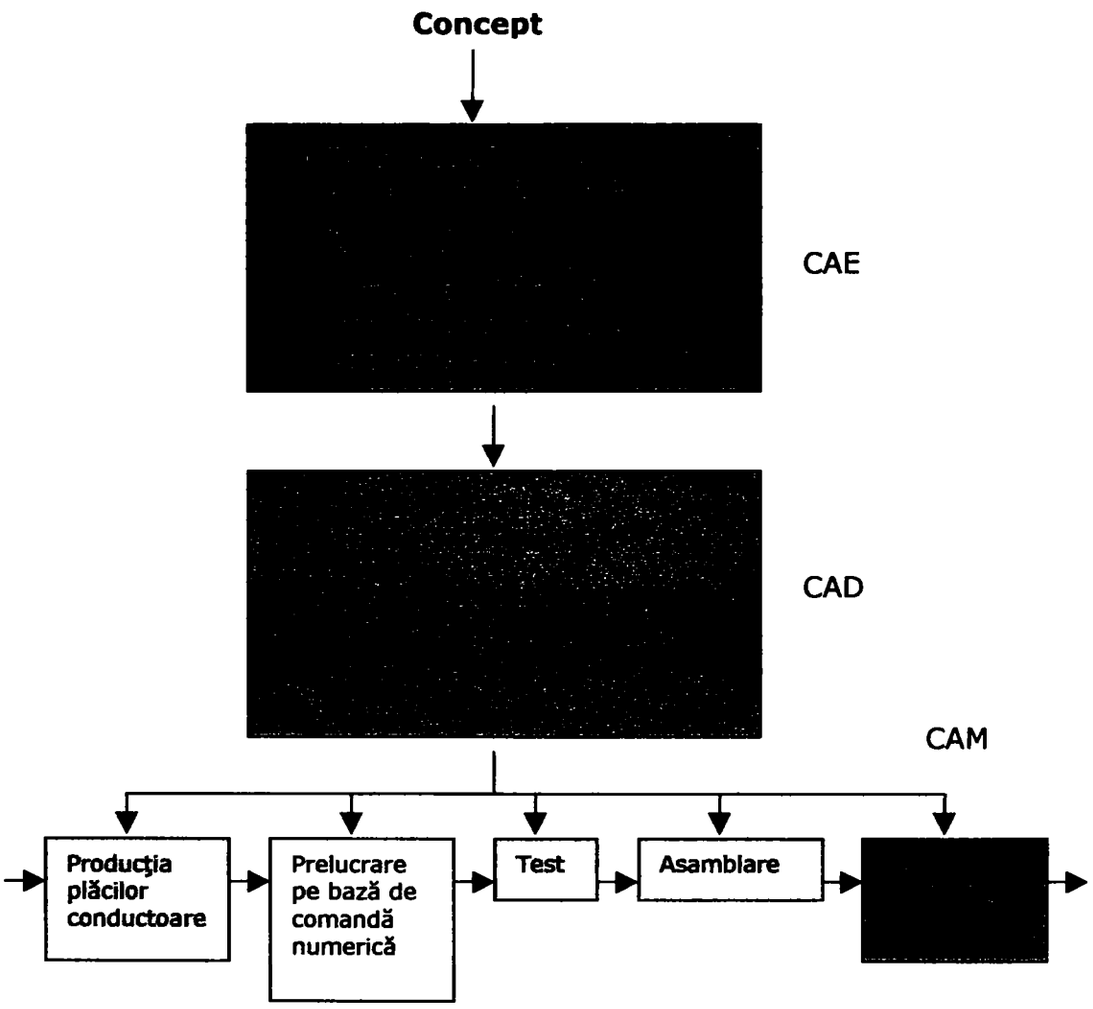


Fig. 3.18

3.10.4 Sisteme expert

Un sistem expert este un sistem de construcţie asistată de calculator care integrează cunoştinţele de specialitate dintr-un anumit domeniu şi furnizează soluţii în cadrul acestui domeniu. Prin îmbinarea cunoştinţelor cu strategiile de rezolvare a problemelor, sistemele bazate pe cunoştinţe ating un nivel superior sistemelor CAD bazate pe date şi algoritmi prin flexibilitatea lor şi printr-o nouă dimensiune calitativă a dialogului om-maşina în procesul de rezolvare a unei probleme.

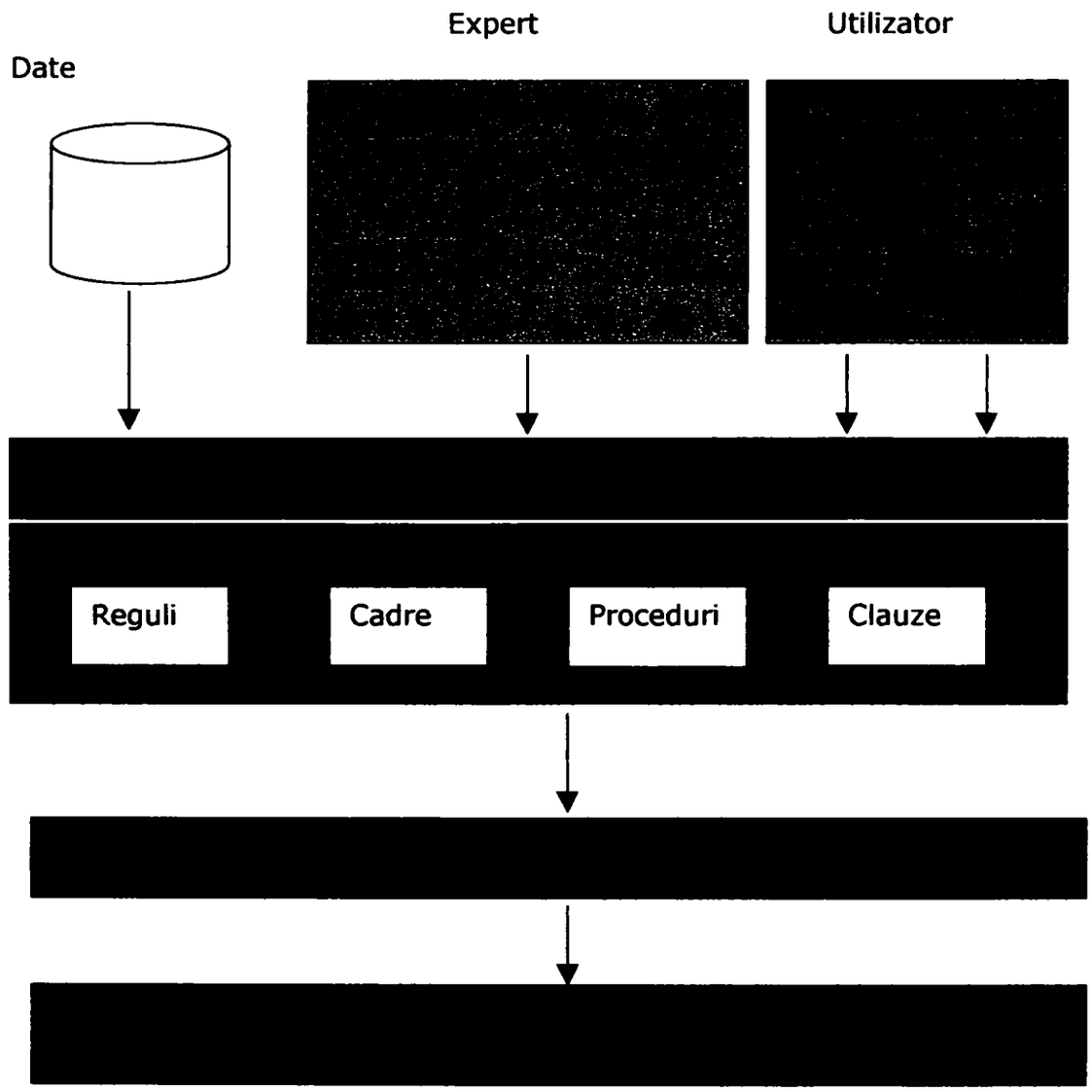


Fig. 3.19

Figura 3.19 ilustrează structura unui sistem expert.

Componentele unui sistem expert sunt componente software având următoarele proprietăţi:

- Baza de cunoştinţe conţine componente pentru soluţionarea problemelor sub forma unor cunoştinţe procedurale (reguli, proceduri, clauze), precum şi cunoştinţe declarative (fapte, explicaţii, indicaţii sub formă de cadre), care pot fi independente de problemă (statice) sau dependente de problemă (dinamice).
- Componentele destinate achiziţiei de cunoştinţe contribuie la extinderea informaţiilor memorate şi a modalităţilor de determinare a soluţiilor.
- Componenta de inferenţă asigură prelucrarea cunoştinţelor pe baza modulelor de decizie şi concepţie. Această componentă lucrează pe de o parte cu relaţii logice, pe de altă parte poate furniza şi rezultate determinate cu un anumit grad de certitudine. Relaţia DACĂ <Condiţie> ATUNCI <Acţiune, Deducţie> stă la baza determinării soluţiei.
- Componenta de dialog contribuie la comunicarea efectivă cu utilizatorul. Pentru un schimb direct de date dintre mai multe sisteme este necesară o inerfaţă.

Sistemele expert se aplică cu succes în industrie în domeniile:

- Conducerea unor procese tehnologice complexe (tehnologii din domeniul electronicii, producţia de oţel, chimie)
- Configurarea unor produse tehnice complexe (echipamente, roboţi, maşini-unelte).

În domeniul construcţiei de echipamente, utilizarea sistemelor expert poate fi justificată astfel:

- Sinteza unor grupe de elemente de construcţie pe baza unor cerinţe complexe (sisteme de poziţionare, comenzi)
- Analiza erorilor şi determinarea unor măsuri de ajustare optime (grupa elementelor din mecanica fină şi optică, sisteme de poziţionare)
- Alegerea metodei de fabricaţie şi optimizarea construcţiei în cazul fabricaţiei pe baza unor părţi componente, montaj, verificare ş. a. în faza de construcţie
- Sinteza principiilor tehnice pentru sisteme complexe formate din elemente mecanice, optice, electrice şi electronice.

Figura 3.20 exemplifică un sistem expert destinat problemelor de construcţie.

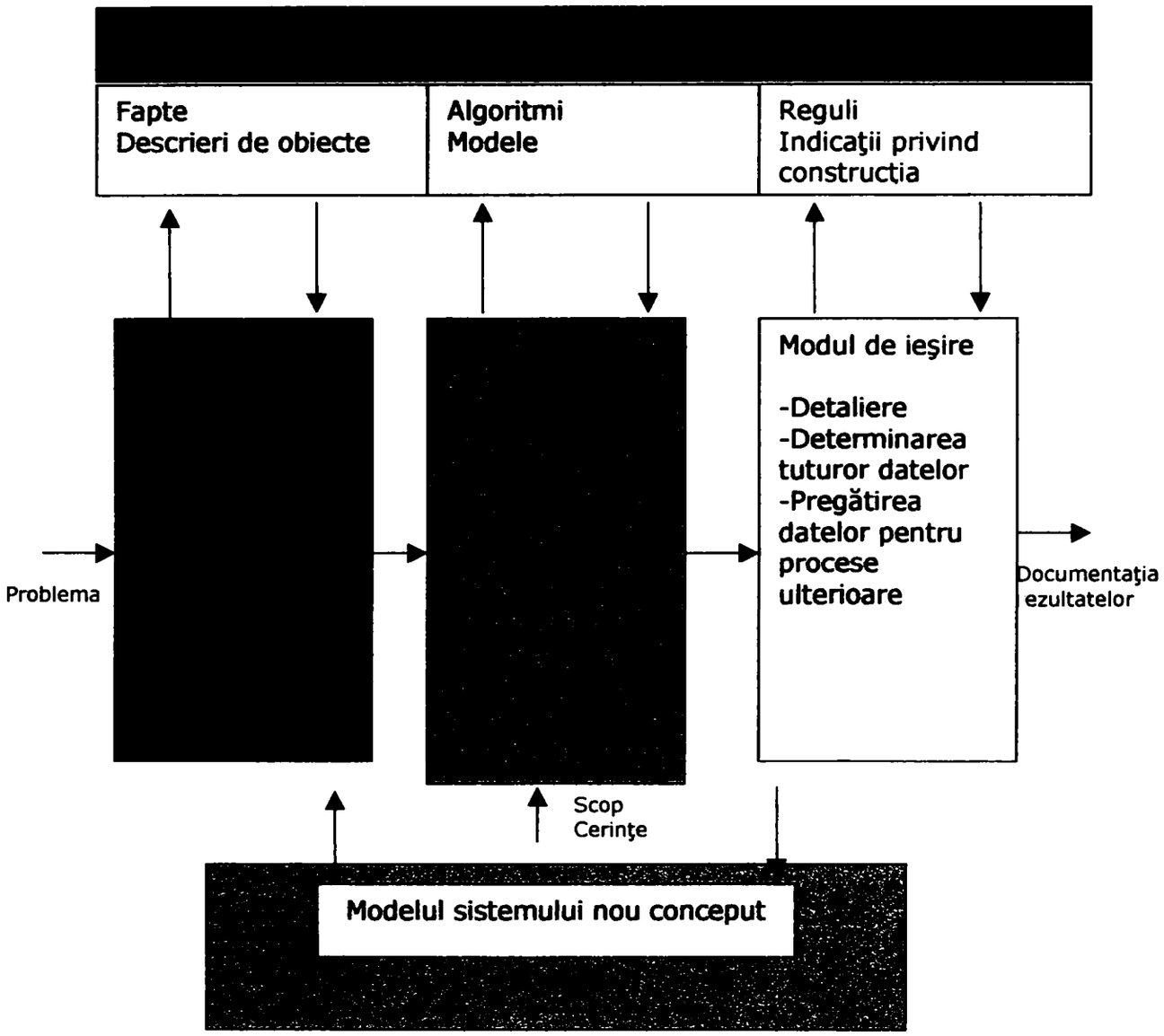


Fig. 3.20

3.11 Concluzii

Pe baza celor prezentate, se pot formula câteva concluzii:

- Utilizarea tehnicii de calcul în proiectarea şi construcţia de echipamente dobândeste o pondere crescândă,
- Metodele de analiză şi design orientate pe obiecte sunt, de asemenea, utile în activitatea de proiectare asistată de calculator,
- Limbajele de programare orientate pe obiecte reprezintă mijlocul ideal pentru realizarea componentelor software,

- Principiile unei arhitecturi moderne centrată pe componente stau la baza sistemelor tehnice flexibile şi adaptabile,
- Printr-o arhitectură bazată pe componente şi posibilitatea de configurare se pot înlănţui procese diferite aparţinând mai multor etape pe proiectare şi construcţie.

Pe parcursul capitolului au fost introduse elemente cu caracter de originalitate, care pot fi prezentate drept contribuţii ale autoarei. Printre acestea se pot enumera următoarele:

- Analiza şi sinteza materialului bibliografic,
- Arhitectura componentelor reutilizabile, publicata si transpusa in practica in proiectele realizate de autoare in cadrul concernului german HDI.

4 Utilizarea conceptelor orientate pe obiecte în construcţia asistată de calculator

În cele ce urmează, vom descrie câteva alternative de soluţii aplicabile în sisteme software destinate proiectării asistate da calculator. Soluţiile se referă la implementarea câtorva din operaţiile fundamentale şi a proprietăţilor descrise în capitolul precedent, în special din domeniul construcţiei şi sintezei variantelor, al soluţiilor CAD bazate pe obiect şi al sistemelor expert. De asemenea, se vor trata aspecte vizând rezolvarea unor probleme de configurare, de reprezentare şi realizare de liste de piese componente, documentatii etc. Prin combinarea componentelor descrise pot realiza aplicatii din domeniul gestionarii ciclului de viata al unui produs.

Deşi aspectele de reprezentare grafică au un rol esenţial într-un sistem CAD, lucrarea de faţă nu se va ocupa detaliat de ele, întrucât în domeniul grafic există o largă paletă de soluţii care se regăsesc în sistemele CAD actuale. Scopul lucrării de faţă este ilustrarea unor aplicaţii posibile pentru un domeniu aflat într-o fază de început, şi anume: determinarea unor soluţii constructive cu ajutorul calculatorului.

Conceptele din domeniul teoriei orientate pe obiecte însoţesc elaborarea soluţiilor atât din punctul de vedere al conţinutului, cât şi al formei.

Principiul care stă la baza acestei teorii, şi anume reprezentarea lumii reale sub formă de obiecte, având anumite proprietăţi, un anumit comportament şi interacţionând cu alte obiecte demonstrează că acest tip de modelare se pretează ideal pentru obiecte din domeniul industriei, respectiv al mecanicii fine.

Instanţierea unor obiecte pe baza unor clase, care descriu proprietăţile şi comportamentul obiectelor asemănătoare prezintă analogii cu procesul de proiectare a unui echipament pe baza informaţiilor memorate despre familia de echipamente din care face parte. Utilizarea conceptelor clasa si instanta conduce la specializarea activitatilor de proiectare in doua faze. In prima faza (macrodesign) intra activitatile legate de specificarea unor obiecte numite tipuri de elemente de constructie, avand un rol analog claselor din teoria orientata pe

obiecte. Aceste obiecte reprezinta la randul lor un model pentru obiecte mai concrete numite elemente de constructie, având un rol analog instantelor din teoria orientata pe obiecte. Activitatile vizand specificarea acestor obiecte concrete reprezinta cea de-a doua faza pe care o vom numi microdesign. Aceasta specializare conduce la cresterea nivelului calitativ si a productivitatii in proiectare.

Din punctul de vedere formal, se vor aplica metode de modelare şi reprezentare orientate pe obiect realizate cu ajutorul unui software de modelare (produsul Paradigm Plus) în limbajul de modelare Unified Modelling Language (UML).

Principiile de mai sus se vor concretiza sub forma unor programe vizând anumite componente ale unui sistem CAD, scoţând în evidenţă aspectele aplicării lor în probleme de construcţie şi configurare.

4.1 Descrierea generală a sistemului CAD

In cele ce urmează vom descrie un produs software în domeniul proiectării asistate de calculator (CAD). Produsul software va oferi asistenţă într-o problemă clasică de construcţie bazată pe un anumit model, în care anumite elemente de bază selectate şi parametrizate se compun după anumite reguli, realizând imaginea pe calculator a pieselor componente ale unui obiect concret (de exemplu, un aparat de măsurat), precum şi automatizarea unor procese referitoare la acest obiect.

Modelul care stă la baza sistemului va permite descrierea de variante şi sinteza bazată pe acestea. În categoria proceselor care pot fi automatizate intră, de exemplu, documentaţia obiectului proiectat şi a planului de fabricaţie sub forma unui text în format rtf, precum şi configurarea platformei de automatizare şi a codului pentru comanda numerică.

Originalitatea acestui sistem consta in asigurarea suportului tehnic privind urmatoarele aspecte:

1. specializarea activitatilor de proiectare corespunzator fazelor macro- si macrodesign,

2. domeniul proiectarii de variante,
3. activitati specifice mai etape de proiectare si constructie.

Produsul software astfel definit va funcţiona ca o unealtă de lucru, care permite definirea unor şabloane pentru obiectele de proiectat, inclusiv reguli pentru construcţia lor, precum şi aplicarea definiţiilor din şabloane pentru cazuri concrete de proiectare.

Se poate observa că "unealta" poate funcţiona în două moduri: modul "definiţie" şi modul "aplicaţie". De asemenea, se poate observa analogia dintre principiul de funcţionare a unui asemenea produs software şi procesul de instanţiere a unui obiect pe baza informaţiilor aparţinând unei clase din teoria programării orientate pe obiecte.

Ca şi în teoria limbajelor de programare, vor exista în sistemul nostru obiecte de tip clasă sub forma unor descrieri de tipuri de elemente de construcţie simple sau structurate, precum şi obiecte de tip instanţă sub forma unor entităţi reprezentând elementele de construcţie concrete, care sunt exemplare aparţinând unui anumit tip.

La nivel de tipuri, vom admite următoarele relaţii:

- generalizare/ specializare: astfel, vom spune ca un tip "mecanism" este mai general, decit tipurile "mecanism cu orologerie" sau "mecanism cu camă", respectiv, tipul "mecanism cu camă este o specializare a tipului "mecanism" (proprietate preluată din teoria orientată pe obiecte);
- compoziţie: astfel, vom putea crea tipuri structurate, precizând elementele lor componente (analog tiparului "Composite Pattern");
- referinţă: astfel se vor putea declara lanţuri şi reţele de obiecte.

Pentru modelarea variantelor vom introduce pentru relaţiile de compoziţie calificatorii "ŞI", "SAU", "OBLIGATORIU", "OPŢIONAL".

Relaţia dintre un obiect de tip clasă şi unul de tip instanţă are următoarele proprietăţi:

- Fiecărui obiect instanţă îi corespunde un obiect clasă,

- Obiectul clasă furnizează structura de bază a obiectului instanţă,
- Obiectul clasă furnizează valori prestabilite pentru anumite atribute ale obiectului instanţă,
- Obiectul clasă determină comportamentul obiectului instanţă (dirijează procese ale obiectului instanţă, de exemplu, anumite calcule sau generarea unui document).

Configuraţia unui obiect clasă înseamna definirea datelor de bază, precum şi a unui şablon pentru obiectele instanţă cu trăsături comune, inclusiv valori predefinite ale atributelor şi informaţii pentru dirijarea unor procese.

Datele de bază reprezintă un pool de obiecte statice definite de utilizator, obiecte care pot fi utilizate în comun atât de către obiectele clasă, cât şi de obiectele instanţă. Datele de bază pot fi la rândul lor obiecte de tip clasă (tipuri de elemente de construcţie) şi obiecte de tip instanţă (elemente de construcţie concrete).

Spre deosebire de sistemele CAD tradiţionale, baza de date (datele de bază împreună cu tipurile de elemente de constructie precum şi elementele de constructie concrete) a sistemului CAD propus trebuie dezvoltată succesiv pe baza unui dialog specializat cu utilizatorul (nu sunt excluse existenţa unui set de tipuri de date elementare sau funcţii de extindere de tip "import"). Această proprietate garantează adaptabilitatea sistemului la cerinţele utilizatorilor din domenii având un grad înalt de specializare.

Organizarea activităţii de proiectare în doua faze (clasă/ definiţie şi instanţă/ aplicaţie) presupune un efort de concepţie ridicat în faza de definiţie (descrierea şablonului constructiv şi funcţional, inclusiv configurarea variantelor), oferă însă o productivitate ridicată în faza de aplicaţie pe baza automatizării proceselor configurate în faza anterioară.

4.1.1 Diagrame use case

În urma analizei activităţilor şi a actorilor sistemului, rezultă următoarele diagrame use case:

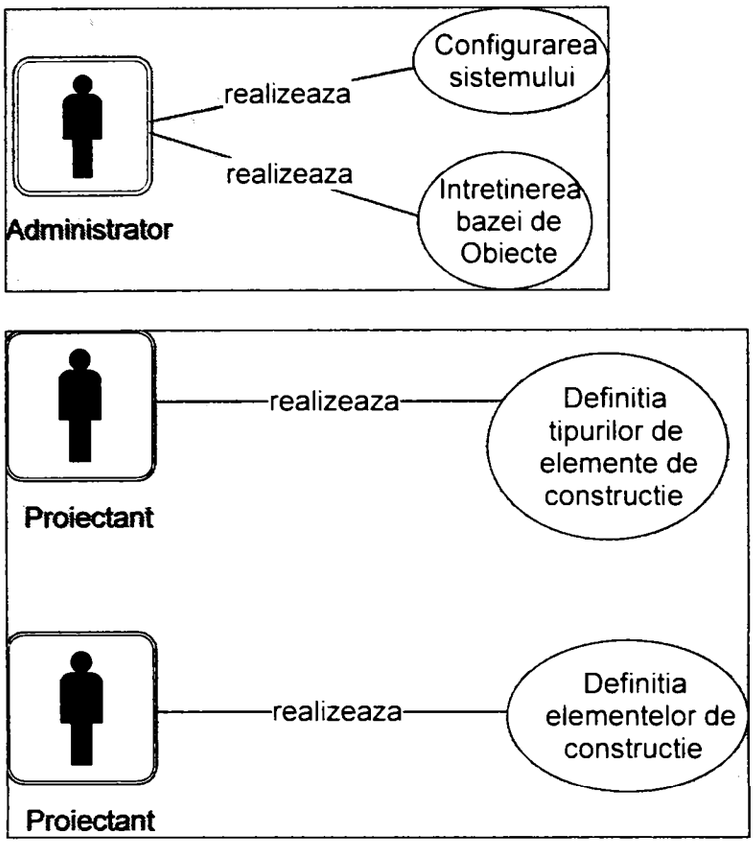


Fig. 4.1

Diagramele din figura 4.1 ilustrează interacţiunile de bază ale actorilor (administrator, proiectanţi) cu sistemul CAD şi modul de specializare a locurilor de muncă.

4.1.2 Arhitectura de specialitate a sistemului

Pornim de la ideea realizării unei arhitecturi orientate pe componente şi deschise.

Sistemul nostru se poate împărţi în componente de specialitate şi componente technice. Componentele de specialitate sunt cele care înglobează funcţii specifice domeniului de specialitate pentru care se realizează proiectul (o astfel de componentă ar fi construirea unei baze de informaţii într-un anumit domeniu de specialitate, de exemplu mecanică fină). Componentele tehnice asigură servicii pentru realizarea cerinţelor dintr-un domeniu de specialitate sau funcţionarea performantă a sistemului.

În cazul de faţă, putem vorbi despre două componente centrale de specialitate care formează nucleul sistemului.

O componentă centrală va asigura definiţia tipurilor de elemente de construcţie (de exemplu, afişaj, regulator, carcasă etc.) şi a elementelor de construcţie concrete pentru diferite domenii de specialitate sau grupe de construcţie (conform principiului construcţiei orientate pe obiecte).

A doua componentă centrală permite elaborarea tipurilor de elemente de constructie şi a elementelor de constructie concrete pe baza şabloanelor predefinite de obiectele abstracte.

Tipurile pot conţine reguli care descriu într-o anumită sintaxă cunoştinţele unui expert uman. Aceste reguli pot descrie calcule reprezentate sub forma unor expresii generice, acţiuni condiţionate sau necondiţionate specifice unor procese de configurare. Astfel, componentele centrale vor comunica cu o componentă tehnică de configurare destinată descrierii şi interpretării unor expresii formale şi reguli.

Deoarece domeniul de specialitate al produsului nostru software nu este limitat, modelul pe care se bazează componentele centrale este generic. Numărul redus de clase aparţinând nucleului şi gradul înalt de abstractizare fac posibilă utilizarea sistemului în diferite domenii. Privit din acest punct de vedere, sistemul de componente apare ca un framework (cadru) uşor de extins sau de integrat într-un anumit peisaj informatic în care modelul va fi îmbogăţit cu tipuri de date şi obiecte definite în exclusivitate de către utilizatori, fără a fi necesară o activitate suplimentară de programare.

În jurul nucleului sistemului se vor grupa componente de specialitate şi tehnice, care realizează diferite aspecte de prelucrare. Separarea aspectelor de prelucrare este argumentată prin utilizarea de componente dedicate diferitelor tipuri de prelucrare, ceea ce conduce la simplificarea nucleului sistemului. Nucleul va conţine un model propriu, adecvat componentelor sale, în timp ce aspectele de prelucrare pot lucra cu alte modele, dupa necesităţi. Informaţii ale modelului intern (specific nucleului) se vor exporta către componentele destinate aspectelor de prelucrare (componente externe).

Vom exemplifica utilizarea componentelor tehnice printr-o componentă destinată realizării cuplajului dintre două sisteme, o componentă destinată descrierii formale şi a interpretării unor expresii şi reguli, precum şi o componentă destinată prelucrărilor de texte. Aspectul de prelucrare utilizat ca exemplu va fi realizarea documentaţiei pentru un element de constructie concret.

Ponim de la ideea utilizării de software standard pentru reprezentarea grafică (de exemplu, AUTOCAD) integrat în sistemul nostru prin intermediul unei interfeţe.

Arhitectura de componente ale sistemului este vizualizată în figura 4.2.

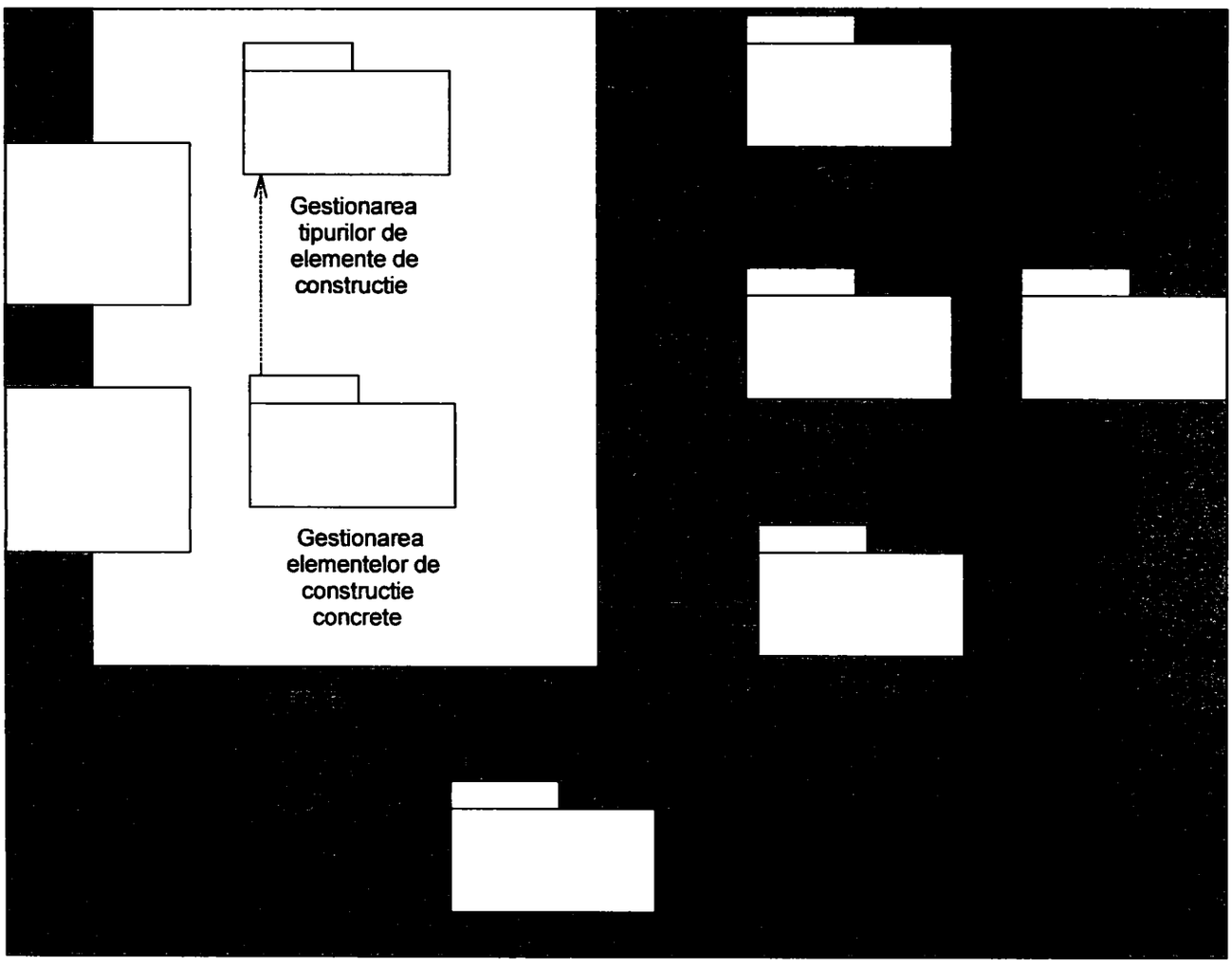


Fig. 4.2

4.1.3 Arhitectura tehnica a sistemului

Din punct de vedere tehnic se vor distinge doua medii de utilizare a componentelor sistemului CAD in functie de destinatia lor. Vor exista doua

grupuri distincte de utilizatori (vezi modelul Use Case): un grup restrans de proiectanti de tipuri abstracte de elemente de constructie, respectiv un grup numeros de proiectanti de elemente concrete de constructie – utilizatori ale informatiilor elaborate de primul grup.

Componenta destinata gestionarii tipurilor de elemente de constructie (GTEC) va avea o arhitectura de tip client-server (fat client) si poate functiona, de exemplu, pe baza unui server Windows NT , deoarece numarul utilizatorilor este restrans. Datele vor fi stocate intr-o baza de date relationala sau orientata pe obiect. Aceasta arhitectura este reprezentata in figura 4.3.

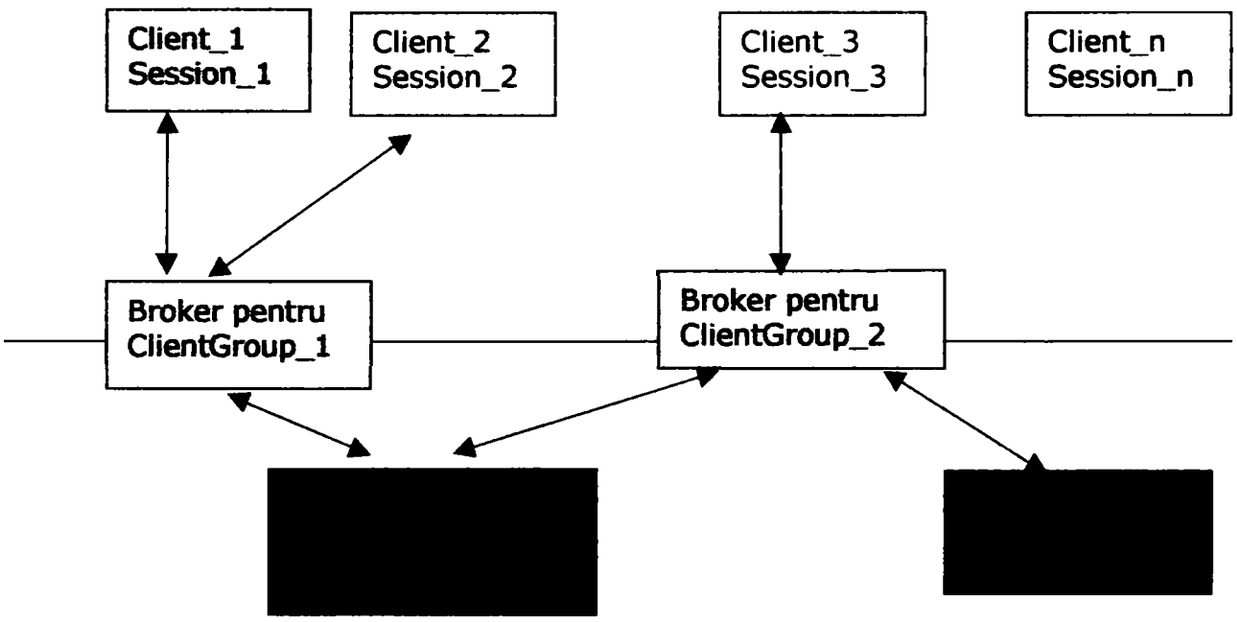


Fig. 4.3

Componenta destinata gestionarii de elemente de constructie concrete (GECC) va avea de asemenea o arhitectura de tip client-server (thin client) si poate functiona, de exemplu, intr-un mediu compatibil cu J2EE. Un numar ridicat de clienti vor solicita serviciile puse la dispozitie de componenta GTEC sub forma unei componente de tip server numita TEC-Adapter. Interfata Adapterului se poate realiza sub forma de EJB (Enterprise Java Bean) pentru a asigura o buna integrabilitate intr-un mediu J2EE. Aceasta arhitectura este reprezentata in figura 4.4.

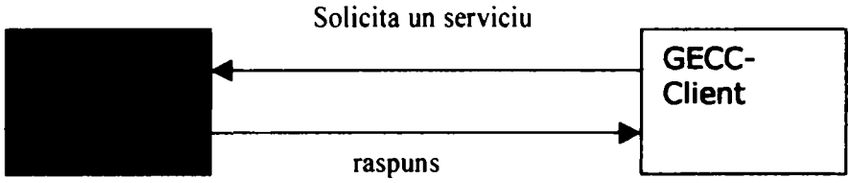


Fig. 4.4

Pentru gestionarea datelor componentei GECC se poate utiliza de asemenea o baza de date relationala sau orientata pe obiect.

4.1.4 Organizarea pe nivele

Privita sub aspectul organizării pe nivele, sistemul nostru va prezenta următoarele trei nivele:

- Nivelul bazei de date, cuprinzând bazele de date şi componenta destinată gestionării accesului la baza de date;
- Nivelul de aplicaţie, cuprinzând componentele
- Nivelul de prezentaţie.

Reprezentarea grafică a acestei organizări este realizată în figura 4.5.

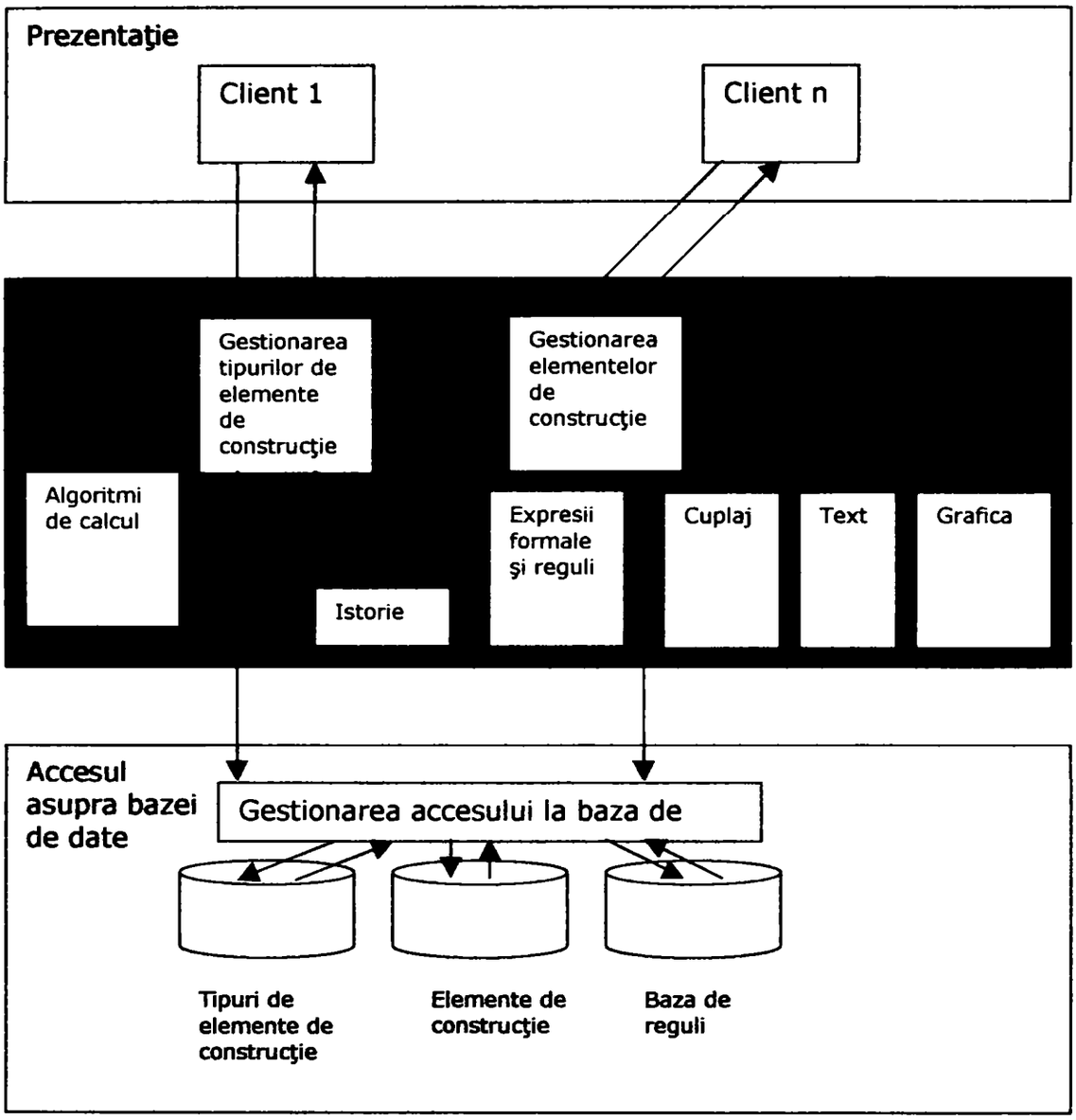


Fig. 4.5

4.2 Modelul de clasă

4.2.1 Tipuri de elemente de constructie

Un tip de element de constructie destinat unui anumit domeniu de specialitate este un element de construcţie abstract, poate intra la rândul său în componenţa altor tipuri de elemente de constructie mai complexe şi se caracterizează prin următoarele proprietăţi:

- reprezintă o descriere de tip (şablon) pentru elemente de constructie concrete, inclusiv valori prestabilite sau formule de calcul pentru anumite atribute,

- prezintă componente care descriu tipul comportamentului elementelor de constructie concrete. Aceaste componente vizează dirijarea unor procese de comportament al produselor concrete (de exemplu, generarea documentaţiei, calcule, asamblare etc.), precum şi cuplajul lejer al unor componente externe.

Descrierea informaţiei de tip dintr-un tip de element de constructie se realizează pe baza unui tipar clasic de compoziţie (composite pattern). Pentru a modela familii sau variante de elemente de constructii, relaţiile de compoziţie se vor caracteriza prin calificatorii "ŞI", "SAU", "OBLIGATORIU", "OPŢIONAL".

Tipurile de elemente de construcţie se modelează printr-o ierarhie de specializare (figura 4.6).

Astfel, se vor defini tipuri simple şi compuse ca specializări ale clasei TipDeElementDeConstructie.

Componentele unui tip compus vor fi numite atribute şi pot fi la rândul lor de asemenea, tipuri compuse. Astfel, se pot construi tipuri de obiecte deosebit de complexe, fiind posibilă modelarea tuturor detaliilor necesare într-un anumit domeniu, inclusiv cele legate de reprezentarea grafică.

Pentru un tip valoric simplu se poate preciza tipul valorii, o valoare iniţială prestabilită, precum şi o expresie de calcul în cazul în care valoarea depinde de alte valori din structură. Tipul valorii este un tip standard. Tipurile standard pot fi următoarele: numeric, alfanumeric, dată etc.

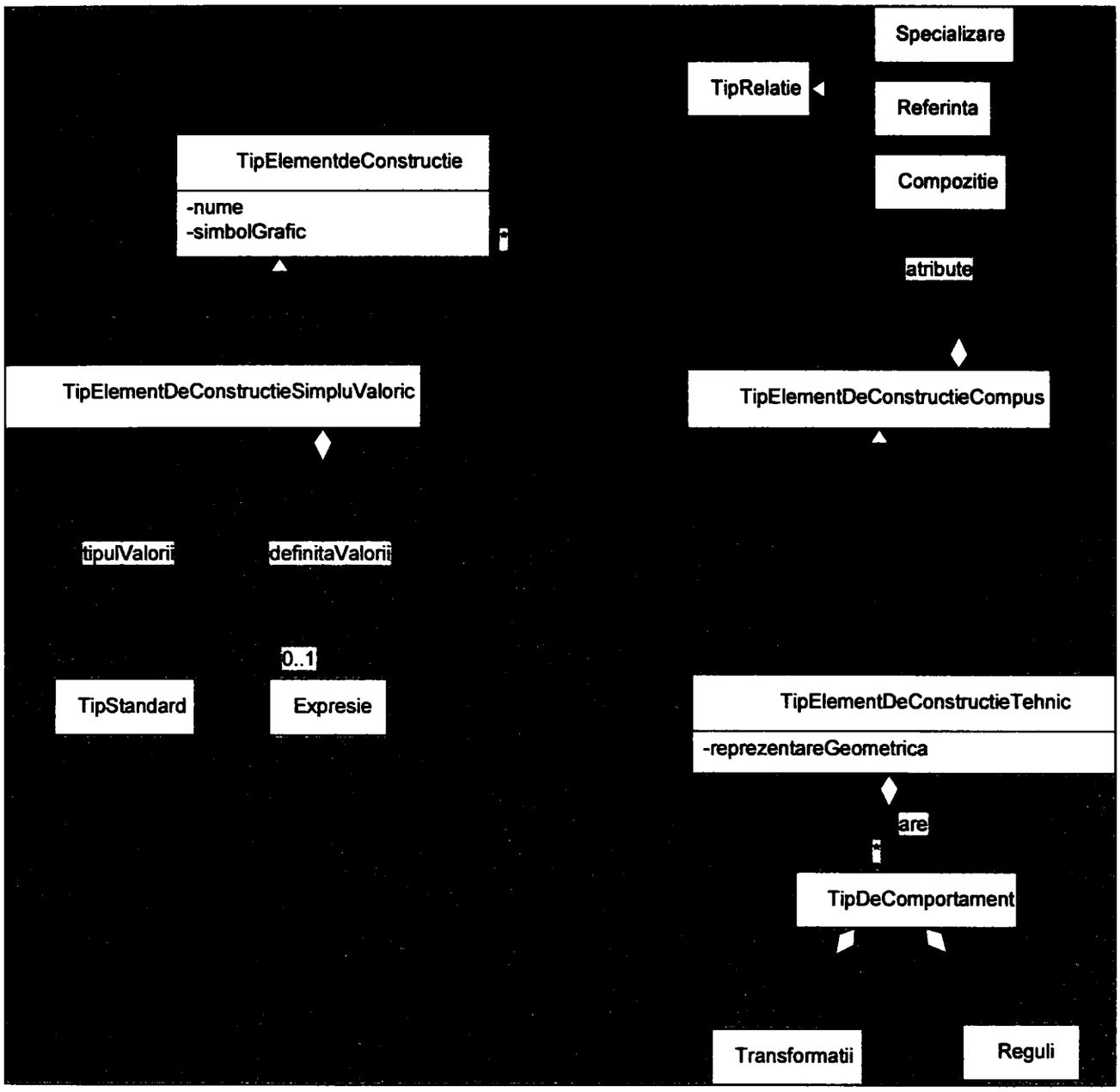


Fig. 4.6

Un element de construcţie tehnic este un tip compus, având o anumită reprezentare geometrică, parametri tehnici (atribute) reprezentaţi prin tipuri valorice şi având un comportament exprimat printr-o mulţime de tipuri de comportament. Un tip de comportament conţine specificarea transformaţiei modelului intern în modelul extern specific comportamentului, precum şi regulile care dirijează comportamentul respectiv.

De exemplu, pentru aspectul de prelucrare "Documentaţie", transformaţiile vor descrie transferul valorilor de atribute din modelul intern în documentul de creat,

iar regulile pot preciza includerea sau excludera unor pasaje de text în concordanţă cu obiectul concret de documentat.

Configurarea unui aspect de comportament se poate realiza si prin indicarea unui algoritm de proiectare sau al unei componente externe de specialitate.

Un exemplu concret de configurare a unui tip de element de constructie avand un anumit tip de comportament va fi tratat intr-un capitol separat in aceasta lucrare. Exemplul provine din domeniul proiectarii automate a sistemelor optice si se refera la specificarea unor subprograme de calcul pentru luneta Kepler privita ca tip de element de constructie.

4.2.2 Elemente de constructie concrete

Elementele de constructie sunt concretizări ale unui tip abstract de element de constructie, având un comportament concret (o mulţime de prelucrări), care rezultă din definiţia tipului respectiv. Elementele de constructie pot fi la rândul lor inglobate în ansamble complexe.

Modelul de clasă al elementelor de construcţie este, analog tipurilor elementelor de constructie, o ierarhie de specializare (figura 4.7).

Fiecare element de construcţie concret posedă o legatură cu tipul căruia îi aparţine. Această legătura permite utilizarea informaţiilor definite la nivel de tip în procese concrete.

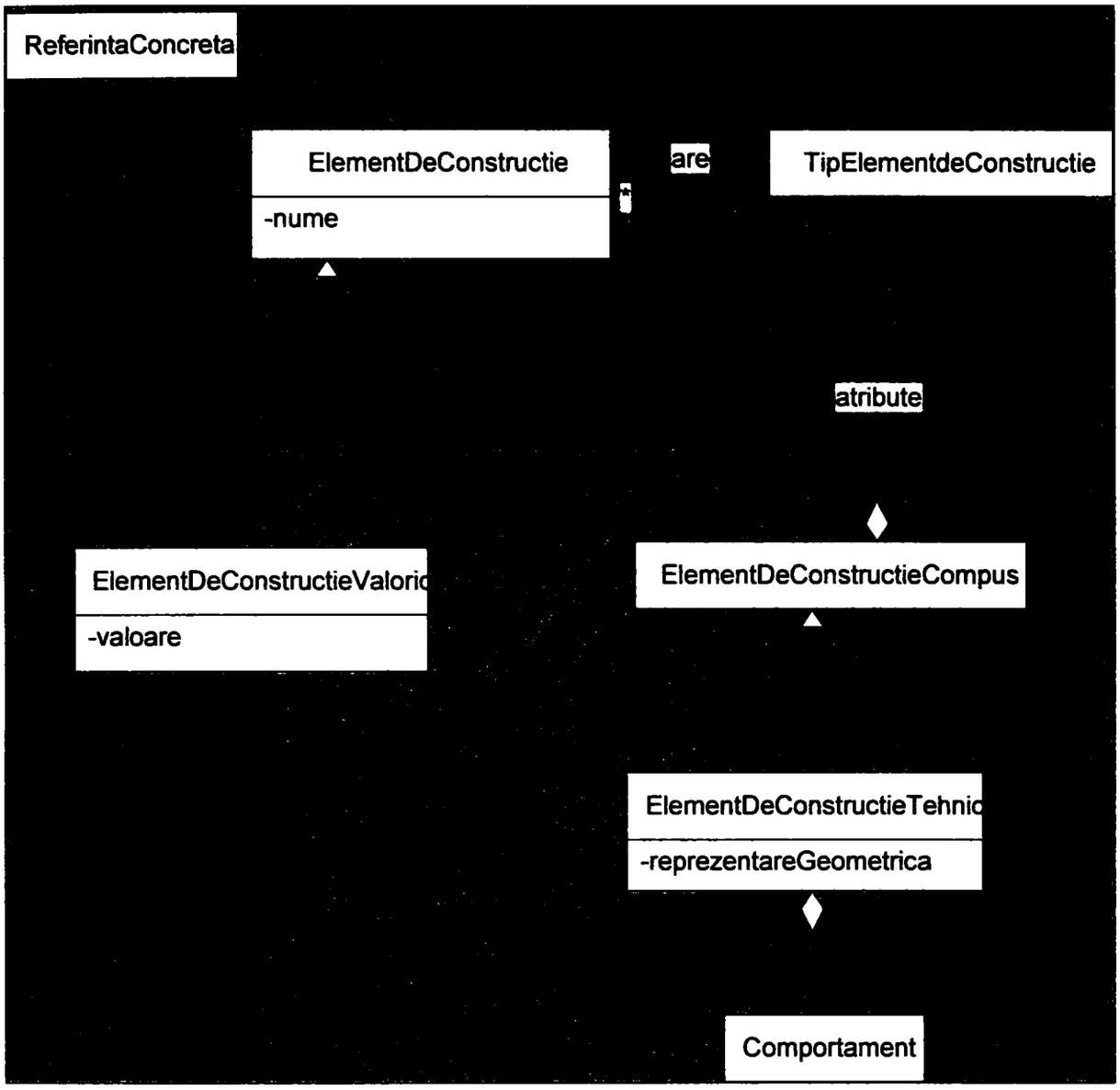


Fig. 4.7

4.3 Descrierea familiilor şi a variantelor de elemente de constructie

Familiile şi variantele de elemente de constructie pot fi modelate ca agregate (structuri compuse) pe baza tiparului "Composite Pattern", folosind legi de compoziţie:

1. Compoziţie bazată pe operatorii de compoziţie "ŞI", respectiv "SAU"

Pornind de la o reprezentare sub forma unui arbore (un caz particular de graf), în cazul compoziţiei pe baza operatorilor "ŞI", respectiv "SAU", structura definită va conţine un nod "ŞI" ca rădăcină. În interiorul structurii pot apare noduri "SAU"

care au ca succesori subarbori alternativi. Nodurile terminale vor reprezenta tipuri de elemente de construcţie.

Prin aducerea grafului la forma normală disjunctivă, se obţine mulţimea tuturor soluţiilor alternative sau a variantelor.

2. Compoziţie bazată pe operatorii de compoziţie "OBLIGATORIU", respectiv "OPŢIONAL"

În cazul compoziţiei bazate pe operatorii "OBLIGATORIU", respectiv "OPŢIONAL", arborele va conţine în interior subarbori opţionali, care se pot include sau exclude în procesul de construcţie. Această variantă de modelare presupune utilizarea implicită a operatorului "ŞI".

3. Compoziţie mixtă, bazată pe utilizarea tuturor operatorilor de compoziţie

Această variantă de compoziţie reuneşte proprietăţile prezentate în cadrul punctelor 1 si 2.

4.3.1 Modalităţi de descriere a comportamentului

Complexitatea descrierii comportamentului unei familii de elemente de constructie este proporţională cu mulţimea de alternative modelată printr-o structură. În funcţie de metoda de modelare adoptată, există diferite mecanisme de descriere a comportamentului.

În cazul modelării pe baza operatorilor "ŞI", "SAU", regulile de transformare a structurii interne a familiei de elemente de constructie într-o structură externă (de exemplu, structura documentaţiei) se vor putea defini cu ajutorul unei componente de cuplaj descrisă în capitolul "Cuplajul lejer dintre două sisteme". Deciziile privind procesele alternative, corespunzând variantelor modelate, se definesc sub formă de reguli cu ajutorul unei componente de configurare descrisă în capitolul "Descrierea şi interpretarea unor expresii formale şi reguli".

În cazul modelării pe baza operatorilor "OBLIGATORIU", "OPŢIONAL", transferul de informaţii către structura externă şi descrierea variantelor de decizie pot fi rezolvate pe baza mecanismelor oferite de componenta de configurare.

Posibilitatea configurării unei familii de soluţii şi utilizarea unor componente pentru transformarea structurilor şi formularea unor definiţii generice pe bază de expresii formale şi reguli apropie sistemul CAD descris de sistemele expert. În acest sens, putem considera că baza de cunoştinţe este formată dintr-o parte declarativă conţinând definiţiile de tipuri şi regulile de transformare, precum şi o parte imperativă formată din regulile de decizie.

4.3.2 Fereastra de configurare şi construcţie

Prototipul fereastrei principale de lucru destinată configurării tipurilor de elemente de construcţie (inclusiv a variantelor constructive), precum şi construcţiei efective este reprezentată în figura 4.8.

În partea stângă a ferestrei se reprezintă structura arborescentă într-un element grafic de tip TreeView. Proprietăţile elementului de construcţie (abstract sau concret) selectat se pot preciza în grupul numit Properties. Atributele valorice se vor configura într-un tabel, proprietăţile geometrice se vor vizualiza într-un compartiment separat. Coloana Composition conţine operatorul de compoziţie ("ŞI", "SAU", "OBLIGATORIU", "OPŢIONAL") prin care s-a inclus elementul de construcţie selectat în structura proiectată. Butonul Behavior serveşte la specificarea comportamentului elementului de construcţie. Aceasta se realizează prin intermediul unor construcţii speciale, recurgând la definiţii de mapping şi reguli.

Meniul va conţine opţiuni grupate în categorii.

Modul de lucru (configurare sau construcţie) se va preciza la începutul unei sesiuni de lucru în cadrul grupei de meniuri Options.

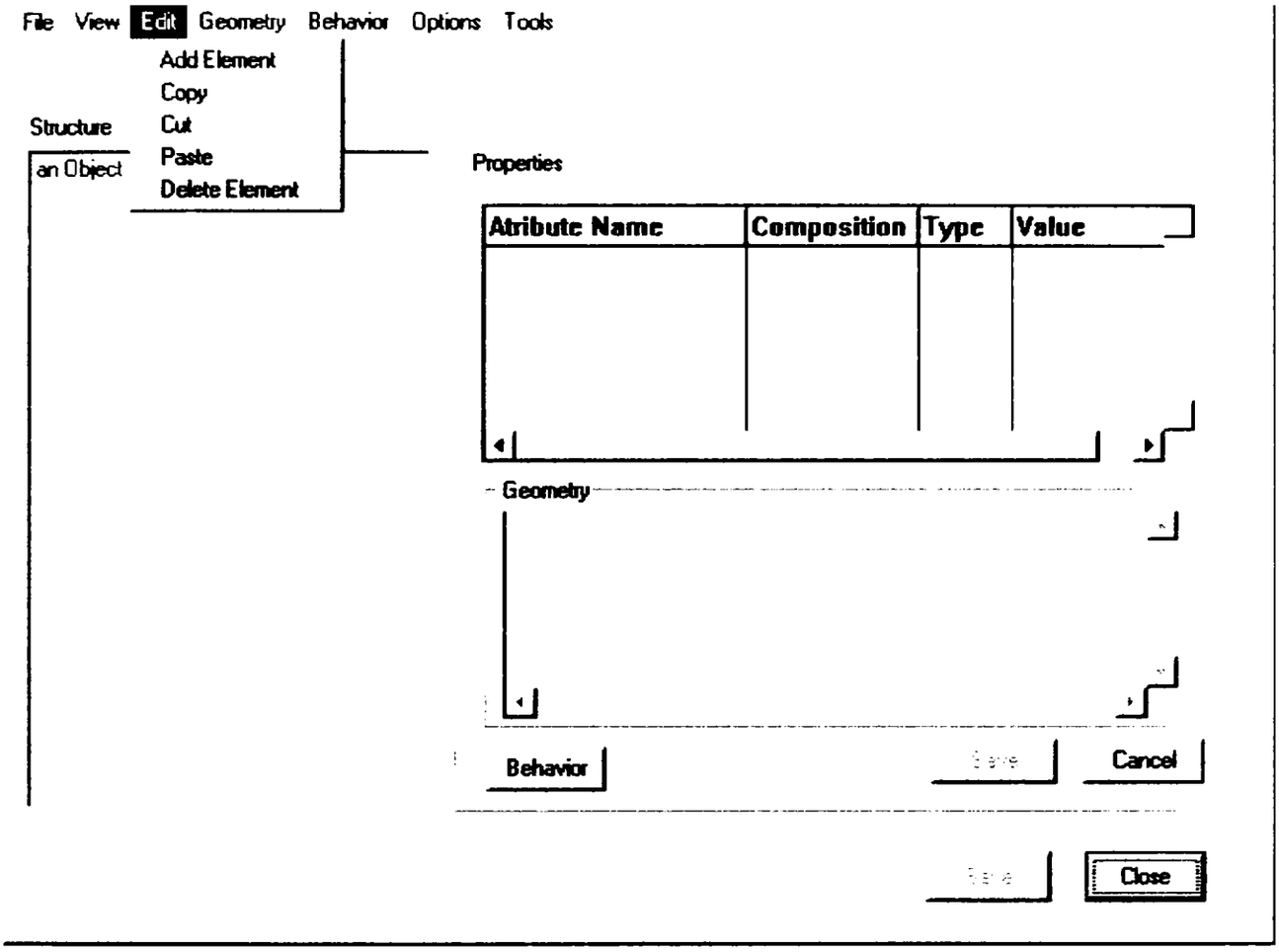


Fig. 4.8

4.4 Generarea şi editarea de texte

Generarea şi editarea de texte se va realiza printr-o componentă specială. Vom numi această componentă TEXT.

Componenta TEXT face posibilă gestionarea elementelor de construcţie de tip text, compoziţia documentelor, precum şi arhivarea lor.

În această ordine de idei, sistemul TEXT va include două părţi componente: o componentă pentru administraţia şi una pentru compoziţia de texte.

Componenta "Administraţie" realizează gestionarea elementelor de text. Evoluţia în timp a unui element este înregistrată printr-un cîmp de tip dată (valabilitate). Prin acest parametru se pot referenţia diferite stări istorice ale unui element.

Pornim de la ideea că elementele de text sunt tipizate după cum urmează: elemente simple, elemente structurate şi elemente locţiitoare.

Elementele simple referenţiază documente în format rtf (inclusiv imagini). Ele pot conţine variabile.

Un element locţiitor exprimă faptul că un anumit element de text poate apare într-un anumit loc o dată, de mai multe ori sau deloc.

Componenta "Compoziţie" este unealta pentru prelucrarea şi vizualizarea de documente. Vizualizarea se realizează prin utilizarea de software standard (Winword).

Legătura dintre un obiect concret şi documentele sale se realizează printr-o cheie.

3.4.1 Modelul de clasa

Modelul de clasă al acestei componente este redată în figura 4.9:

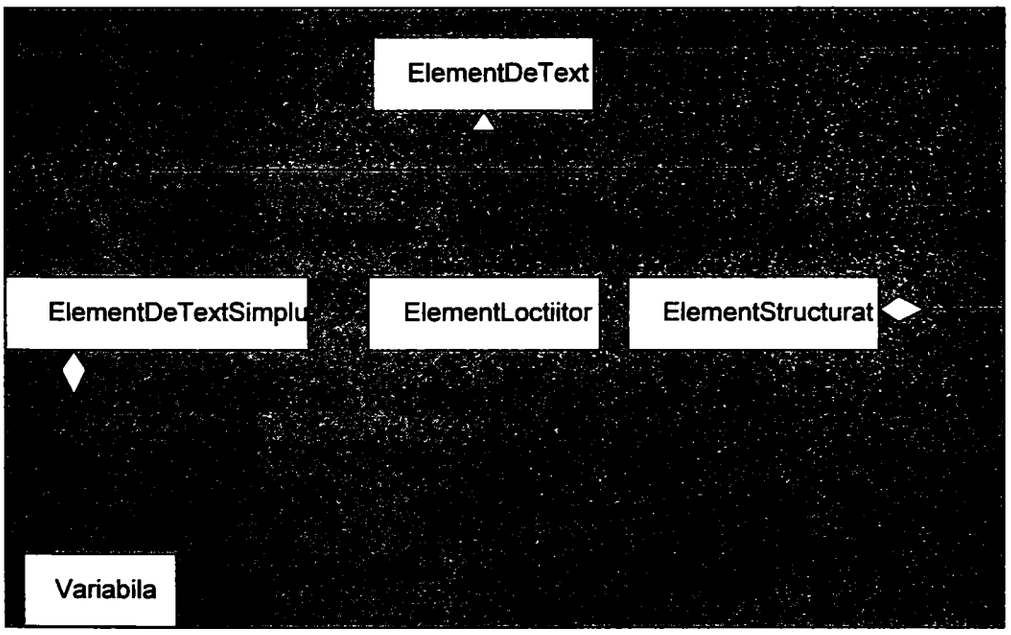


Fig. 4.9

Interacţiunea dintre componenta TEXT şi componenta client se poate realiza pe baza unei componente de cuplaj (această variantă este exemplificată în capitolul "Cuplajul lejer dintre două sisteme") sau pe baza unei componente de configurare (exemplificată în capitolul "Descrierea şi interpretarea unor expresii formale şi reguli").

4.4.2 Aplicaţii în rezolvarea unor probleme de construcţie sau configurare

Componenta TEXT se poate aplica în următoarele domenii:

4.4.2.1 Documentatia unui element de constructie

Documentaţia se poate descompune în elemente de text şi grafice reutilizabile, documentul fiind compus cu ajutorul calculatorului din aceste elemente.

4.4.2.2 Documentarea planului de fabricaţie

Planul de fabricaţie se poate descompune, de asemenea, în elemente de text şi grafice reutilizabile, fiind compus cu ajutorul calculatorului din aceste elemente.

4.4.2.3 Editarea şi structurarea codului de program pentru comanda numerică

Codul de program poate fi descompus în elemente pentru a fi asamblat ulterior pe baza unor reguli de configurare.

4.5 Descrierea şi interpretarea unor expresii formale şi reguli

În dialogurile purtate cu beneficiarii unor produse software, precum şi cu unii experţi umani, sunt frecvente exprimările sub forma unor reguli. O regulă descrie în principiu o acţiune sau o secvenţă de acţiuni condiţionată sau necondiţionată. Condiţia - dacă este prezentă- descrie o situaţie în care se execută acţiunea sau secvenţa de acţiuni.

În sistemul nostru cad orientat pe obiecte, regulile descriu interdependenţe complexe şi interacţiuni între diferitele obiecte, extinzând flexibilitatea configurării de produse peste graniţele cunoştinţelor programate. Sistemul nostru se aseamănă prin aceasta cu un sistem expert bazat pe reguli de producţie, care este format dintr-o bază de date (care pe lângă fapte conţine şi regulile de prelucrare şi înlănţuirile dintre informaţii), o bază de reguli şi un interpreter pentru reguli. În acelasi timp, apelul şi execuţia regulilor poate fi privită şi ca un

apel de procedură la distanţă, considerând că un obiect solicită declanşarea (uneori condiţionată) a unor acţiuni care fac parte din serviciile oferite de un alt obiect.

- Regulile sunt descrise formal sub forma unor construcţii sintactice formate din cuvinte cheie şi expresii pe baza cărora se generează un arbore sintactic. Regulile din sistemul de faţă se vor modela sub forma unor structuri de control dintr-un limbaj de programare imperativ (structuri de tip secvenţă şi structuri ramificate de tipul „IfThen" , „IfThenElse" cât şi „Select").

După efectul produs, putem clasifica regulile în două categorii:

- **Reguli care nu modifică starea sistemului** (în această categorie intră implicaţiile prin care se deduce valoarea de adevăr a unei expresii, de exemplu, "dacă produsul proiectat conţine componenta X, componenta Y este implicit activă").
- **Reguli care modifică starea sistemului** (în această categorie intră interacţiunile dintre obiecte, de exemplu, dacă în produsul de proiectat apar componentele A şi B, atributul i al componentei X va primi valoarea "AB" şi va fi activată componenta opţională Y).

Componenta de faţă oferă utilizatorilor un framework pentru:

- Definiţia de reguli,
- Traducerea faptelor din reguli descrise generic într-un obiect (context) destinaţie.

Pentru realizarea celor propuse mai sus, sunt necesare: un mecanism pentru identificarea obiectelor într-un anumit context şi un mecanism pentru realizarea "dialogului" dintre obiecte pe bază de mesaje.

Astfel, fiecare obiect din sistem va fi identificat printr-un tipar unic (tip+nume+ identificator sau drumul dintr-un graf) şi va implementa un protocol special (o listă de atribute vizibile în afara obiectului, precum şi metode care realizează accesul la elementele descrise printr-un tipar, concretizarea unei acţiuni, reprezentarea externă într-o anumită formă etc).

Regulile pot fi clasificate în module. Un modul se poate executa în intregime, astfel încât putem privi un modul ca o acţiune complexă.
Regulile se vor executa într-o ordine fixată. Aceasta se poate extinde şi la nivelul modulelor.

Regulile vor fi definite în totalitate de către utilizatorii sistemului în procesul definirii obiectelor de tip clasă. Fiecare regulă va avea un proprietar (un obiect de tip clasă) care precizează contextul în care se vor interpreta informaţiile descrise formal. Deoarece un obiect de tip instanţă are o legatură permanantă cu un obiect de tip clasă, poate utiliza regulile obiectului de tip clasă. În acest scop, obiectul instanţă trebuie să comunice contextul propriu la apelul regulei.

Se poate observa că soluţia de faţă nu necestită integrarea unui sistem expert clasic. Aceasta se explică prin faptul că sistemul descris nu conţine reguli complexe din punctul de vedere al raţionamentului determinării unei soluţii, nu sunt necesare soluţii alternative, verificarea unor soluţii etc. Astfel, problema propusă de noi nu prezintă un argument pentru integrarea unei componente complexe care ar presupune operaţii de convertire a faptelor într-o altă descriere formală, în ambele sensuri (înspre şi dinspre sistemul expert), ceea ce ar afecta nejustificat performanţa sistemului.

4.5.1 Expresii

Expresiile sint construcţii sintactice care fac posibilă definiţia valorilor unor atribute şi a regulilor.

Expresiile se împart în:

- Expresii de acces,
- Expresii de tip valoare,
- Expresii compuse.

4.5.1.1 Expresii de acces

Expresiile de acces descriu formal referinţe asupra unui anumit obiect sau asupra unui atribut al unui obiect.
În cazul în care expresia de acces descrie un atribut al unui obiect, ea realizează descrierea unei legături de tip obiect-mesaj. Atributul precizat se va accesa prin

intermediul unei metode de acces (getter) care se va apela în timpul execuţiei (interpretării expresiei). Din nou este evidentă asemănarea cu apelul de metodă la distanţă.

Pentru descrierea obiectului de identificat trebuie ales un tipar adecvat.

În sistemele cu structuri statice, în care obiectele de tip instanţă se încadrează în întregime în structurile definite de obiectele de tip clasă, se recomandă identificarea obiectelor pe baza unui tipar descris de drumul de la rădăcina arborelui care reprezintă contextul până la obiectul căutat.

În sistemele care lucrează cu structuri care pot varia în anumite limite şi, deci, nu corespund în întregime celor din obiectele clasă, se poate utiliza un tipar unic compus din tipul obiectului, precum şi un identificator care permite selectarea unui anumit exemplar.

Rezultatul evaluării tiparului depinde întotdeauna de context. Astfel, este posibilă descrierea unei reguli utilizând ca punct de plecare un obiect de tip clasă şi evaluarea ulterioară a acestei reguli în contextul unui obiect de tip instanţă.

Exemple:
Aparat optic / Sistem optic / Sistem de inregistrare
Sistem optic / Obiectiv / Deschidere / Distanta focala
Ocular / Grosisment

Se poate observa că primul exemplu utilizează accesul pe baza drumului din arbore, în timp ce ultimele două exemple se referă la exemplarul nr. 1 al tipului sistem optic din context.

4.5.1.2 Expresii de tip valoare

Expresiile de tip valoare descriu valori (constante) de tip alfanumeric, numeric, procent, logic, dată, obiect, listă, preţ etc.

Exemple:
'Acesta este un text'

5 %

4.5.1.3 Expresii complexe

O expresie complexă este formată dintr-un operator şi una sau două expresii legate prin operatorul respectiv. În procesul de compoziţie a unei expresii complexe se poate recurge la operatori matematici şi operatori speciali.

Expresiile complexe se împart în: expresii de selecţie, expresii matematice şi expresii speciale.

- Expresii de selecţie

Expresiile de selecţie realizează descrierea selecţiei unui obiect sau a unei liste de obiecte cu anumite trăsături. Identificarea obiectelor se realizează pe baza unui tipar mai general care exprimă condiţia de căutare (în acest caz nu se recomandă descrierea obiectelor pe baza unui drum din graf).
Definiţia tiparului conţine tipul unui obiect şi o expresie logică reprezentând proprietatea dorită (de exemplu: motor cu randamentul mai mare decât 50 %). Mulţimea soluţiilor se poate parametriza (PrimulElement, UltimulElement, ToateElementele).
Rezultatul evaluării unei expresii de selecţie depinde de context.

Exemple:
SELECTIE: ToateElementele *(sistem optic, grosisment > 10 X)*
SELECTIE: PrimulElement *(obiectiv, deschidere relativa q>1/2,8)*
SELECTIE: UltimulElement *(ocular, grosisment > 8 X)*

- Expresii matematice

Din categoria expresiilor matematice fac parte expresii numerice "clasice", expresii logice şi comparaţii. Compoziţia acestor expresii se realizează pe baza unor operatori unari sau binari.

Exemple:
(sistem optic (1).grosisment > 20 X)

NOT (sistem optic. grosisment > 20 X)

- Expresii speciale

În această categorie se includ expresii care utilizează operatori speciali (de exemplu: operatorul AreValoareDefinita aplicat unei expresii de acces conduce la o expresie având o valoare de tip logic).
Rezultatul evaluării unei expresii speciale depinde, de asemenea, de context.

Exemple:
AreValoareDefinita (Sistem optic.Grosisment>40x)

4.5.1.4 Plauzibilităţi

În procesul de compoziţie se efectuează teste de plauzibilitate (numărul şi tipul operanzilor) pentru eliminarea erorilor sintactice şi semantice.

4.5.1.5 Evaluarea expresiilor

Fiecare expresie livrează o valoare având unul din tipurile standard acceptate de sistem. Rezultatul evaluării unei expresii poate depinde de contextul evaluării.

4.5.1.6 Modelul de clasă

Modelul de clasă al expresiilor este reprezentat în figura 4.10.

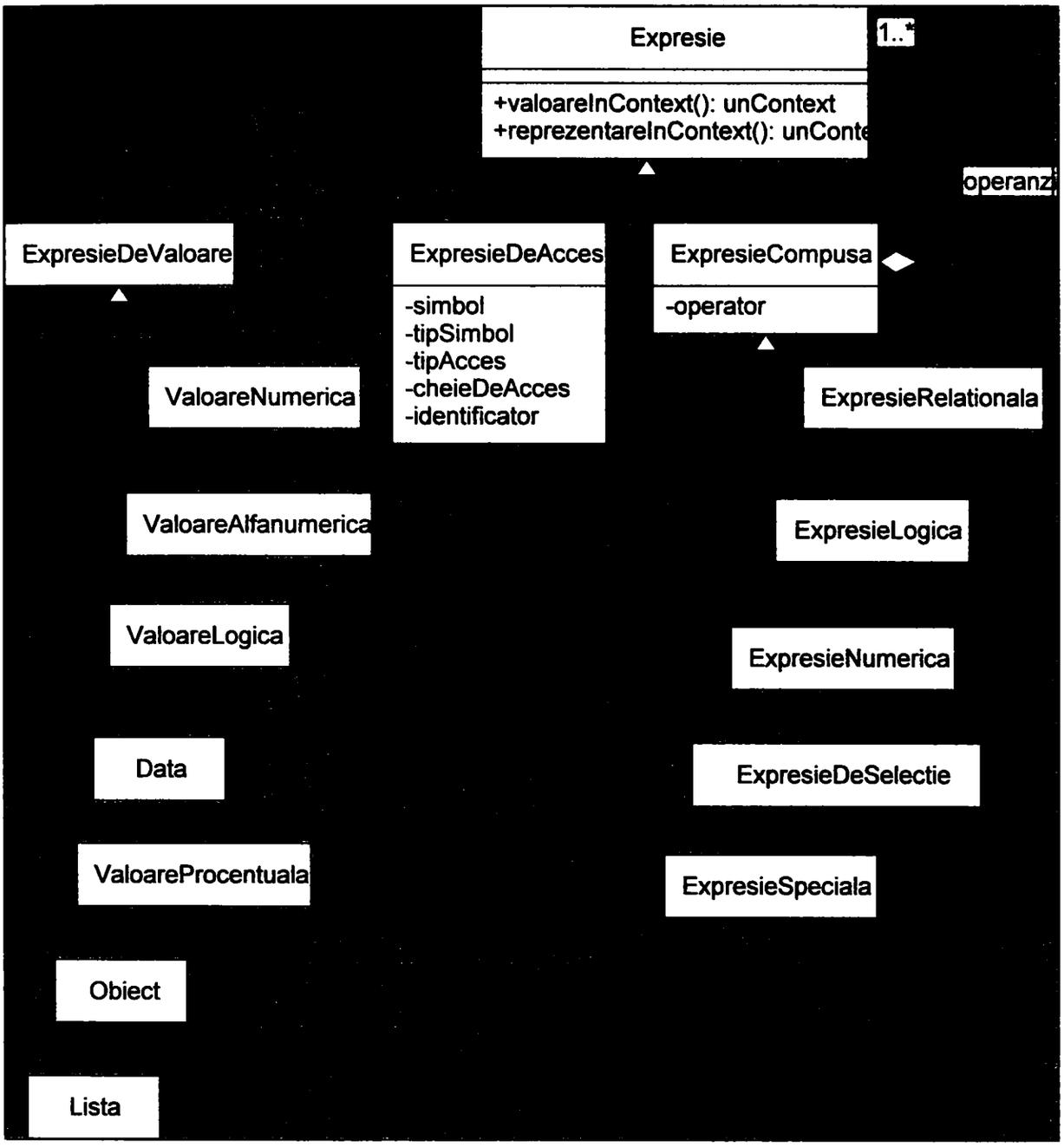


Fig. 4.10

4.5.1.7 Editorul de expresii

Editorul de expresii este o componentă software prin care se pot edita şi evalua expresii formale.

Figura 4.11 ilustreaza o modalitate de realizare a unui editor de expresii.

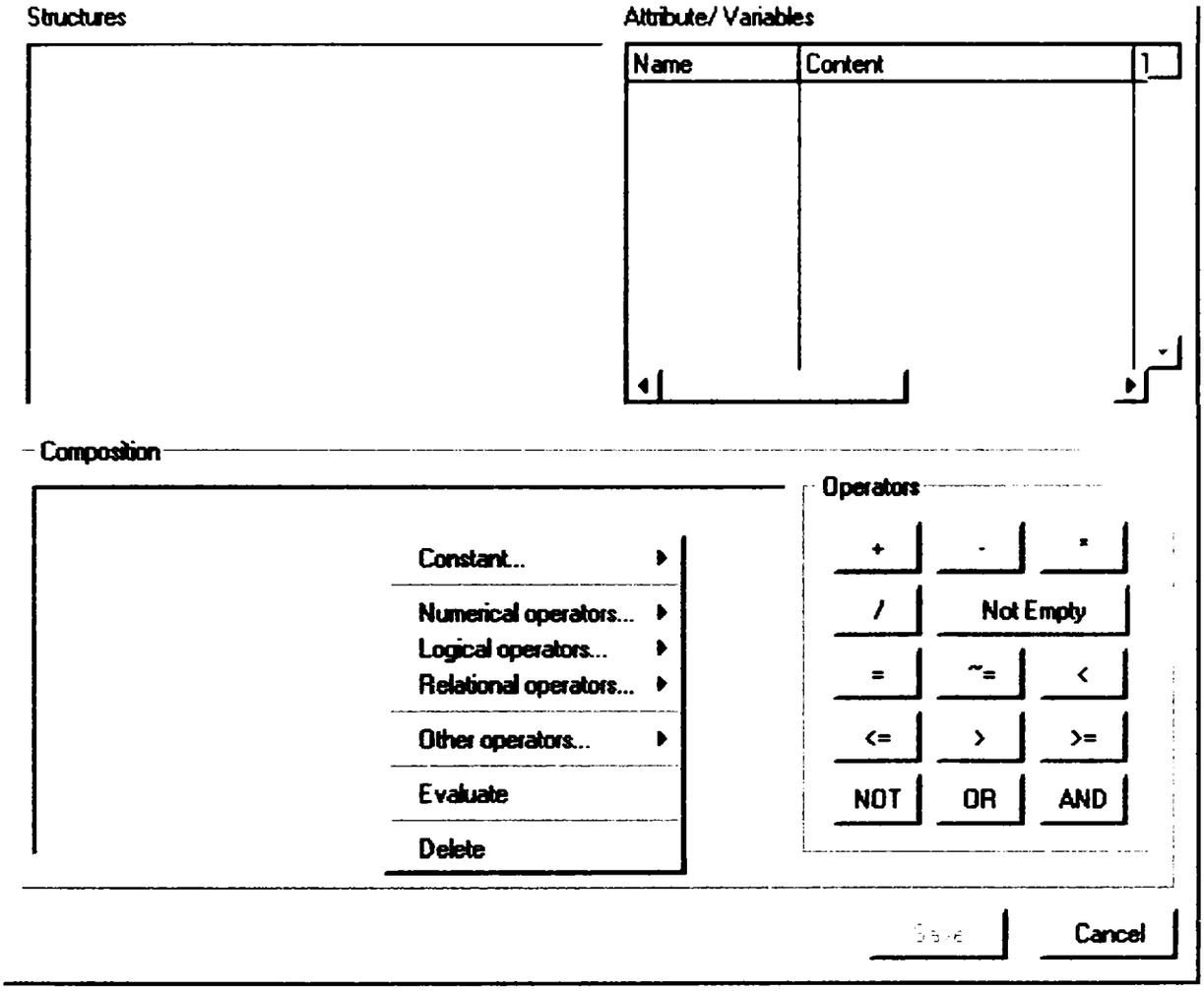


Fig. 4.11

Fereastra "Structures" conţine structura sau structurile care reprezintă contextul expresiilor (de exemplu un element de constructie concret sau un tip de element de constructie). Fereastra "Attributes/ Variables" conţine lista atributelor obiectului-context selectat. Fereastra "Composition" permite editarea şi evaluarea expresiilor. Posibilitatea evaluării imediate a expresiilor se poate exploata şi pentru interogaţii referitoare la contextul actual. Prin utilizarea expresiilor de selecţie, se pot formula interogaţii generice la nivel de tipare (şabloane).

4.5.2 Reguli

4.5.2.1 Categorii de reguli

Regulile se pot grupa în functie de destinaţia lor în: reguli pentru diferite domenii de activitate speciale, reguli de configurare, reguli pentru verificarea unor plauzibilităţi.

Regulile pentru domenii de activitate speciale se pot grupa pe domenii, de exemplu reguli pentru întocmirea documentaţiei produsului proiectat. Această categorie conţine reguli care cercetează starea actuală a contextului şi execută acţiuni într-un domeniu bine precizat (în cazul exemplului de faţă: întocmirea documentaţiei). Prin execuţia acestor reguli, starea contextului ramâne neschimbată.

Regulile de configurare pot codifica secvenţe de acţiuni care se execută automat ca parte integrantă a activităţii de proiectare. Execuţia acestor reguli poate schimba starea actuală a contextului.

Regulile de plauzibilitate cercetează starea actuală a contextului verificând corectitudinea şi integritatea acesteia, informând proiectantul în cazul detectării unor erori. Execuţia acestor reguli nu conduce la modificarea stării contextului. Este utilă protocolarea mesajelor obţinute.

Figura 4.12 reprezintă fereastra principală a unui editor de reguli (prototip).

În funcţie de categoria selectată, se afişează o listă de reguli ordonată după numere de ordine definite de către utilizator. O regulă selectată se poate prelucra în fereastra "Property" (descrierea regulei, numărul de ordine sau conţinutul).

O regulă nou creată se defineşte tot în fereastra "Property". În acest caz se va preciza şi tipul regulei.

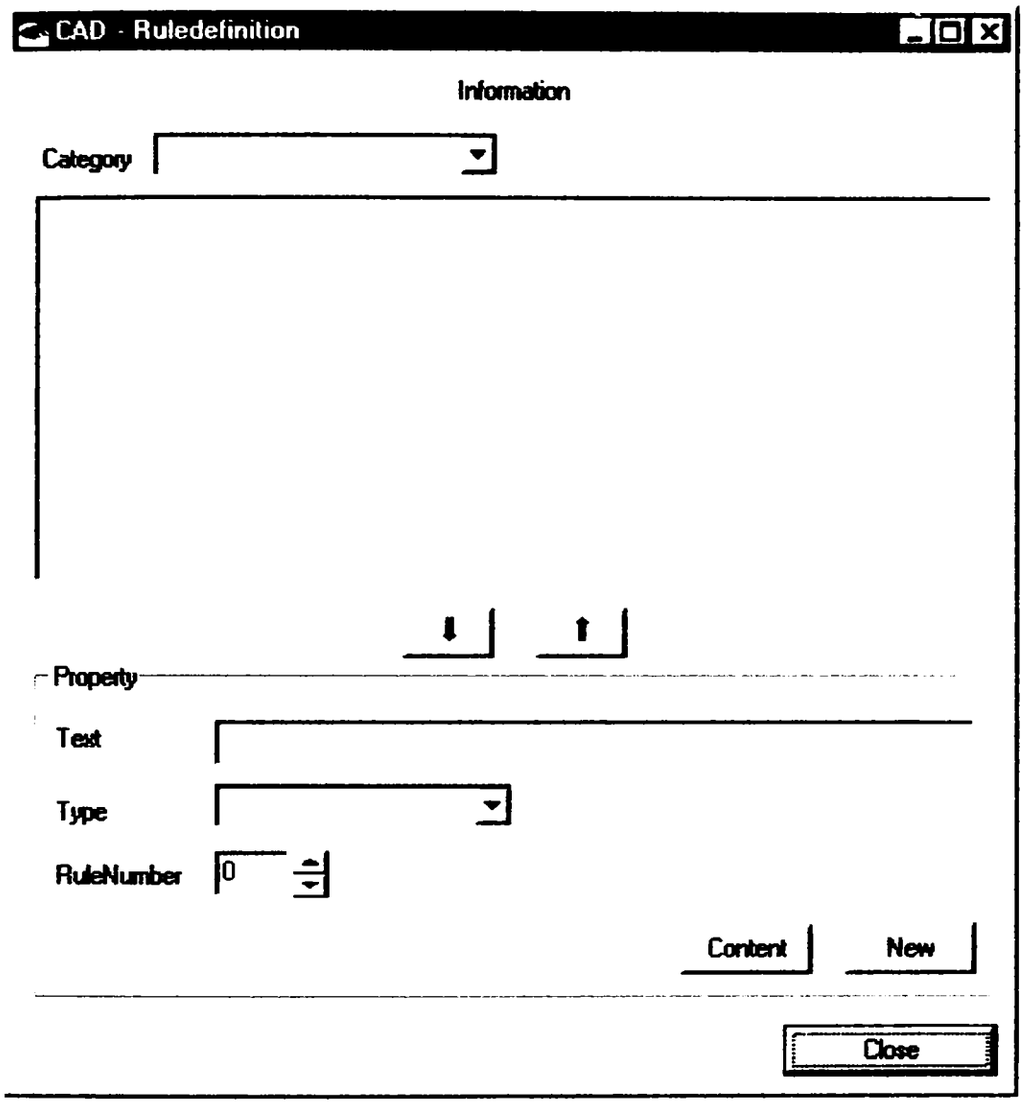


Fig. 4.12

4.5.2.2 Acţiuni simple

- Includerea sau excluderea unei componente opţionale

Prin această acţiune se poate preciza includerea sau excluderea unei anumite componente opţionale dintr-o structură.

Acţiunea are următoarea sintaxă:

Obiect inclus: ConstantaLogica

Obiectul este descris printr-o expresie de acces. ConstantaLogica poate avea una dintre valorile: true, respectiv false.

Figura 4.13 reprezintă prototipul ferestrei de editare a unei astfel de operaţii.

Un obiect sau o listă de obiecte precizate printr-o expresie generică de acces se include (activează) sau exclude (deactivează) în funcţie de valoarea logică precizată. Expresiile se editează utilizând editorul de expresii.

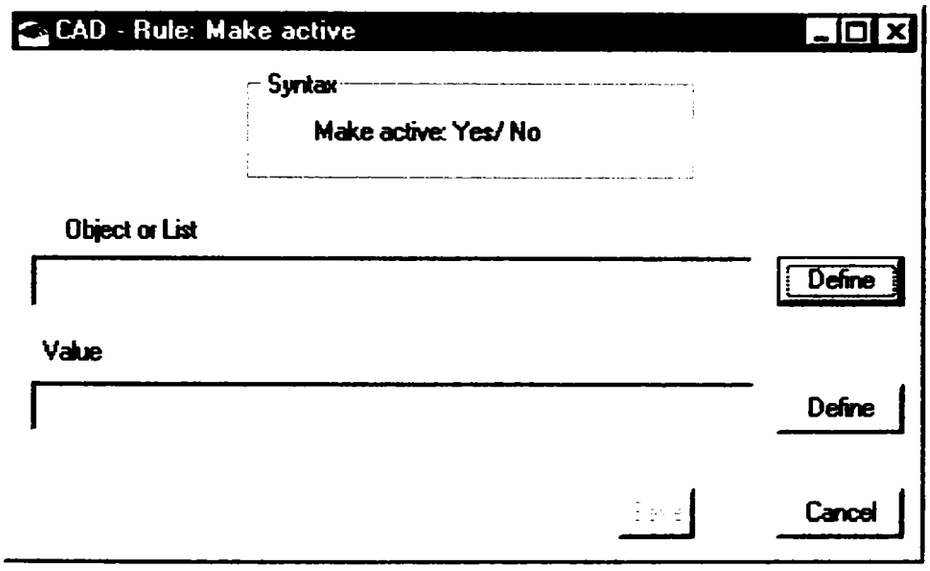


Fig. 4.13

- Atribuire

O atribuire descrie de asemenea o acţiune simplă pe baza căreia unei variabile i se atribuie o anumită valoare.
Atribuirea are următoarea sintaxă:

Atribut := Expresie

Atributul este precizat printr-o expresie de acces şi se referă la un atribut al unui obiect definit de utilizator. Expresia descrie generic valoarea pe care o va primi atributul în urma execuţiei acestei acţiuni. Tipul valorii furnizate de expresie trebuie să fie compatibil cu tipul atributului.

Figura 4.14 reprezintă prototipul ferestrei de editare a unei operaţii de atribuire.

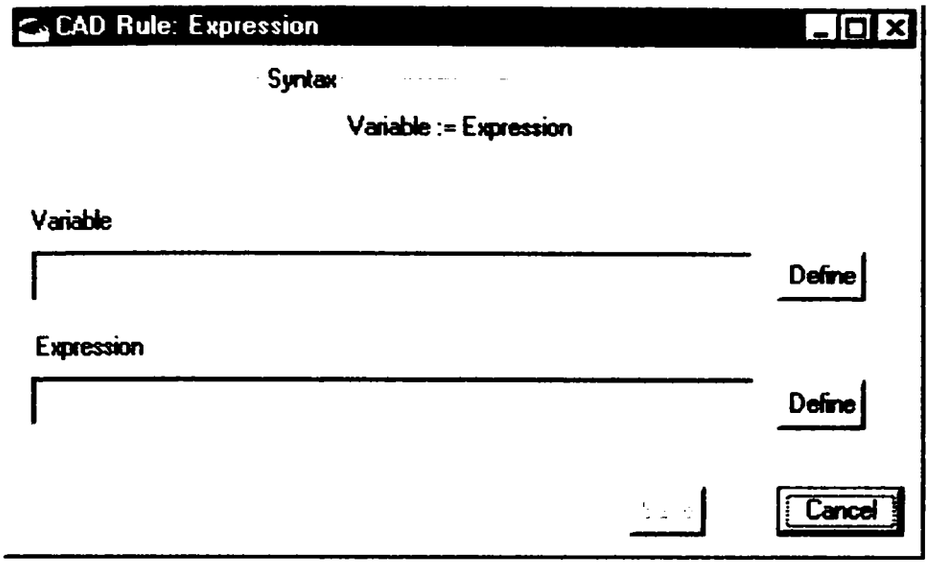


Fig. 4.14

O expresie de acces precizează formal atributul (variabila) căreia i se va atribui în timpul interpretării operaţiei valoarea precizată de expresia generică. Expresiile se editează utilizând editorul de expresii.

Implicaţii însoţite de transferul de valori

O astfel de implicaţie se defineşte pentru un anumit domeniu de activitate special (de exemplu, generarea documentaţiei) ca fiind o relaţie generică de tipul „n - 1". Implicaţia se reprezintă prin expresii de acces.

În cazul exemplului nostru (generarea de documente) faptul că obiectul X implică elementul de construcţie de tip text x înseamnă că dacă obiectul X este inclus în context, elementul de construcţie x va fi inclus în document şi va prelua valoarea atributelor cu acelaşi nume din obiectul X. În cazul în care X este o listă de obiecte, x se va include o dată pentru fiecare exemplar al listei X, actualizînd de fiecare dată valoarea atributelor preluate sau calculate pe baza unei formule matematice.

Astfel de reguli se plasează în structura unor obiecte complexe. Pentru editarea acestor reguli se utilizează editorul de expresii.

4.5.2.3 Acţiuni complexe

- IfThen

O structură de control IfThen descrie o secvenţă de acţiuni simple care se execută în funcţie de o anumită condiţie.

Structura de control are următoarea sintaxă:

If
    Condiţie
Then
    SecvenţaDeAcţiuni
End If

Condiţia este o expresie care furnizează o valoare logică. Secvenţa de acţiuni este formată din acţiuni simple.

Figura 4.15 reprezintă un prototip de editor pentru acest tip de reguli.

Conform sintaxei precizate, se defineşte condiţia logică (utilizând editorul de expresii). Lista de reguli (acţiuni simple) se defineşte cu ajutorul editoarelor de reguli anterior prezentate. Numărul de ordine al unei reguli (acţiuni) este modificabil, generându-se o nouă ordine de sortare.

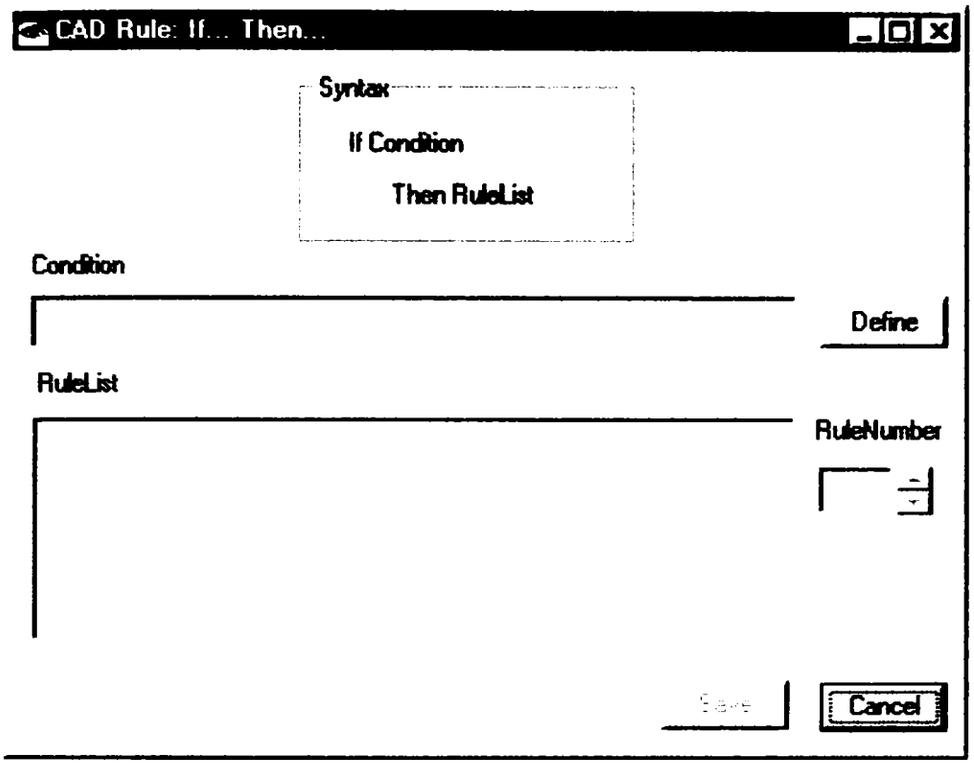


Fig. 4.15

- IfThenElse

O structură de control IfThenElse descrie două secvenţe de acţiuni simple care se execută alternativ în funcţie de o anumită condiţie.

Structura de control are următoarea sintaxă:

If

Condiţie

Then

SecvenţaDeAcţiuniThen

Else

SecvenţaDeAcţiuniElse

End If

Condiţia este o expresie care furnizează o valoare logică. Secvenţele de acţiuni sunt formate din acţiuni simple.

Figura 4.16 reprezintă un prototip de editor pentru reguli IfThenElse.

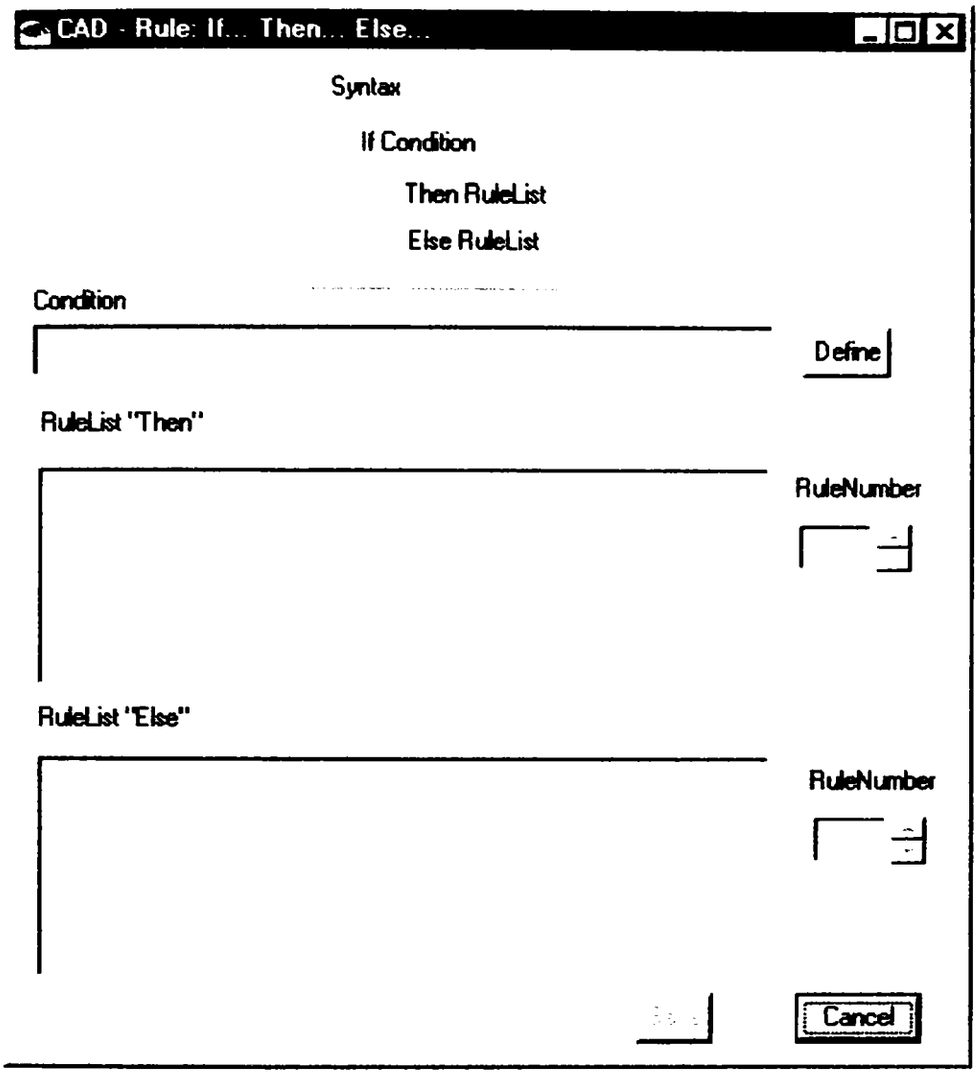


Fig. 4.16

Editarea regulei IfThenElse se realizează analog celei de tip IfThen. În plus apare doar lista de reguli (acţiuni simple) corespunzătoare ramificaţiei "Else".

- Select

O structură de control de tip Select descrie o listă de secvenţe de acţiuni condiţionate (IfThen). Execuţia acestei structuri de control presupune selectarea din listă şi execuţia primei alternative IfThen în care este îndeplinită condiţia.

Sintaxa:

Select

If

Condiţie1

Then

SecvenţaDeAcţiuni1

End If

.....

If

CondiţieN

Then

SecvenţaDeAcţiuni N

End If

End Select

Figura 4.17 reprezintă prototipul ferestrei de editare a unei reguli de selecţie.

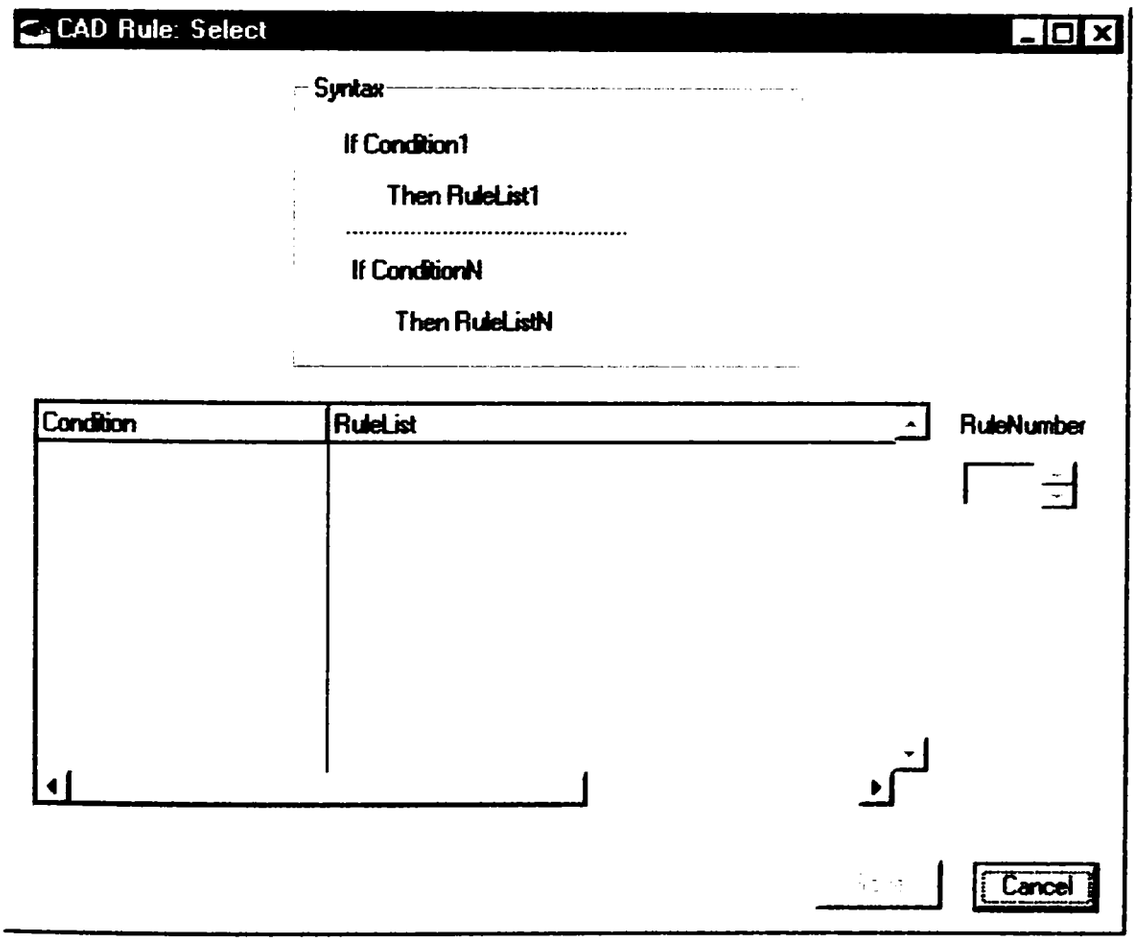


Fig. 4.17

Regulile de tipul IfThen se editează cu ajutorul ferestrelor descrise mai sus. Este posibilă modificarea numărului de ordine al regulilor din listă.

4.5.2.4 Exemplu

Categoria: Întocmirea documentaţiei

SELECT

IF

(AreValoareDefinită (componentaX) SI AreValoareDefinită (componentaY))

THEN

Activează: elementTXT_XY (1)

END IF

IF

AreValoareDefinită (componentaX)

THEN

Activează: elementTXT_X (1)

END IF

IF

AreValoareDefinită (componentaY)

THEN

Activează: elementTXT_Y (1)

END IF

END SELECT

IF

(componentaX (1). diametru > 50)

THEN

Activează: elementTXT_Diametru_peste_50 (1)

END IF

IF

(elementTXT_Diametru_peste_50 (1). activ)

THEN

Activează: elementTXT_d (1).

Deactivează: elementTXT_l (1).

END IF

4.5.2.5 Modelul de clasă

Figura 4.18 ilustrează modelul de clasă pentru reguli.

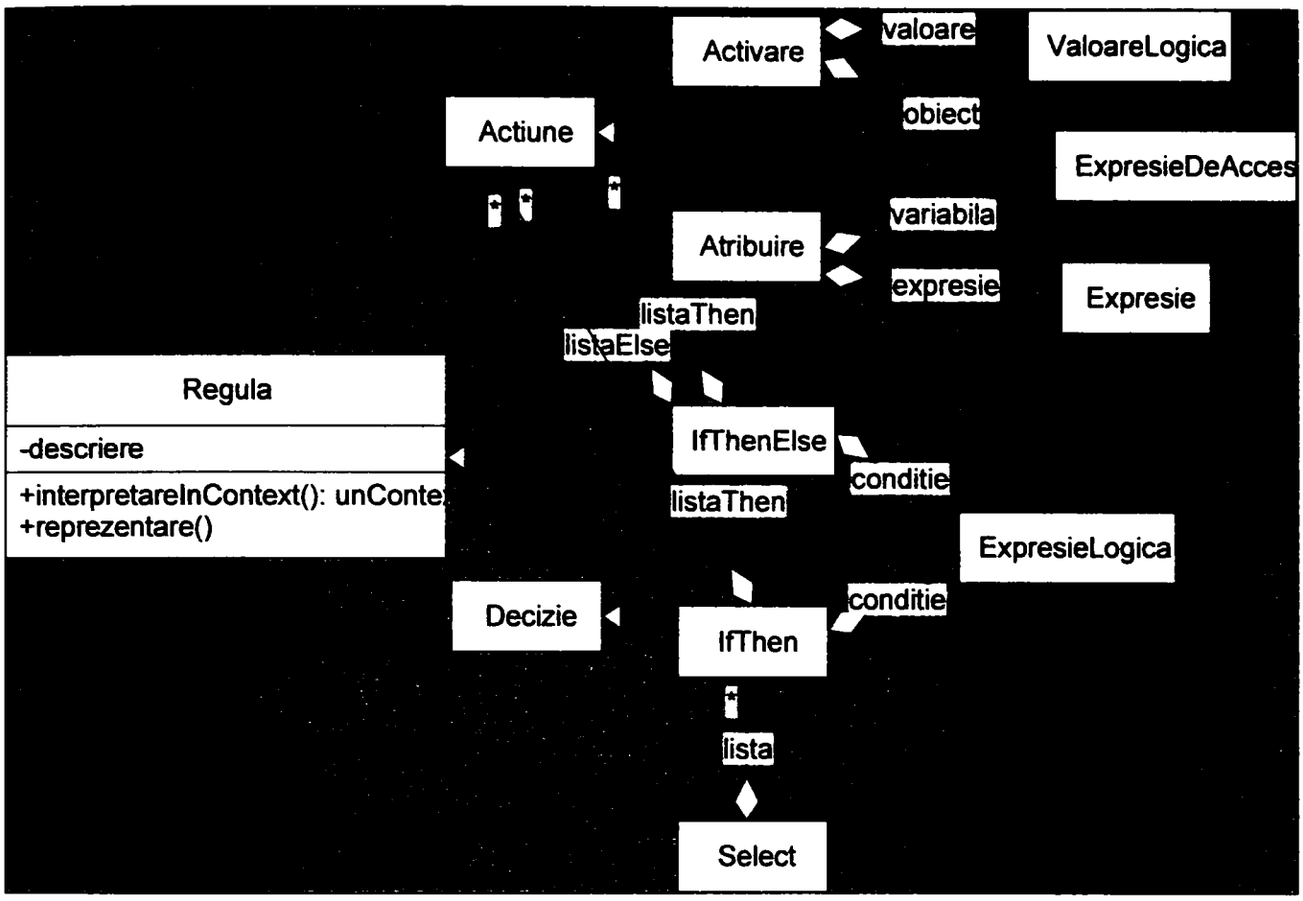


Fig. 4.18

4.5.3 Aplicaţii în rezolvarea unor probleme construcţie sau configurare

4.5.3.1 Transferul de informaţii dintre componente

O modalitate de a descrie transferul de informaţii la nivelul unei componente de cuplaj este specificarea unui obiect sursă care este răspunzator pentru transferul de valori într-un obiect destinaţie. Specificarea obiectului sursă se poate realiza printr-o expresie de acces sau de selecţie.

În acest caz nu este necesară traducerea completă a sursei, dar în prealabil trebuie aleasă o convenţie pentru transferul de informaţii (de exemplu, după nume) şi eventual o acţiune asociată transferului (de exemplu multiplicarea structurii destinaţie în cazul transmiterii unor mulţimi de parametri).

Obiectele sursă şi obiectele destinaţie trebuie să realizeze un protocol în sensul transferului de informaţii. Astfel, obiectul destinaţie va "cunoaşte" protocolul

obiectului "sursă" care trebuie să-i livreze date şi printr-un protocol stabilit va solicita informaţia dorită.

4.5.3.2 Configurarea variantelor

Mecanismele de configurare prezentate se pot utiliza în configurarea variantelor în contexte diferite:

- Configurarea unor familii şi variante de elemente de constructie,
- Configurarea unor procese automatizate (de exemplu, întocmirea documentaţiei)
- Configurarea codului pentru comanda numerică a proceselor,
- Configurarea platformelor de automatizare având o arhitectură flexibilă.

4.5.3.3 Interogaţii

Utilizatorul poate edita şi interpreta expresii formale, analog interogaţiilor din cadrul unor sisteme expert.

4.6 Cuplajul lejer dintre două sisteme

Un procedeu special de reprezentare şi traducere (transfer) a informaţiilor va asigura cuplajul confortabil dintre componentele interne şi cele externe, chiar dacă acestea lucrează cu obiecte diferite şi au protocoale diferite. Fiecare legătură cu un sistem extern se va abstractiza şi încapsula într-o componentă de cuplaj şi se poate configura intern sau în dialog (de către utilizator). Doar astfel se poate asigura integrarea unei componente într-un peisaj heterogen atât din punct de vedere technic, cât şi de specialitate.

Interacţiunea dintre două sisteme este reprezentata in figura 4.19.

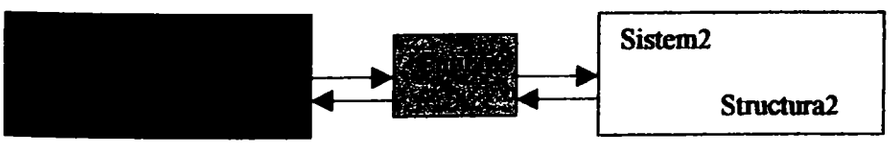


Fig. 4.19

4.6.1 Cuplajul bazat pe grafuri

Ideea care stă la baza unei componente de cuplaj este reprezentarea informaţiilor sub formă de grafuri şi definirea unor reguli de transfer a informaţiilor din graful sursă în graful destinaţie. Deoarece produsul software la care ne referim prezintă două moduri de lucru (definiţie şi aplicaţie), cuplajul va cunoaşte de asemenea două faze:

- În faza de definiţie vor fi formulate regulile pentru traducerea arborelui sursă (în realitate un graf de tip corespunzător structurii interne) în graful destinaţie.

- În faza de aplicaţie se va genera, pe baza structurii interne (de data aceasta un graf conţinând entităţi) şi regulile de traducere formulate în faza de definiţie, graful corespunzător structurii destinaţie.

Cuplajul componentelor externe se realizează prin componente de cuplaj. În acest scop, fiecare componentă trebuie să permită în faza de definiţie exportul protocolului ei în sistemul de tipuri al componentei de cuplaj, iar în faza de aplicaţie necesită un adaptor pentru sistemul de obiecte al componentei de cuplaj.

Ideea care stă la baza exemplului de mai jos este transformarea structurilor de tip graf în structuri de tip arbore şi de a "traduce" ramuri întregi dintr-un arbore în ramuri întregi al celuilalt arbore. Ramurile vor fi identificate prin secvenţa denumirilor cantelor din graful original. Deoarece modelul nostru nu exclude recursiuni la nivelul tipurilor şi nici specificarea de alternative pentru diferite construcţii, graful din faza de definiţie va descrie de fapt o mulţime de alternative de realizare pentru faza de aplicaţie. În asemenea condiţii, pentru specificarea procesului de traducere se procedează mai întâi la eliminarea recursiunilor, ceea ce conduce la obţinerea de subgrafuri nerecursive (arbori) şi apoi aducerea acestor arbori la forma normală conjunctivă.

Conform principiului de mai sus rezultă că în forma sa originală, graful va conţine noduri terminale (corespunzătoare tipurilor de date standard) şi noduri interne de tip "ŞI", precum şi de tip "SAU", dintre care unele pot fi recursive. Un nod este recursiv dacă include referinţe asupra lui însuşi, ceea ce este frecvent la nivelul descrierilor de tipuri.

Deoarece în modelul iniţial , nodurile de tip "ŞI" şi "SAU" pot apare la diferite nivele, se procedează la normalizarea structurii, ceea ce înseamnă aducerea structurii într-o formă în care un nod "SAU" va apare cel mult o dată în rădăcină, nivelele inferioare conţinând numai noduri "ŞI" sau terminale (forma normală conjunctivă).

Reguli de transformare pentru arbori:

1. Rădăcina arborelui sursă se va suprapune peste rădăcina arborelui destinaţie
2. Un nod "SAU" se va suprapune peste un nod "SAU"
3. Un nod "ŞI" se va suprapune peste un nod "ŞI"
4. Un nod terminal se va suprapune peste un nod terminal de tip compatibil
5. Un nod recursiv se va suprapune peste un nod recursiv.

În faza de specificare se vor parcurge următorii paşi:

1. Se elimină nodurile recursive din grafurile de tip (sursă şi destinaţie)
2. Se normalizează subgrafurile obţinute la punctul 1.
3. Se defineşte traducerea subgrafurilor sursă în subgrafurile destinaţie pe baza regulilor de mai sus. Pentru memorarea informaţiilor de traducere se va utiliza reprezentarea drumurilor din graf sub formă de listă de etichete (numele cantelor).

În faza de aplicaţie se vor parcurge următorii paşi:

1. Pe baza modelului (structura internă) se crează subarbori technici corespunzători tiparelor descrise de arborii de tip din faza de definiţie. Subarborii astfel obtinuţi vor servi drept sursă în procesul de traducere.
2. Se traduc subarborii de la punctul 1. Traducerea va afecta doar nodurile relevante (rădăcina şi nodurile terminale). Pentru fiecare nod relevant din sursă, se va crea un nod în arborele destinaţie. Din informaţiile din faza de specificare (punctul 3) se extrage numele nodului destinaţie nou creat. Valoarea conţinută în nodul terminal al sursei va fi transferată în nodul destinaţie. Astfel se vor obţine o mulţime de subarbori destinaţie.
3. Subarborii destinaţie se vor compune într-un singur arbore, rezultând astfel arborele destinaţie.

Pentru a putea suprapune două structuri diferite, este necesară mai întâi transformarea lor într-o formă mai generală. În continuare, se va ţine cont de următoarele principii:

1. Fiecare structură este reprezentată printr-un arbore.
2. Nodurile interne sunt fie de tip "ŞI", fie de tip "SAU". Fiecare exemplar concret al unui nod "ŞI" va avea pentru fiecare tip de succesor câte un exemplar concret. Exemplu: dacă tipul "Maşină" este compus din tipurile "Motor" şi "Caroserie", fiecare maşină concretă va avea un motor şi o caroserie concrete. Fiecare exemplar concret al unui nod de tip "SAU" va avea un singur succesor concret pentru unul dintre tipurile de succesori specificaţi. Exemplu: Proprietarul maşinii poate fi o persoană fizică sau una juridică . Fiecare maşină concretă are un singur proprietar, o persoană fizică sau una juridică.
3. Nodurile frunză ale arborelui au un tip elementar. Din mulţimea tipurilor elementare fac parte: numeric, şir de caractere, dată etc.
4. Fiecare nod are un nume.
5. Succesorii unui nod au întotdeauna nume diferite.
6. Structurile iniţiale pot fi recursive. Pentru a le aduce la structura de arbore, se va proceda la eliminarea recursiunilor şi transformarea structurilor recursive în părţi componente nerecursive. Subarborii astfel obţinuţi vor conţine noduri referinţă care vor ţine locul unor structuri detaliate într-un alt subarbore.

Modelul structurii interne în faza de specificaţie este reprezentat în figura 4.20.
Modelul structurii interne în faza de aplicaţie este reprezentat in figura 4.21.

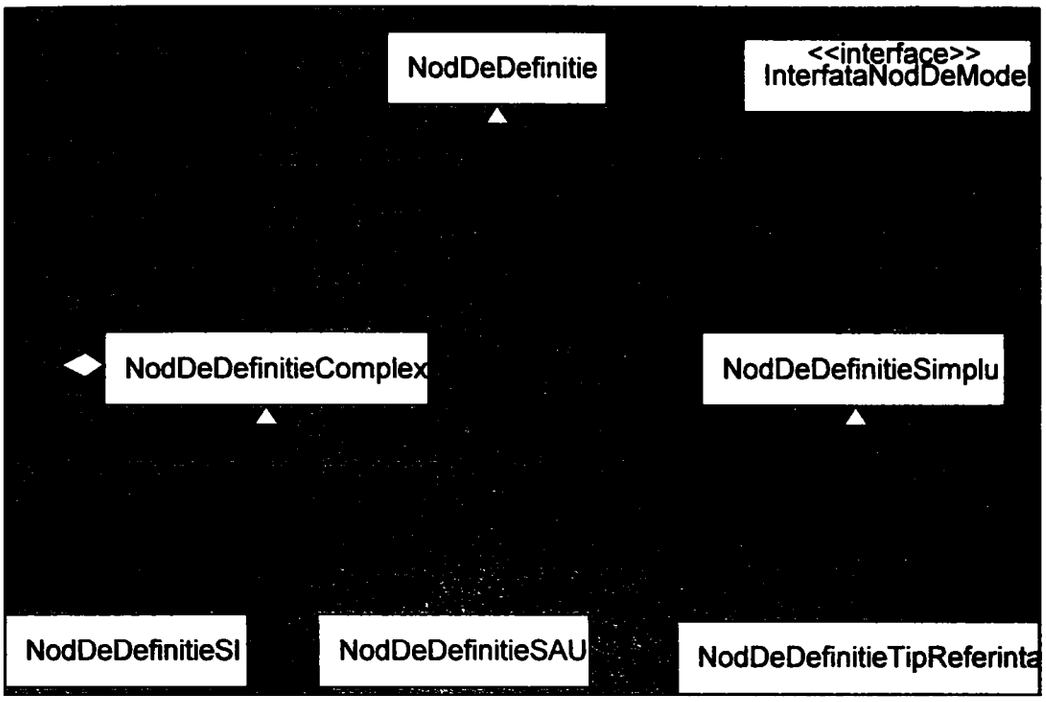


Fig. 4.20

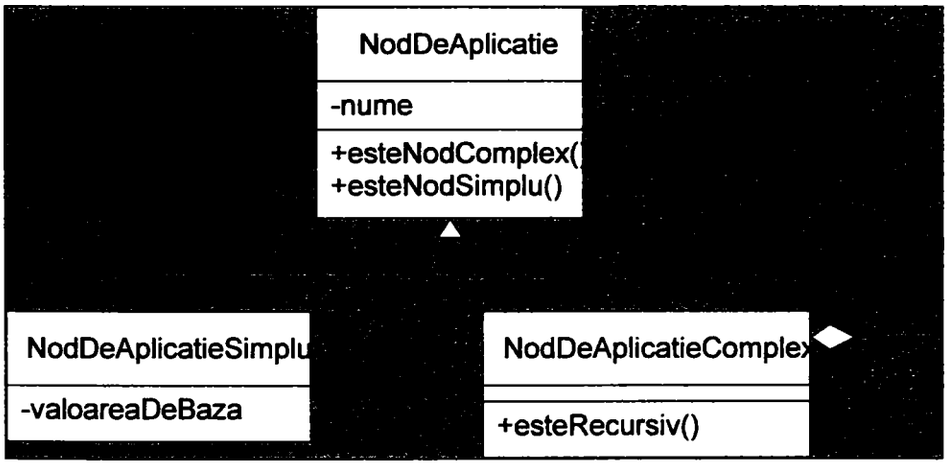


Fig. 4.21

Nodurile realizează următorul protocol:

- Fiecare nod trebuie să poată furniza numele tipului său (acelaşi ca şi în faza de specificaţie). Aceasta se va realiza printr-un accessor numit "nume".
- Fiecare nod trebuie să poată furniza felul tipului său. Aceasta se va realiza prin metodele "*esteNodComplex*" şi "*esteNodSimplu*".
- Un nod complex trebuie să implementeze metoda "*successori*", care furnizează lista descendenţilor nodului respectiv.
- Un nod terminal (simplu) trebuie să implementeze metoda "*valoare*" care furnizează valoarea asociată nodului respectiv.

- Un nod complex va implementa metoda "*esteRecursiv*" sau o altă modalitate de a obţine această informaţie (tip recursiv).

Arborele sursă în faza de aplicaţie

Arborele sursă în faza de aplicaţie va fi compus din mai mulţi subarbori şi se modelează conform figurii 4.22.

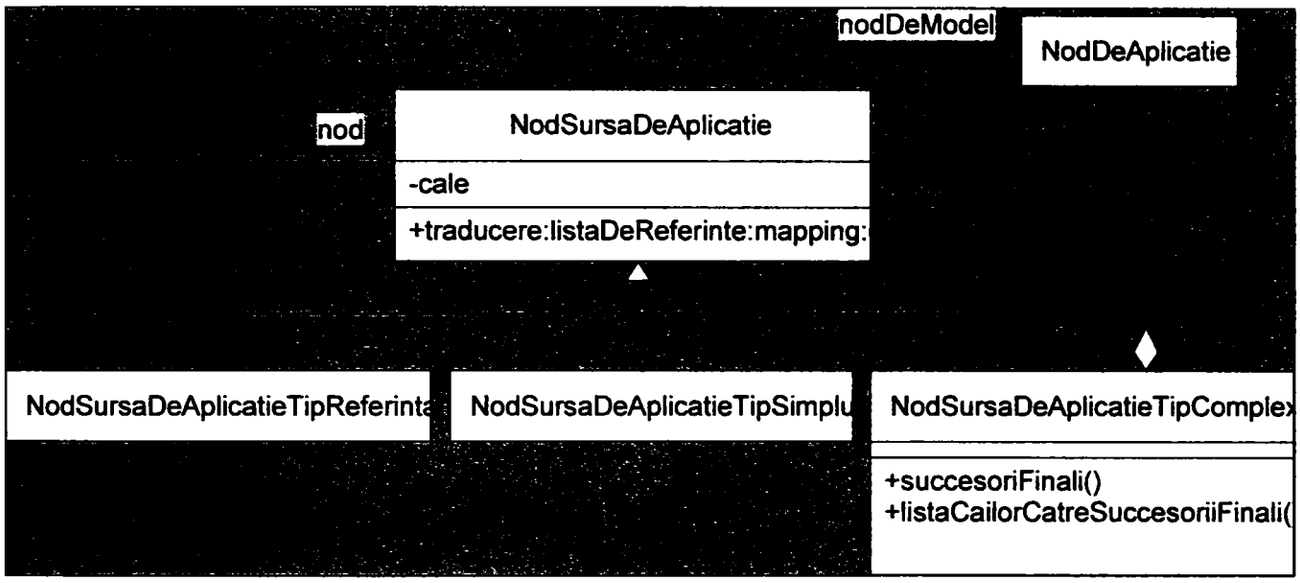


Fig. 4.22

Subarborii vor fi compuşi prin intermediul nodurilor referinţa corespunzător specificaţiei.

Obiectele din structură sunt în măsura de a se traduce (vezi metoda *traducere:listaDeReferinte:mapping:*). Pentru traducere se va ţine cont de regulile de transformare (mapping) existente. Lista nodurilor referinţă va servi construcţiei arborelui final din părţile componente (subarbori).

Arborele destinaţie

Arborele destinaţie se va modela conform figurii 4.23.

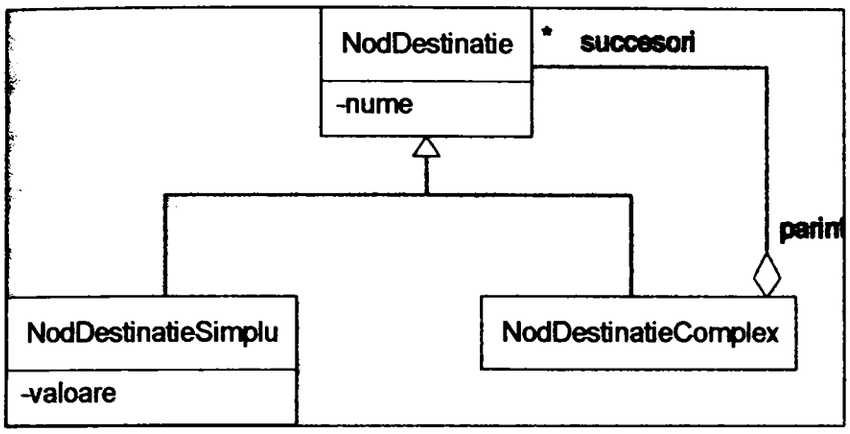


Fig. 4.23

Nodurile vor respecta următorul protocol:

- Fiecare nod îşi cunoaşte numele (*nume*).
- Nodurile simple (frunză) vor furniza valoarea (*valoare*).
- Nodurile complexe furnizează lista succesorilor lor (*successori*).

4.6.2 Exemplu

4.6.2.1 Faza de definiţie

Graful sursă este reprezentat in figura 4.24.

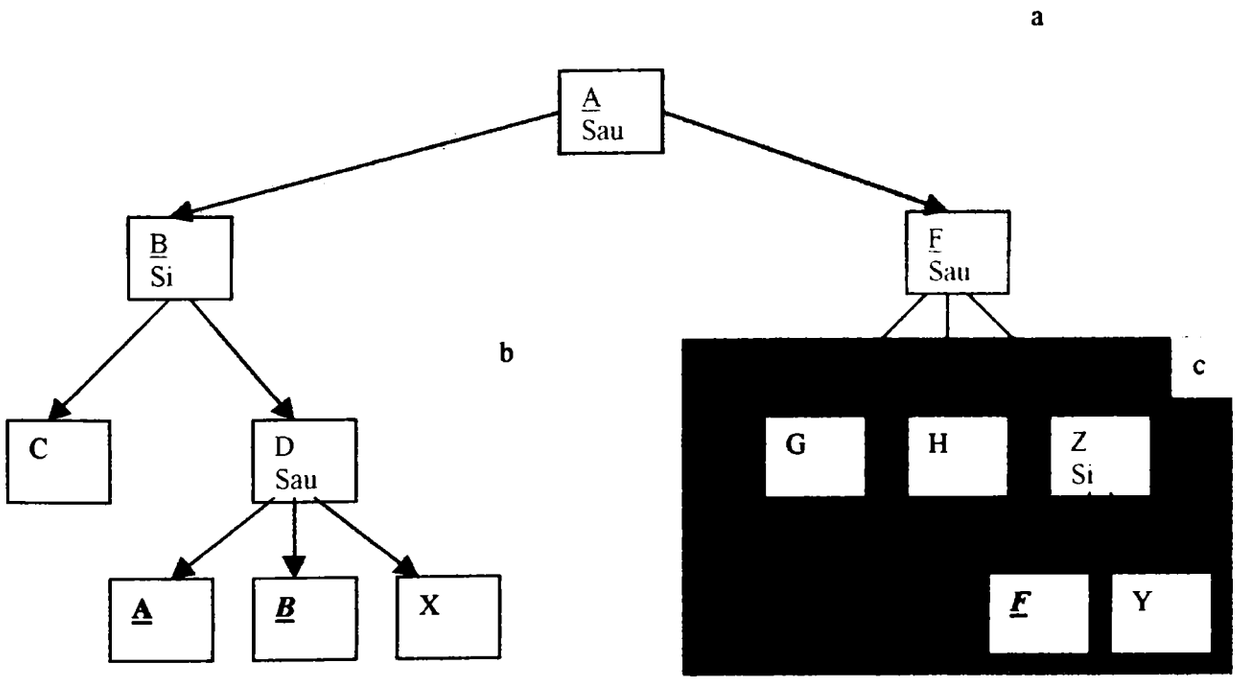


Fig. 4.24

A = [(B), (F)]

B = [(C, A), (C, B), (C, X)]

F = [(G), (H), (F, Y)]

Altfel exprimat:

A = [(AB), (AF)]

B = [(BC, BDA), (BC, BDB), (BC, BDX)]

F = [(FG), (FH), (FZF, FZY)]

Noduri recursive: A, B, F.

Subgrafuri nerecursive (arbori): a, b, c.

Graful destinaţie in care trebuie transformat graful sursa este reprezentat in figura 4.25.

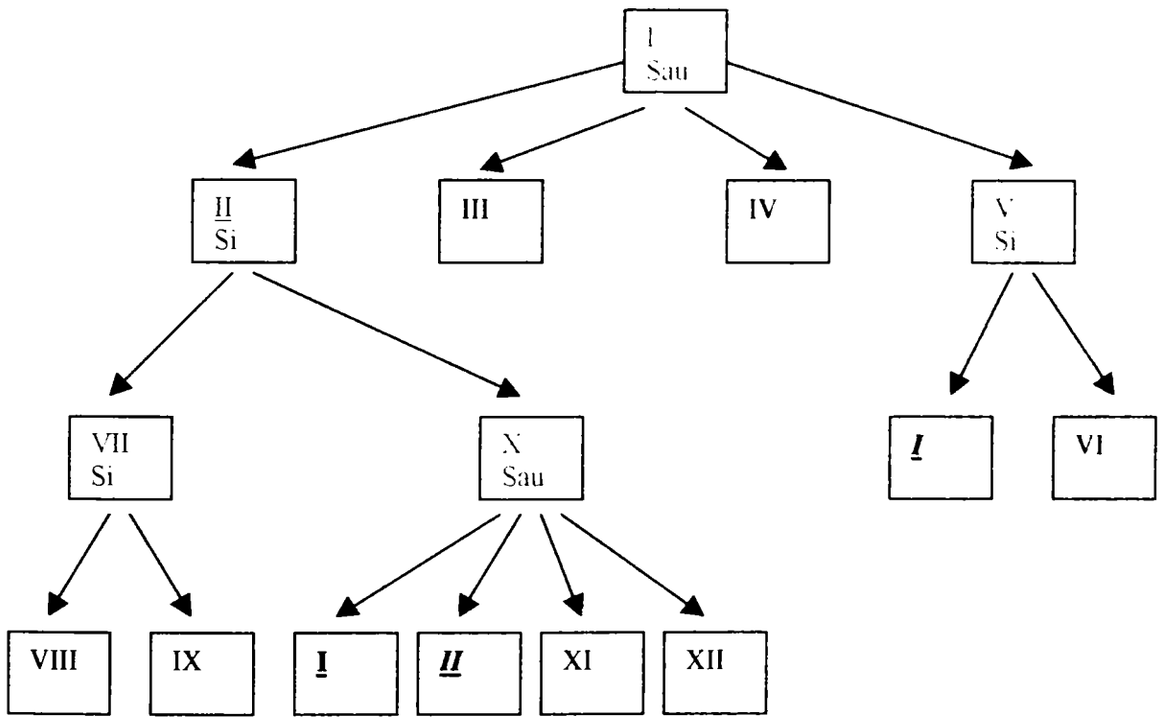


Fig. 4.25

I = [(II), (III), (IV), (I, VI)]

II = [(VIII, IX, I), (VIII, IX, II), (VIII, IX, XI), (VIII, IX, XII)]

Noduri recursive: I, II.

Reguli de traducere

(A) → (I)

Varianta 1

(B) → (II)

Varianta a 2-a

(F) → (I)

(B) → (II)

Varianta 1

(C) → (VIII)

(A)→ (I)

Varianta a 2-a

(C) → (IX)

(A)→ (II)

Varianta a 3-a

(A)→ (VIII)

(X) → (IX)

(F) → (I)

Varianta 1

(G) → (III)

Varianta a 2-a

(H) → (IV)

Varianta a 3-a

(F) → (I)

(Y) → (VI)

4.6.2.2 Faza de aplicaţie

Transformarea grafului sursă in graful destinatie este reprezentata in figura 4.26.

Paşi de lucru:

- Obţinerea arborelui sursă,
- Împărţirea arborelui sursă în subarbori nerecursivi conform specificaţiei (tiparele: "a", "b", "c"),
- Traducerea separată a subarborilor.

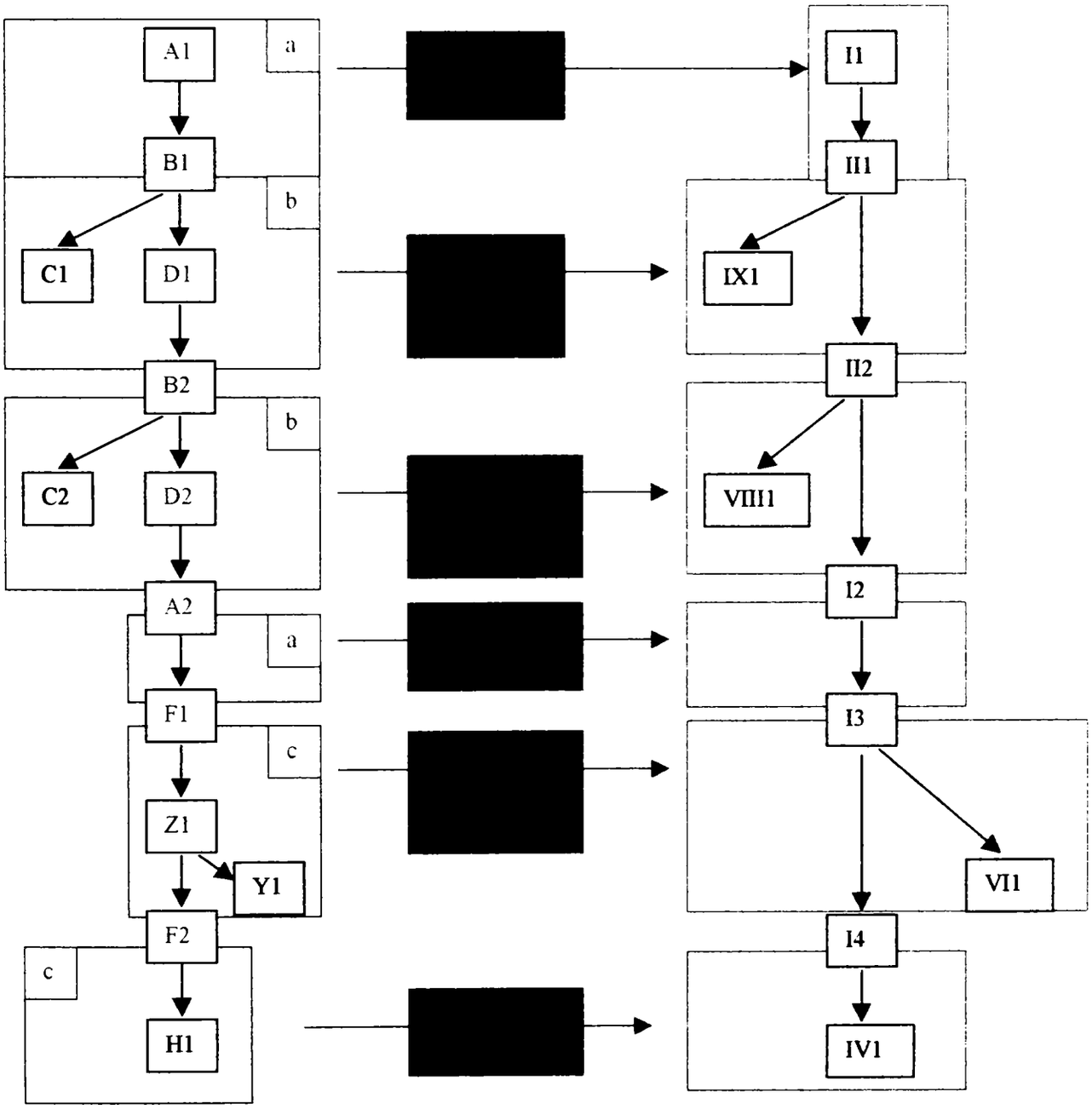


Fig. 4.26

4.6.3 Exemplu concret de cuplaj de componente

Specificaţia

În etapa de specificare se configurează transformarea structurii interne în structura externă.

Structura externă corespunde unui document, deci, o structură conţinând obiecte ale sistemului TEXT, dar încă nu putem vorbi despre un document concret în această fază. Această structură poate conţine şi părţi variabile sau alternative (elemente locţiitoare), care se concretizează în mod diferit în documentele concrete.

Structura externă se construieşte cu ajutorul componentei administrative a sistemului de prelucrare de texte TEXT şi se va exporta în componenta de cuplaj care permite definirea regulilor de transformare dintre structurile sursă şi destinaţie.

Exemplu

Să considerăm ca punct de plecare un tip de element de constructie simplu (vezi figura 4.27). Tipul de element de constructie X va conţine componentele Fabricant şi Maşină, definite ca tipuri.

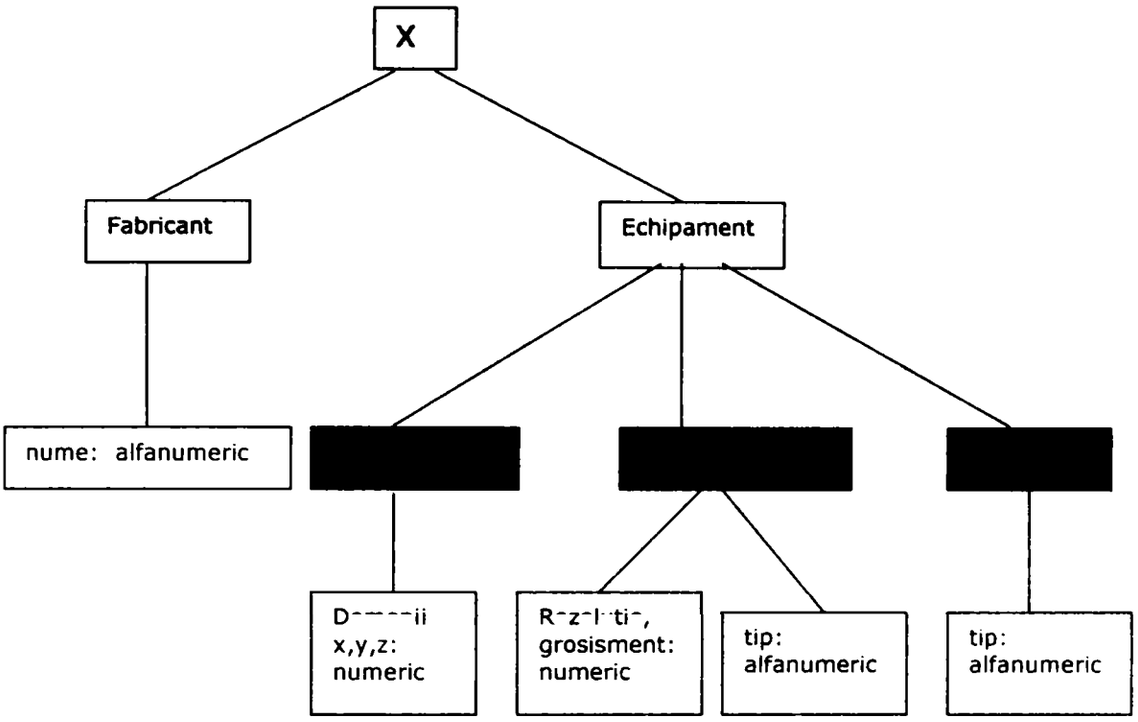


Fig. 4.27

Textul care va trebui generat pentru fiecare echipament în parte este reprezentat in figura 4.28.

Datele fabricantului: <<<nume>>>

Datele echipamentului optic:
Grosismentul sistemului optic: <<<Γ>>>.
Rezolutie: <<<R>>>.
Obiectiv <<Aβ>>:
Ocular: <<<f',Γoc>>>.

Fig. 4.28

Din analiza textului rezultă următoarele elemente de text diferite:

- T1: "*Datele fabricantului: <<<nume>>>*"
- T2: *"Datele echipamentului optic:"*
- T3: *" Grosismentul sistemului optic: <<< Γ >>>."*
- T4: *"Rezolutia sistemului optic: <<<R>>>."*
- T5: *"Obiectiv:"<<tip>>*
- T6: *"Ocular: <<<tip>>>."*

Structura documentului este reprezentata in figura 4.29.

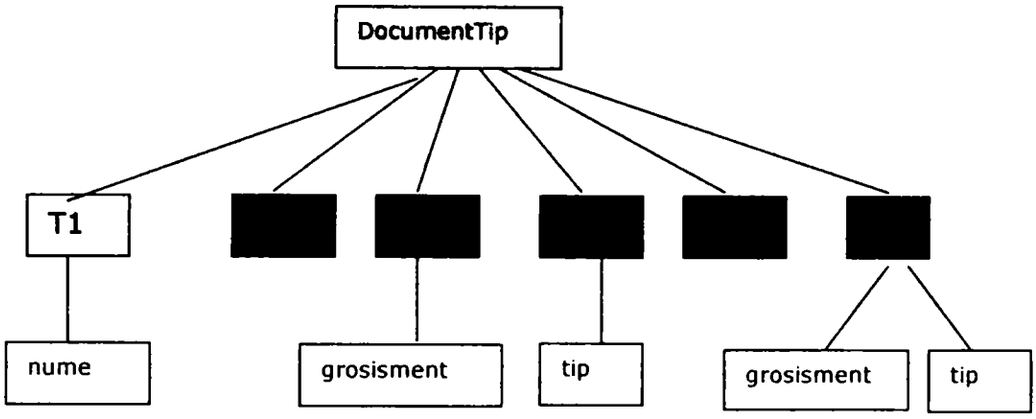


Fig. 4.29

Regulile de transformare pentru cele două structuri (internă, respectiv externă) sunt exemplificate în figura 4.30. Se poate observa că se suprapun drumuri ale grafului structurii interne peste drumuri ale grafului structurii externe. Deoarece atributele, respectiv variabilele se reprezintă prin frunze ale structurilor de arbore, componenta de cuplaj poate prelua şi transportul valorilor din structura internă spre cea externă.

```
X/ Fabricant/ nume ->
        DocumentTip/ T1/ nume
X/ Echipament/Sistem optic/grosisment ->
        DocumentTip/ T3/ grosisment
X/ Echipament/Sistem optic/tip ->
        DocumentTip/ T4/ tip
...
```

Fig. 4.30

Aplicaţia

În faza de aplicaţie se generează o structură de date de tip arbore utilizând structura internă a obiectului concret, precum şi regulile de transformare şi se exportă în sistemul TEXT. Aceasta se realizează automat prin componenta de cuplaj.

Cu ajutorul componentei TEXT "Compoziţie", este posibilă prelucrarea ulterioară a documentului astfel obţinut (de exemplu, adăugarea de elemente

suplimentare, eliminarea unor elemente existente), precum şi vizualizarea sub formă de document Word şi memorarea documentului.

Exemplu

În continuare vom considera ca punct de plecare rezultatele exemplului pentru faza de specificare: obiectul abstract X împreuna cu regulile de transformare.

Definim un element de constructie concret pe baza tipului sau (analog procesului instanţierii unui obiect pe baza unei clase), având structura reprezentată în figura 4.31.

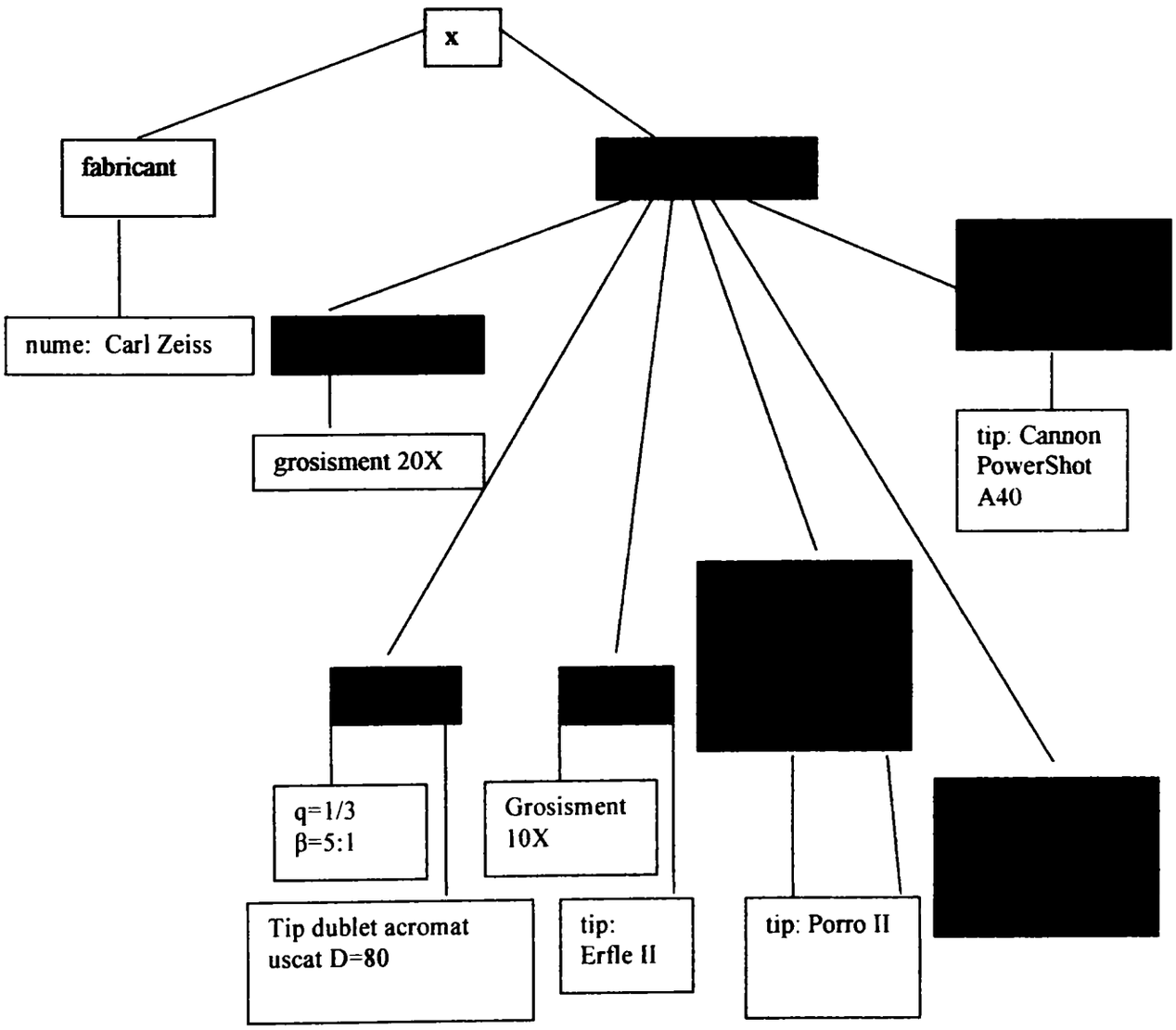


Fig. 4.31

Documentul corespunzător obiectului concret se poate crea automat, pe baza structurii interne şi a regulilor de traducere (figura 4.32).

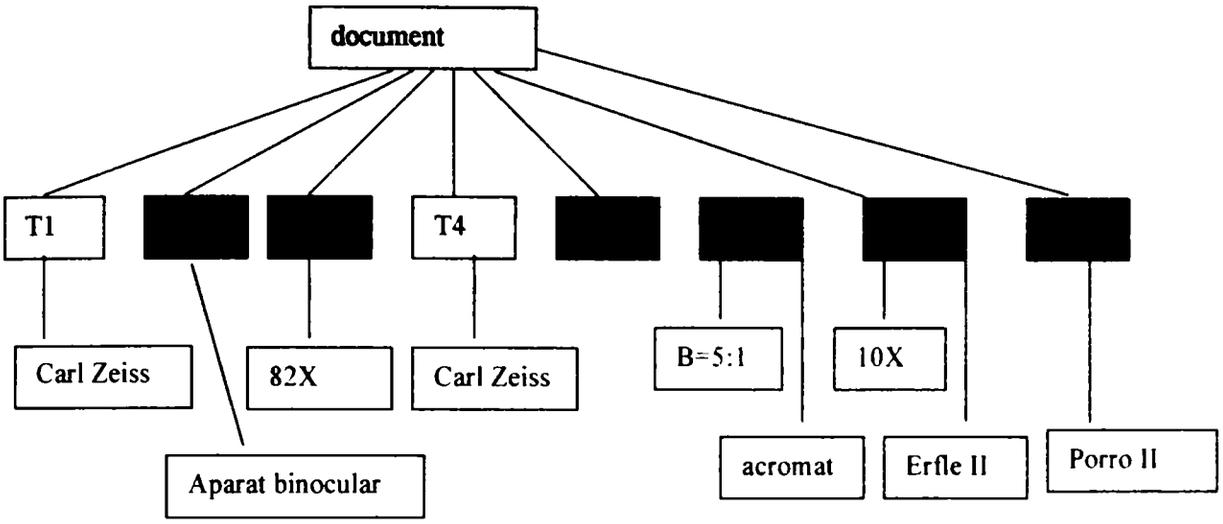


Fig. 4.32

Documentul concret este reprezentat în figura 4.33 şi se poate prelucra ulterior utilizând componenta "Compoziţie" a sistemului TEXT.

Datele fabricantului: Carl Zeiss

Datele echipamentului optic
Sistem optic: afocal
Grosisment 10X
Obiectiv q=1/3
Ocular Erfle II
Γ=20X
Aparat de fotografiere digitala Cannon
Sistem de redresare Porro II

Fig. 4.33

4.6.4 Aplicaţii în rezolvarea unor probleme de construcţie sau configurare

1. Transferul de informaţii dintre componente

O modalitate de a descrie transferul de informaţii la nivelul unei componente de cuplaj este specificarea regulilor de transformare necesare transferului de valori dinspre structura sursă către structura destinaţie.

Componentele cuplate astfel trebuie să realizeze un protocol în sensul transferului de informaţii. Astfel, componenta sursă trebuie să fie în măsură să exporte structura sa internă, iar componenta destinaţie trebuie să poată importa forma tradusă a structurii sursă pe baza unui protocol stabilit.

2. Configurarea variantelor

Mecanismele de configurare prezentate se pot utiliza în configurarea variantelor în contexte diferite:

- Configurarea unor familii şi variante de produse,
- Configurarea unor procese automatizate (de exemplu, întocmirea documentaţiei)
- Configurarea codului pentru comanda numerică a proceselor,
- Configurarea platformelor de automatizare având o arhitectură flexibilă.

Experienţa din diferite proiecte demonstrează ca este posibilă realizarea unui cuplaj lejer dintre componente. Trebuie ţinut cont însă de faptul că volumul informaţiilor referitoare la cuplaj, precum şi timpul de execuţie al programului depind de modelul folosit şi complexitatea operaţiilor de realizat.

Deoarece fiecare componentă de cuplaj va conţine şi acţiuni specifice domeniului de destinaţie, se va alege o astfel de arhitectură, care separă părtile generalizate de cele speciale (divizarea componentei). S-ar putea realiza un cadru global de cuplaj (framework) cu structură şi protocol "preconfecţionate" pentru a asigura respectarea riguroasă a principiilor cuplajului lejer, precum şi a realizării cuplajului într-o manieră unitară.

4.7 Gestionarea evolutiei istorice a componentelor

Procesele de configurare şi construcţie a unor produse (elemente de constructie abstracte sau concrete) sunt procese de lungă durată. De multe ori lucreaza simultan mai multe persoane asupra unor componente diferite din sistem. Din acest motiv este indicată utilizarea unui concept adecvat de reprezentare a evolutiei istorice a componentelor şi imbinarea rezultatelor in cadrul prelucrarii componentelor de catre mai multi utilizatori ai sistemului CAD.

Conceptul de istorie propus în lucrarea de faţă se bazează pe următoarele elemente:

- Toate atributele modificabile vor fi extrase din modelele statice şi vor fi gestionate sub forma unor stări ale obiectelor statice. Astfel, un obiect istoric va poseda mai multe stări (conform tiparului "State").
- Selectarea unei stări istorice a unui obiect se realizează pe baza unei date calendaristice numite valabilitate
- Mulţimea de stări care caracterizează o structură istorică complexă (de exemplu un tip de element de constructie) la un moment dat se poate filtra din mulţimea tuturor stărilor printr-un construct special. Acest construct va fi numit versiune în cazul în care se doreşte îngheţarea mulţimii stărilor care caracterizează obiectul la un moment dat, respectiv, ediţie, în cazul în care se permite modificarea mulţimii stărilor.
- Prelucrarea unei versiuni ale unui obiect istoric se realizează prin deschiderea unei noi ediţii.
- Este recomandabilă introducerea noţiunii de release care reprezintă o versiune închisă şi "dată în producţie" a configuraţiei unui tip de element de constructie. Crearea de elemente de constructie concrete se va desfăşura doar pe baza unui release al tipului aferent (excepţie de la această regulă fac scopurile de test).

Modelul de istorie descris mai sus este reprezentat în figura 4.34.

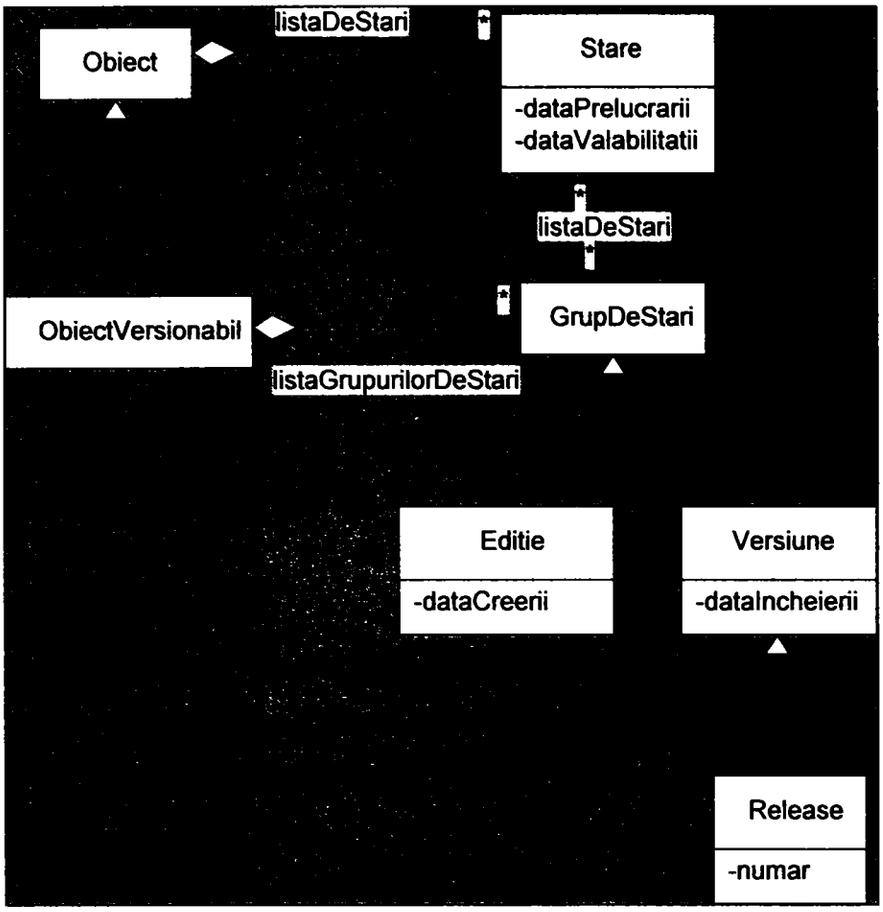


Fig. 4.34

4.8 Concluzii

Pe baza celor prezentate, se pot formula câteva concluzii:

- Notiunile de clasa si instanta din teoria orientata pe obiecte se pot utiliza cu succes si in activitatea de proiectare asistata de calculator,
- Metodele de analiza si design orientate pe obiecte sunt, de asemenea, utile in activitatea de proiectare asistata de calculator,
- Utilizarea unei arhitecturi moderne centrata pe componente asigura flexibilitatea si adaptabilitatea sistemelor CAD,
- Specializarea activitatilor in macrodesign si microdesign conduce la cresterea eficientei in activitatea de proiectare prin utilizarea in faza de microdesign a unor tipare preconfigurate in faza de macrodesign,
- Specializarea activitatilor in macrodesign si microdesign conduce la cresterea calitatii in activitatea de proiectare prin utilizarea tiparelor si a unor solutii standard.
- Printr-o arhitectura bazata pe componente si posibilitatea de configurare se pot inlantui procese diferite apartinand mai multor etape pe proiectare si constructie,

- Evolutia istorica in procesul de proiectare se poate memora printr-un framework adecvat.

Intregul capitol, inclusiv figurile, se caracterizeaza prin originalitate, reprezentand contribuţii ale autoarei, o parte dintre ele fiind publicate sau aplicate in practica in cadrul proiectelor realizate in cadrul concernului german HDI unde autoarea isi desfasoara activitatea ca arhitecta si conducatoare de proiecte. Printre acestea se pot enumera următoarele:

- Analiza, design-ul si implementarea componentelor prezentate,
- Aplicarea metodelor de proiectare orientate pe obiecte,
- specializarea activitatilor de proiectare corespunzator fazelor macro- si macrodesign,
- domeniul proiectarii de variante,
- un limbaj de configurare a solutiei in faza de macrodesign,
- cuplajul componentelor corespunzator activitatilor specifice mai multor etape de proiectare si constructie, inclusiv algoritmul de transformare a structurilor,
- gestionarea istoriei atat in faza de design cat si in faza de microdesign.

5 Exemplu de proiectare automată a sistemelor optice afocale

5.1 Particularităţi ale echipamentelor bazate pe sisteme optice

5.1.1 Generalitati

Din mulţimea posibilă a tipurilor de echipamente de mecanică fină, cum ar fi echipamente de comandă, control, măsurare, observare etc., în continuare, sunt vizate cele care incorporează în structura lor sistemele optice, ca principale subansambluri funcţionale.

Echipamentele optice sunt ominprezente în peisajul tehnic actual. Destinaţia lor este foarte diversă şi, funcţie de aplicaţie, reprezintă ansambluri de sine stătătoare sau intră în componenţa unor echipamente complexe. Utilizarea subansamblurilor optice permite realizarea următoarelor funcţii:

- observarea obiectelor îndepărtate, a celor de dimensiuni microscopice sau a detaliilor (lupa, luneta, microscopul de uz general),
- măsurarea cu precizie ridicată a lungimilor (lupe, lunete, microscoape de măsurare, telemetre),
- formarea imaginilor asemenea obiectelor pe suprafaţa unor receptori fizici - ecran, placa foto, film – (aparate de proiecţie, aparate foto sau de filmare),
- formarea unor figuri luminoase care informează asupra unor caracteristici ale obiectelor (măsurarea rugozităţii prin microscopie de interferenţă, studiul mediilor transparente cu indici de refracţie diferiţi etc.),
- generarea semnalelor luminoase în scopul convertirii acestora în semnale electrice (traductori incrementali de deplasare, viteză etc., obţinerea imaginilor digitale utilizând radiaţia vizibilă sau infraroşie cu camere CCD, aparate de observare pe timp de noapte, dispozitive de imagerie medicală etc.),
- transmiterea semnalelor optice la distanţe mari (cabluri optice utilizate în telecomunicaţii),
- obţinerea spectrelor de emisie sau absorbţie ale structurilor materiale (spectrofotometre de emisie sau absorbţie),

- ghidarea şi focalizarea fasciculelor laser (cu utilizare în industriile prelucrătoare, în medicină etc.).

În principiu, se vor avea în vedere modelele de echipamente optomecanice, pentru care s-au constituit baze de date şi o parte din programele de calcul aferente, ca aplicaţii.

Indiferent de structura echipamentului optomecanic, optoelectric sau optoelectronic, acestuia i se stabileşte schema optică, pe baza căreia urmează să se proiecteze sistemul optic asociat.

Calculul optic se deosebeşte de celelalte calcule tehnice prin câteva trăsături specifice:

- caracteristicile optice ale elementelor din schema optică se determină cu maximă acurateţe (spre deosebire de algoritmii de proiectare din alte domenii, în care se admit rotunjiri, valori acoperitoare, diverşi factori, care nu influenţează funcţia piesei proiectate).
- obţinerea unui sistem optic performant presupune minimizarea aberaţiilor geometrice şi cromatice, respectiv desfăşurarea unui calcul de optimizare, aplicat unor funcţii cu un număr mare de variabile şi care se poate derula numai iterativ.
- nu există metode globale de abordare a sistemelor optice, ci algoritmi de tratare din aproape în aproape, pe şirul de dioptri care constituie sistemul. Rezultă un volum de calcul foarte mare.
- nu s-au elaborat metode standardizate de calcul al componentelor, subansamblurilor sau sistemelor optice, aşa cum există în alte domenii. Calculul de gabarit şi, mai ales, sinteza şi optimizarea componentelor dau rezultate care depind mult de abilitatea şi experienţa proiectantului.

Având în vedere cele arătate mai sus, rezultă că proiectarea în domeniul opticii reclamă un calcul laborios, de mare acurateţe şi rafinament matematic. Toate aceste trei aspecte pledează pentru oportunitatea introducerii calculului automat. Performanţele actuale ale sistemelor de calcul şi limbajele de programare existente pun la dispoziţia opticianului instrumentele necesare automatizării calculului şi obţinerii unor soluţii cu caracteristici superioare, în timp redus.

Schema logică cea mai generală de proiectare a sistemelor optice este prezentată în figura 5.1.

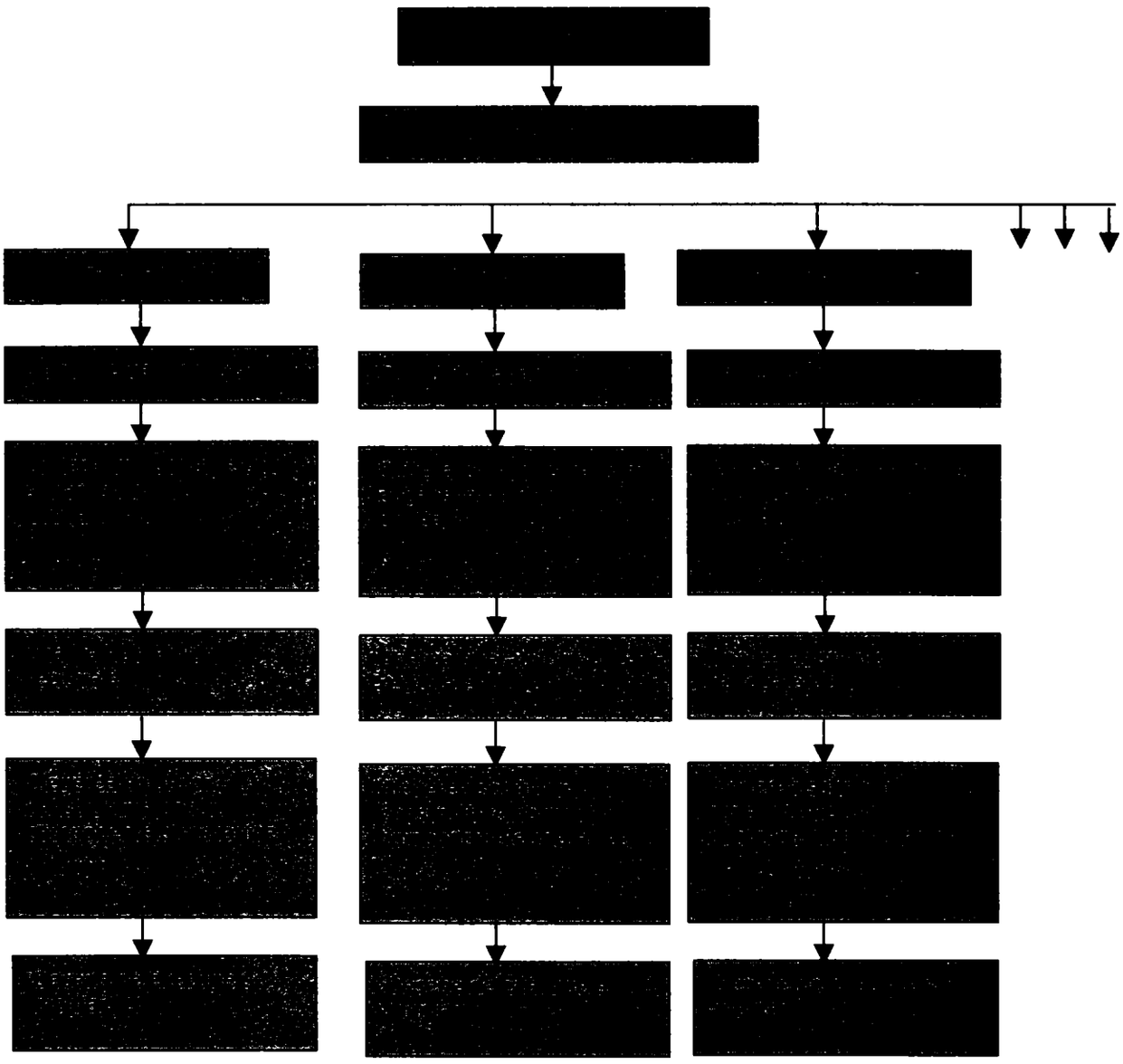


Fig.5.1

Pe baza cerinţelor impuse sistemului optic se alege schema optică necesară, apoi, pentru fiecare tip de aparat optic, de urmăreşte un traseu general care include calculul de gabarit, sinteza componentelor şi subansamblurilor, analiza aberaţiilor reziduale şi alegerea soluţiilor constructive pentru componentele mecanice ale tubului. Pentru fiecare tip de aparat aceste etape au un caracter particular, astfel încât programul general se derulează cu secvenţe proprii fiecărei ramuri sau se poate diviza într-o serie de subprograme expert destinate câte unei scheme optice anume. La rândul lor, aceste subprograme conţin module legate prin intercondiţionări reciproce şi apelări iterative.

Rezultatele furnizate de program necesită prezentarea sub forma unui şir de date numerice pentru parametrii geometrici şi optici ai componentelor şi reprezentări grafice pentru evaluarea aberaţiilor reziduale, astfel încât, la proiectarea automată în domeniul opticii, se pretează foarte bine limbajele de programare noi, orientate pe obiecte, care asigură o comunicare rapidă, intuitivă şi flexibilă cu utilizatorul.

Programul de proiectare automată alege componentele şi subansamblurile optice optime pe baza a două criterii:

- satisfacerea caracteristicilor funcţionale de ansamblu ale echipamentului (aparatului) optomecanic (deschidere, unghi de câmp obiect, gabarit, grosisment etc.)
- asigurarea calităţii imaginii prin teste specifice.

Se definesc, în continuare, câteva noţiuni esenţiale pentru componente şi subansambluri optice:

- **dioptru** – suprafaţa de separaţie a două medii optice omogene diferite; dioptrii se numerotează în ordine crescătoare de la stânga la dreapta, în lungul axei optice.

În figura 5.2 este prezentată clasificarea dioptrilor după criteriile funcţie optică şi formă geometrică.

- **sistem optic elementar (piesă optică sau element optic**) – un corp delimitat de doi dioptri succesivi, astfel concepuţi încât mediul imagine al primului să constituie mediul obiect pentru cel de-al doilea
- **sistem optic** – succesiunea mai multor dioptri centraţi pe aceeaşi axă optică
- **subansamblu optic** – mai multe piese optice centrate, cu scopul realizării unei funcţii optice definite (de ex.: obiectiv, ocular, condensor, redresor etc.).

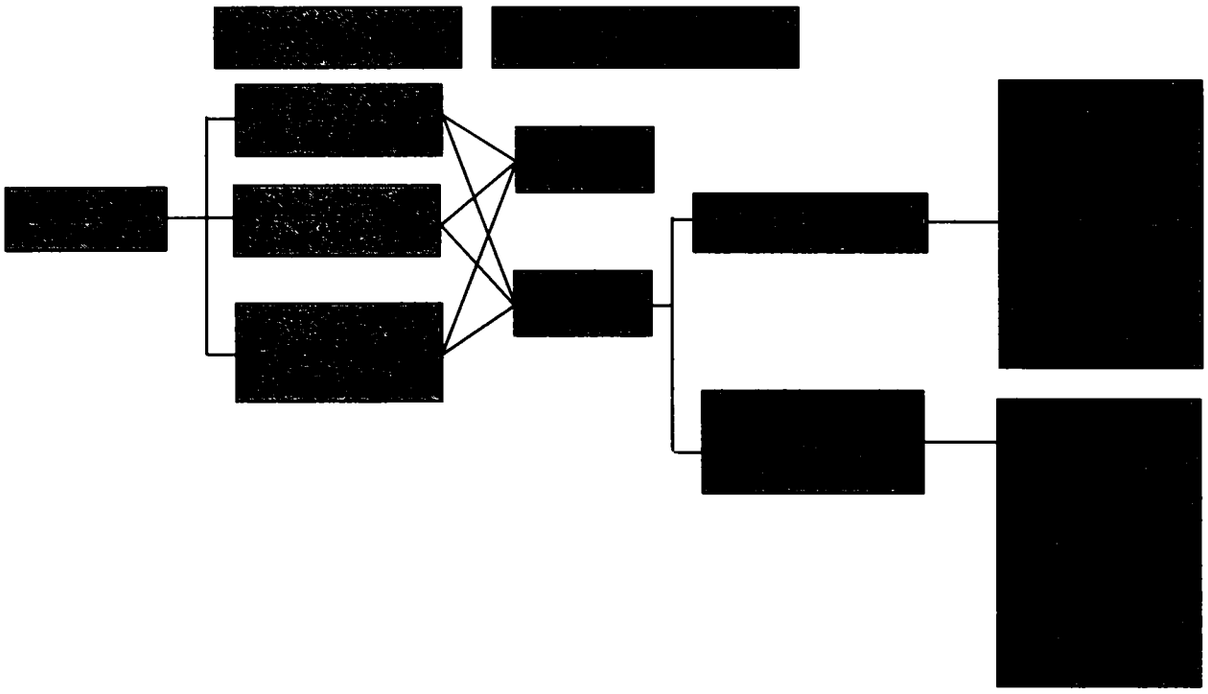


Fig.5.2

5.1.2 Aspecte privind structura optică modulară din construcţia unor aparate şi echipamente optomecanice şi optoelectronice reprezentative

În figura 5.3 sunt prezentate conexiunile cele mai probabile dintre diversele tipuri de aparate şi echipamente de mecanică fină şi optică şi componentele, subansamblurile şi materialele asociate acestora. Aceleaşi conexiuni rezultă din tabelul 5.1.

I	Lupe		1-3-6-8-10-12
II	Colimatoare Autocolimatoare		1-4-5-6-7-12-15-18-20
III	Aparate optice afocale	Lunete	21-23-24-27 1-4-6-7-12-14-18-23
IV		Binocluri	-24-25-26-27 1-4-6-7-12-14-18-21
V		Telescoape	23-24-25 1-5-6-7-12-18-20-21
VI	Aparate de mărit, de proiecţie, profil-proiectoare		23-24-27 1-4-5-6-7-8-12-23-27
VII	Aparate cine-foto		30 1-6-7-8-10-11-12-13-14
VII I	Microscoape		18-20-21-23-27 1-2-4-5-6-7-8-10-11-12
IX	Aparate optoelectronice de măsurare în teren		14-16-17-18-19-20-21 22-23-24-27-30 1-3-4-5-6-7-11-12-18
X	Aparate optice de măsurare şi investigare de laborator		20-21-22-23-24-27-28 29-30 1-2-3-4-5-6-7-11-15-16
XI	Aparate spectrale		17-18-20-21-23-24-27 30 1-2-4-5-6-7-15-16-17
XII	Fotometre Radiometre		18-19-21-24-26-27-30 1-2-4-5-6-7-11-15-16
XII I	Aparatura optoelectronică		17-18-19-21-24-27-30 1-6-7-9-12-21-23-24 28-29-30

1	Sticla optică incoloră	Materiale de uz general
2	Cristale optice	
3	Sticla organică	
4	Materiale pentru filtre	
5	Materiale pentru oglinzi	
6	Materiale pentru acoperiri	
7	Materiale metalice	
8	Materiale plastice	
9	Materiale utilizate în domeniul IR şi UV	
10	Lentile şi sisteme de lentile	Componente optice
11	Lame plan-paralele	
12	Reticule	
13	Lame de protecţie	
14	Lame compensatoare	
15	Lame divizoare	
16	Lame cristaline	
17	Reţele de difracţie	
18	Filtre optice	
19	Discuri şi rigle optice	
20	Oglinzi (plane şi curbe)	
21	Prisme de deviere, reflexie, dispersie, polarizare, divizare	
22	Componente optoelectronice	
23	Obiective	Subansambluri optice
24	Oculare	
25	Sisteme optice de redresare	
26	Compensatoare	
27	Sisteme optice de iluminare	
28	Detectori de radiaţie	
29	Dispozitive de conversie a radiaţiei IR	
30	Componente şi subansambluri pentru prelucrarea, stocarea şi afişarea informaţiei	

Fig.5.3

Tabelul 5.1

Modul	Simbolul numeric al aparatului												
I	I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII
1	*	*	*	*	*	*	*	*	*	*	*	*	*
2								*	*	*	*	*	
3	*									*			
4		*	*	*		*		*	*	*	*	*	
5		*			*	*		*	*	*	*	*	
6	*	*	*	*	*	*	*	*	*	*	*	*	*
7		*	*	*	*	*	*	*	*	*	*	*	
8	*					*	*	*		*			
9													*
10	*						*	*		*			
11							*	*	*		*	*	*
12	*	*	*	*	*	*	*	*	*	*			
13							*			*			
14				*			*	*					
15		*								*	*	*	
16								*		*	*	*	
17								*		*	*	*	
18		*	*	*	*		*	*	*	*	*	*	
19								*			*	*	
20		*			*		*	*	*	*			
21		*		*	*		*	*	*	*	*	*	
22								*	*				
23		*	*	*	*	*	*	*	*	*			*
24		*	*	*	*			*	*	*	*	*	*
25			*	*									
26			*								*	*	
27		*	*		*	*	*	*	*	*	*	*	
28									*				*
29									*				*
30						*		*	*	*	*	*	*

5.2 Consideratii generale asupra programului

5.2.1 Schema logică generală a programului

Programul de calcul are ca principale etape:

- stabilirea structurii modulare a echipamentelor optomecanice pe baza parametrilor optici şi geometrici generali
- adoptarea schemei optice cu module echivalente, avându-se în vedere distantele focale, puterile optice, caracteristicile de deschidere etc.
- calcule de gabarit
- calculul caracteristicilor de referinţă ale sistemelor optice
- analiza şi verificarea calităţii imaginii.

În figura 5.4 este redată schema logică de principiu a proiectării automate a sistemelor optomecanice.

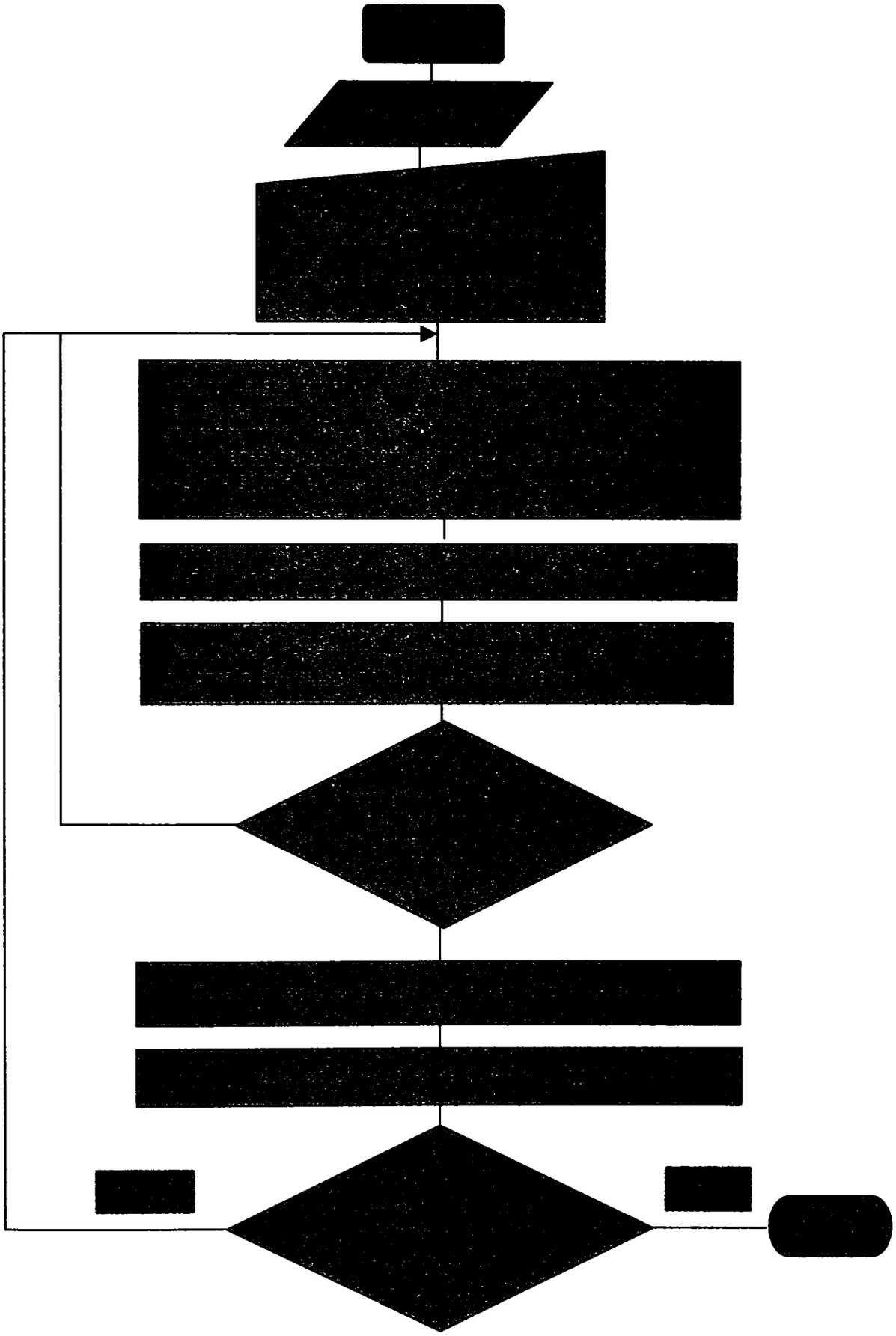


Fig.5.4

5.2.2 Particularităţi ale bazei de date

În cele ce urmează (fig.5.5) se prezintă structura bazei de date. În esenţă, aceasta conţine:

- materiale optice identificate după sort şi caracterizate prin indici de refracţie, dispersie principală şi numărul Abbe
- componente şi subansambluri optice şi optoelectronice, definite prin elemente specifice fiecărui tip.

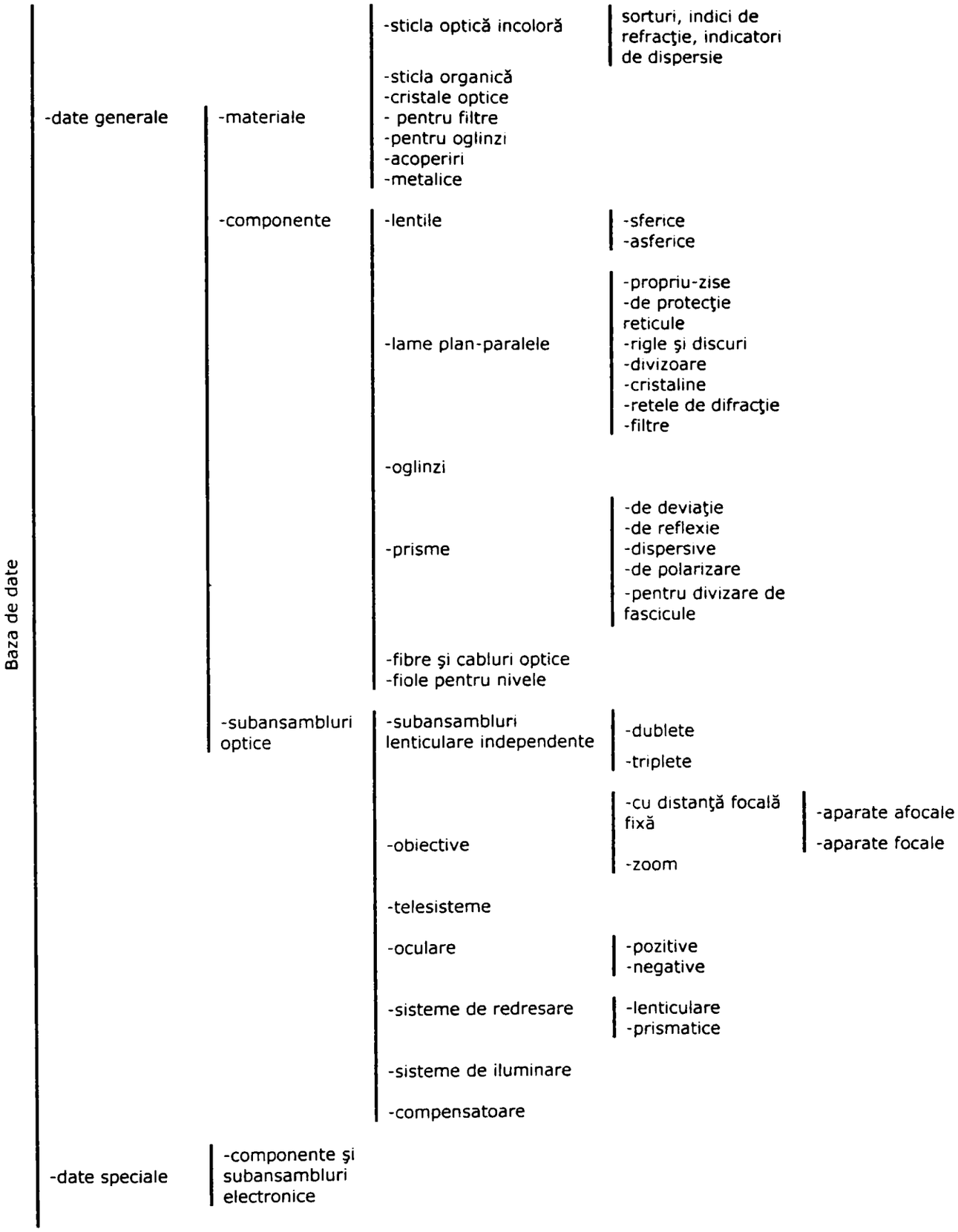

Baza de date
-date generale
-materiale
-sticla optică incoloră
sorturi, indici de refracţie, indicatori de dispersie
-sticla organică
-cristale optice
- pentru filtre
-pentru oglinzi
-acoperiri
-metalice
-componente
-lentile
-sferice
-asferice
-lame plan-paralele
-propriu-zise
-de protecţie
reticule
-rigle şi discuri
-divizoare
-cristaline
-retele de difracţie
-filtre
-oglinzi
-prisme
-de deviaţie
-de reflexie
-dispersive
-de polarizare
-pentru divizare de fascicule
-fibre şi cabluri optice
-fiole pentru nivele
-subansambluri optice
-subansambluri lenticulare independente
-dublete
-triplete
-obiective
-cu distanţă focală fixă
-aparate afocale
-aparate focale
-zoom
-telesisteme
-oculare
-pozitive
-negative
-sisteme de redresare
-lenticulare
-prismatice
-sisteme de iluminare
-compensatoare
-date speciale
-componente şi subansambluri electronice

Fig.5.5

5.3 Calitatea sistemelor optice

5.3.1 Generalitati

Calitatea sistemelor optice trebuie reflectată prin gradul în care acestea satisfac cerinţele impuse în tema de proiectare.

Sisteme optice au principala funcţie de a forma imaginea unui obiect caracterizat prin mărime, structură, poziţie şi distribuţie spectrală. Reprezentarea optică a obiectului se va caracteriza prin mărime, poziţie, structură şi distribuţie spectrală diferite de ale obiectului. Compararea caracteristicilor imaginii cu cele ale obiectului sau cele ale unei imagini dorite, respectiv impuse prin tema de proiectare, permite aprecierea calităţii imaginii date de sistemul optic.

Evaluarea calităţii imaginii se face prin verificarea unui set de condiţii, numite criterii de calitate a imaginii.
Criteriile de calitate pot fi locale sau globale, după cum se referă la imaginea unui punct sau a unei linii dintr-o anumită zonă a imaginii sau la întreaga imagine a unui obiect extins.

O altă clasificare are în vedere nivelul de aproximare al formării imaginii prin considerarea unor criterii optico-geometrice, respectiv a unor criterii optico-ondulatorii.

Pentru a putea aplica criteriile de calitate menţionate mai sus trebuie descrisă calitatea imaginii sistemului prin globalizarea calităţii imaginii punctului sau liniei.

În literatură, [D1, H1],acest lucru se recomandă a se face prin gruparea sistemelor în funcţie de natura obiectului de reprezentat, de natura receptorului şi de distribuţia de energie. Funcţie de stadiul de corecţie al sistemului se folosesc diferite criterii de calitate.

Berek, [B2] şi FIluge au propus primele sisteme de evaluare pe baza aberaţiilor de ordinul III, care permit atât compararea performanţelor diferitelor sisteme optice, cât şi semnalarea punctelor slabe ale sistemului. Aceste criterii se consideră că reprezintă o aproximare acceptabilă doar pentru sistemele

subcorectate. Totuşi, această aproximare poate fi suficientă pentru sistemele optice simple, cu deschidere şi câmp mici, respectiv cerinţe calitative reduse.

Criterii foarte eficiente se pot formula având în vedere aberaţiile geometrice şi variaţia cromatică a acestora, stabilite – predilect – prin drumuiri vectoriale sau trigonometrice.

Criteriile optico-geometrice, având la bază drumuirea vectorială, pot reprezenta o măsură pentru calitatea sistemului până la atingerea limitelor difracţionale ale acestuia.

În cazul unui sistem cu rezoluţie înaltă – cum este, de exemplu, categoria sistemelor optice telescopice – efectul fenomenului de difracţie nu este neglijabil şi, prin urmare, pentru descrierea calităţii imaginii se va apela la criteriile optico-ondulatorii.

Criteriul cel mai frecvent este cunoscut sub numele de Criteriul $\lambda/4$, denumit şi Criteriul Rayleigh. Conform acestui criteriu, dacă aberaţia de undă este mai mică decât $\lambda/4$, atunci imaginea punctului se consideră că nu este afectată de aberaţii.

Aberaţia de undă permite formularea altui criteriu care are în vedere numărul de inele de interferenţă din interferograma frontului de undă. Sthrel arată că dacă criteriul Rayleigh este satisfăcut, atunci luminozitatea de definiţie este mai mare de 0.8.

În cazul sistemelor care folosesc ca receptori CCD-uri s-a stabilit că cele mai practice criterii sunt cele care implică distribuţia energiei pe domenii circulare în imaginea punctului – "energie encicle".

Această mărime exprimă raportul dintre cunantumul energetic distribuit în cercuri concentrice cu diametre de diverse valori şi energia totală :

$$EE0(r) = \frac{\iint_{\infty} E(u,v)dudv}{\iint_{D} E(u,v)dudv} \quad Y \quad D = \left\{(u,v) \middle| u^2 + v^2 \le r^2\right\}. \qquad (5.1)$$

Energia enciclată se poate exprima, în aproximare geometrică, prin numărul de raze incluse în cercuri de diferite diametre.

Un alt criteriu are la bază diametrul zonei în care se găseşte 90% din energie. Brecht consideră că sistemul este bun dacă 80% din energie se află într-un cerc cu diametrul de 15 μm.

Ambele criterii s-au aplicat la sisteme telescopice – sisteme asupra cărora se vor face aplicaţii în teză. Limitarea diamterului petei de difizie este utilizată drept criteriu şi pentru sinteza obiectivelor. În cazul obiectivelor Nicon diametrul petei de difuzie este de 0.03 mm, iar la obiectivele Zeiss, de 0.025 mm.

Criterii foarte practice şi moderne se formulează cu ajutorul funcţiei optice de transfer [D1, R1, R2, R3, S1]. Potrivit acestor criterii, un sistem telescopic modern dă satisfacţie dacă valoarea contrstului este mai mare decât 30% la frecvente spaţiale de până la 50 pl/mm, pentru orice lungime de undă, câmp sau configuraţie.

În [AI 18] se arată că un obiectiv foarte bun trebuie să aibă pe axă un contrast de cel puţin 50% la o frecvenţă spaţială de 40 pl/mm. Un contrast bun la 40 pl/mm devine necesar în cazul măririi imaginii.

Unele criterii de calitate au în vedere distorsiunea. Astfel, se consideră că pentru obiectivele fotografice distorsiunea nu trebuie să depăşească 4%.

Asupra calităţii imaginii o influenţă relativ mare o are vignetarea fasciculelor înclinate, datorită diafragmelor şi monturilor. Aceasta determină scăderea intensităţii luminoase în planul imaginii de la centru spre periferie. O vignetare de 50% nu este perceptibilă pe un ecran TV. În general, o iluminare relativă de 40% este acceptabilă.

Prin micşorarea valorilor aberaţiilor de ordinul III şi a diametrului petei de difuzie se obţine o micşorare a aberaţiei de undă cu consecinţe immediate în creşterea rezoluţiei şi a luminozităţii de definiţie, respectiv în îmbunătăţirea distribuţiei energiei în zone circulare prin concentrarea acesteia şi atingerea limitei difracţionale.

O influenţă deosebită asupra calităţii imaginii o are poziţia planului imagine întrucât pentru fiecare punct obiect, lungime de undă, câmp şi configuraţie există o poziţie optimă a planului imagine, un optim local, corespunzător minimizării globale a funcţiei de merit [M1].

5.3.2 Criterii de calitate locale

5.3.2.1 Criterii deduse din diafragma spoturilor

Un criteriu local are în vedere verificarea dimensiunilor petei de difuzie:

$$D_{80\%} \le D_{LIM}, \forall \lambda \in D\lambda,$$
sau . (5.2)
$$D_{RMS} \le D_{LIM}, \forall \lambda \in D\lambda$$

unde:

$D_{LIM} = \frac{2l}{3400}$, pentru ca două puncte cu distanţa minimă între ele să fie percepute de ochi ca puncte distincte de la distanţa l,

$D_{LIM} = \frac{2l}{\beta 3400}$, pentru ca două puncte cu distanţa minimă între ele să fie observate pe o fotografie, obţinută prin mărirea de β ori a imaginii, aflate la distanţa l de ochi.

$D_{LIM}=D_{pcx}$ pentru CCD-uri.

Dacă $D_{LIM}<D_{AIRY}$, calitatea imaginii punctului nu este influenţată de aberaţiile sistemului optic, ci doar de fenomenul de difracţie.

5.3.2.2 Criterii deduse din distributia energiei

Un criteriu foarte practic şi important se obţine folosind energia enciclată. potrivit acestuia, o anumită parte din energie trebuie să se afle distribuită într-un cerc cu un anumit diametru. de regulă, se acceptă ca 80% din energie să se afle în interiorul unui cerc cu diametrul D_{LIM}:

$$D_{E80} \le D_{LIM}, \forall \lambda \in D\lambda . \quad (5.3)$$

Acest criteriu se recomandă la sistemele limitate difracţional cum sunt telescoapele şi obiectivele de mare rezoluţie.

4.3.2.3 Criterii deduse din functia optica de transfer

Se consideră un obiect sub forma unei reţele cu factor de transmisie sinusoidal şi cu o anumită frecvenţă spaţială (fig.5.5).

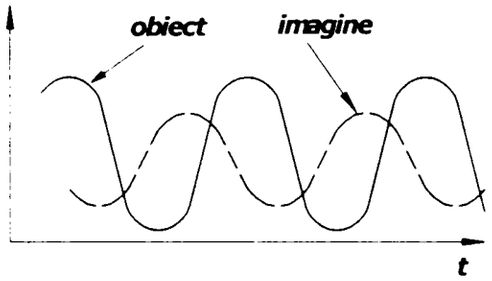


Fig.5.5

Dacă acest obiect este iluminat necoerent, imaginea formată de un sistem optic cu aberaţii, este tot de formă sinusoidală, dar prezintă contrast redus şi salt de fază.

Variaţia constrastului şi a fazei, în funcţie de frecvenţa spaţială sunt date de funcţia de transfer optic a modulaţiei (FTOM), respectiv de funcţia de transfer optic a fazei (FTOF).

În acest context, se defineşte contrastul sau modulaţia:

$$K = \frac{L_{max} - L_{min}}{L_{max} + L_{min}}, \qquad (5.3)$$

unde L este o mărime corespunzătoare tipului de obiect (intensitate, factor de transmisie etc.).

Funcţia de transfer optic a modulaţiei indică factorul de contrast K în funcţie de frecvenţa spaţială s:

$$MTF(s) = \frac{K'(s)}{K(s)}, \qquad (5.4)$$

unde K(s) este contrastul în planul obiect, iar K'(s) este contrastul în planul imagine.

Modificarea fazei în funcţie de frecvenţa spaţială este dată de funcţia de transfer optic a fazei: FTOF(s).

Funcţia de transfer optic (FTO) complexă a sistemului este dată de relaţia:

$$FTO = FTOM(s) \cdot e^{iFTOF(s)}. \qquad (5.5)$$

Funcţia de transfer optic complexă, bidimensională, se defineşte ca fiind transformata Fourier inversă a funcţiei intensităţii în planul imagine:

$$FTO(s,t) = \frac{\int_{-\infty}^{\infty}\int E(u,v)e^{2\pi i(su+tv)}dudv}{\int_{-\infty}^{\infty}\int E(u,v)dudv}, \qquad (5.6)$$

unde s şi t sunt frecvenţele spaţiale în direcţia u, respectiv v, unitatea lor de măsură fiind număr de perechi de linii cuprinse într-un milimetru (pl/mm).

Pentru o pupilă circulară de rază a, frecventele spaţiale maxime sunt:

$$s_{max}t_{max} = \frac{2a}{\lambda R}. \qquad (5.7)$$

Pentru un sistem lipsit de aberaţii, funcţia optică de transfer este descrisă de ecuaţia:

$$FTO(s) = \frac{2}{\pi}\arccos\frac{\lambda Rs}{2a} - \frac{\lambda Rs}{\pi a}\sqrt{1-\frac{\lambda^2R^2s^2}{4a^2}}. \qquad (5.8)$$

Prin urmare,

$$FTOM(s,t) = \sqrt{FTO_{real}^2 + FTO_{imag}^2}, \qquad (5.9)$$

iar

$$FTOF(s,t) = \text{arctg}\frac{FTO_{imag}}{FTO_{real}}. \qquad (5.10)$$

FTO reprezintă un indicativ direct al calităţii sistemului optic.

Cu cât FTOM este mai mare, pentru o frecvenţă spaţială, cu atât contrastul obiectului se regăseşte in contrastul imaginii.

Un criteriu având la bază FOTM îl reprezintă verificarea modulaţiei (contrastului) pe un domeniu de frecvenţe spaţiale:

$$FTOM(s) \geq K_{MIN}, \forall s \in Ds, \forall \lambda \in D\lambda, \qquad (5.11)$$

unde K_{MIN} este contrastul minim acceptat, iar Ds este domeniul frecvenţelor spaţiale de interes.

5.3.3 Criterii de calitate globale [D1]

Dacă sunt satisfăcute simultan condiţiile:

1.1. $D_{RMS(80\%)} \leq D_{LIM}, \forall \lambda \in D\lambda$ (5.12)

sau 1.2. $D_{E80} \leq D_{LIM}, \forall \lambda \in D\lambda$ (5.13)

sau 1.3. $FTOM(s) \geq K_{MIN}, \forall s \in Ds,$ (5.14)

2. $DIS \leq DIS_{LIM}$, (5.15)

3. $IR \geq IR_{LIM}$, (5.16)

atunci calitatea imaginii sistemului optic corespunde celei impuse în tema de proiectare.

Prin satisfacerea condiţiilor 1.1, 1.2 sau 1.3 se obţin imagini clare. Condiţia 2 asigură o imagine asemenea cu obiectul, iar condiţia 3 asigură o iluminare suficientă a extremităţii imaginii.

În funcţie de tipul proiectului şi etapa de proiectare se folosesc diferite criterii. Astfel, în etapa de optimizare grosieră (după sinteza seideliană) se pot utiliza primele două criterii, indiferent de destinaţia sistemului. Pentru optimizarea fină va fi accesat criteriul energetic pentru sisteme care au ca receptor CCD-uri şi FTO pentru sisteme care realizează imaginea pe film sau hârtie.

5.3.4 Valori optime ale indicatorilor de calitate

Cei mai utilizaţi receptori folosiţi în tehnică sunt ochiul şi CCD-ul. Indiferent de tipul receptorului, limita superioară a calităţii imaginii este dată de caracteristicile acestuia, astfel că receptorul, peste această limită, nu este sensibil la creşterea calităţii imaginii sistemului optic.

Receptoarele de tipul CCD-ului au cunoscut o dezvoltare continuă, numărul de pixeli crescând de la 100x100 (1974) la 2048x2048. Astfel, dimensiunea unui pixel D_{pcx} poate ajunge la 3μm la un CCD de ¼", adică:

$D_{pcx} \geq 3\mu m$.

În cazul CCD-ului, două puncte mai pot fi percepute ca puncte distincte dacă imaginea unui punct se formează pe un pixel, iar imaginea celui de-al doilea pe un pixel adiacent. Prin urmare, dimensiunea minimă a imaginii punctului este limitată de dimensiunea unui pixel.

Ochiul uman normal, ca receptor optic, are puterea de rezoluţie 60" (aproximativ a 3400-a parte din distanţa până la cele două puncte). Contrastul minim perceput de ochi este de $K_{MIN}=0.02$ pentru o pupilă a ochiului de 0.6 mm. [M21] Prin urmare, ochiul uman nu percepe ca distincte peste 6 perechi de linii pe mm (6 pl/mm) dacă acestea sunt observate de la o distanţă de 250 mm.

Valorile optime ale indicatorilor de calitate reprezintă limita superioară a acestor valori după care orice modificare a acestora în sensul creşterii calităţii nu mai este percepută de senzor.

Pentru ca o imagine fixată pe hârtie fotografică format A4 să fie percepută clar trebuie să se rezolve 6 pl/mm. Această imagine se obţine prin mărirea de 8 ori a formatului filmului. Prin urmare, acesta trebuie să aibă o rezoluţie de cel puţin 48 pl/mm, adică spaţiul ocupat de un punct pe film trebuie să fie de cca. 0.02 mm. Astfel, se poate determina valoarea optimă a indicatorilor de calitate deduşi din imaginea punctului.

Dacă diametrul petei de difuzie este de 0.025 mm, atunci două linii distincte se află la o distanţă de 0.0125 mm. Frecvenţa spaţială pe film este de 80 linii/mm, respectiv 40 pl/mm. Cu un astfel de film se poate realiza o fotografie de dimensiuni maxime A4.

Pentru un CCD, limita maximă a frecventei spaţiale care poate fi reprodusă este $R_{max}=1/2D_{pcx}$ [pl/mm].
Dacă frecvenţa spaţială creşte peste R_{MAX}, imaginea are o frecvenţă spaţială mai mică decât cea a obiectului.

Practic, informaţia transmisă de CCD după limita R_{MAX} este falsă si nu prezintă interes.

O apreciere de ansamblu a calităţii imaginii se poate face şi prin reprezentarea într-o diagramă a razei geometrice a petei de difuzie în funcţie de configuraţie, pentru câmpurile extreme.

5.4 Etapele calculului de proiectare a aparatelor optice

5.4.1 Calculul de gabarit

Având stabilită schema optică şi cunoscând valorile datelor de intrare se poate desfăşura calculul de gabarit al sistemului. Pe baza legilor de formare a imaginii

şi a caracteristicilor acestora, în formularea dată de optica geometrică pentru domeniul paraxial, se determină:

- puterea optică (sau distanţa focală imagine) a subansamblurilor componente (obiectiv, ocular, redresor, condensor etc.),
- diametrul util al componentelor,
- poziţia componentelor pe axa optică,
- mărimea şi poziţia difragmelor de deschidere şi de câmp (poziţia şi mărimea pupilelor şi lucarnelor).

În această etapă schema optică include componente reduse, la care grosimea se neglijează.

5.4.2 Sinteza componentelor şi subansamblurilor

Structura subansamblurilor optice este determinată de putere, diametru util şi aberaţii geometrice sau cromatice care trebuie controlate. În funcţie de destinaţie şi valorile concrete ale caracteristicilor optice, subansamblurile pot fi formate din o singură lentilă, un dublet (lipit sau nelipit), un triplet (lipit sau nelipit) sau o combinaţie a acestora.

Pentru cazul când proiectantul are la dispoziţie un catalog de subansambluri optice, acestea se aleg luând în considerare următoarele criterii:

- distanţa focală imagine,
- diametrul util,
- unghiul de câmp obiect/imagine,
- mărimea aberaţiilor.

Subansamblurile alese au caracteristici optice ale căror valori pot să nu coincidă în totalitate cu cele determinate prin calculul de gabarit. De aceea, acesta se reia pentru a face mici corecţii. De asemenea, având cunoscută geometria subansamblurilor, se recalculează distanţele între componente cu luarea în considerare a grosimilor la centru.

Pentru cazul în care cataloagele nu oferă subansambluri cu caracteristici apropiate celor rezultate din calculul de gabarit, este necesară proiectarea efectivă a componentelor optice. Sinteza lentilelor, dubletelor sau tripletelor reprezintă partea cea mai dificilă şi laborioasă a calculului de proiectare optic.

În principiu, numărul lentilelor dintr-un subansamblu optic este determinat de condiţiile care se impun asupra caracteristcilor sale. În majoritatea cazurilor, sorturile de sticlă se aleg, astfel încât indicii de refracţie intră în calcule ca mărimi cunoscute. Fiecărui dioptru i se asociază un grad de libertare, reprezentat de raza sa.

Prima condiţie care se impune oricărei componente este puterea optică. Celelalte variabile din sistem sunt utilizate pentru a asigura un nivel cât mai scăzut al aberaţiilor.
Din punct de vedere matematic, scrierea condiţiilor referitoare la putere şi corectarea aberaţiilor nu conduce la un sistem liniar de ecuaţii, ci la o serie de subsisteme liniare sau neliniare, care se reiau iterativ de un număr de ori care depinde de convergenţa soluţiilor.

Numărul lentilelor subansamblului poate creşte şi datorită condiţiilor de gabarit transversal.

Din calculul de sinteză rezultă toate caracteristicile geometrice (raze, grosime la centru, grosime la margine, diamteru util, diametru total) şi optice (distanţe focale şi frontifocale, abscisele planelor principale) ale lentilelor.

5.4.3 Analiza aberaţiilor reziduale

În urma sintezei componentelor trebuie verificat dacă nivelul aberaţiilor pentru întreg sistemul optic se află sub limita admisă. În principiu, toleranţele acceptate pentru aberaţii se înscriu în jurul valorilor rezoluţiei receptorului.

Dificultatea în controlul aberaţiilor constă în faptul că acestea se influenţează reciproc şi pot avea sensuri de variaţie opuse. Diminuarea unei aberaţii poate conduce la creşterea nepermisă a unei alte aberaţii, astfel încât este necesară o optimizare a setului de valori care le definesc. Tehnica de calcul actuală permite execuţia unui volum mare de calcule, astfel încât obţinerea unor soluţii performante din punct de vedere al calităţii imaginii este la îndemâna proiectantului.

Pentru sistemele simple, analiza aberaţiilor se desfăşoară concomitent cu sinteza componentei. Pentru cele formate din mai multe subansambluri, acestea se verifică separat, iar rezultatele se compun pentru a putea evalua comportamentul global al sistemului.

Calculul aberaţiilor presupune drumuiri paraxiale şi trigonometrice. În tabelul 4.2 sunt prezentate sintetic relaţiile utilizate pentru drumuirea paraxială şi trigonometrică directă, scrise pentru un dioptru (fig.5.6).

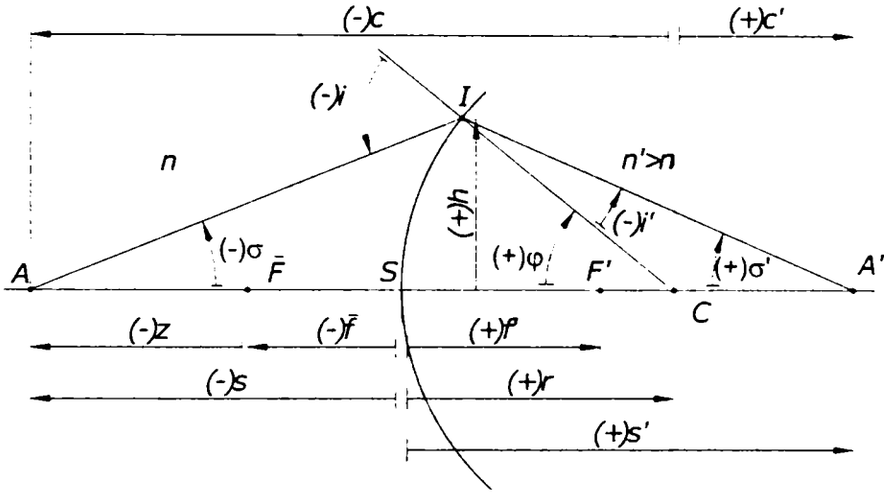


Fig.5.6

Tabelul 5.2

Drumuirea paraxială	Drumuirea trigonometrică
$s' = \dfrac{n'}{\dfrac{n}{s} + \dfrac{n'-n}{r}}$ (5.17)	$\sin\tilde{i} = -\dfrac{(\tilde{s}-r)\sin\tilde{\sigma}}{r}$ $\sin\tilde{i}' = \dfrac{n}{n'}\sin\tilde{i}$ $\tilde{\sigma}' = \tilde{\sigma} - \tilde{i} + \tilde{i}'$ $\tilde{s}' = r\left(1 - \dfrac{\sin\tilde{i}'}{\sin\tilde{\sigma}'}\right)$ $\sin\tilde{i}' = -\dfrac{(\tilde{s}'-r)\sin\tilde{\sigma}'}{r}$ $\sin\tilde{i} = \dfrac{n'}{n}\sin\tilde{i}'$ $\tilde{\sigma} = \tilde{\sigma}' - \tilde{i}' + \tilde{i}$ $\tilde{s} = r\left(1 - \dfrac{\sin\tilde{i}}{\sin\tilde{\sigma}}\right)$

unde s' este distanţa imagine,

s – distanţa obiect,
n' – indice de refracţie al mediului imagine,
n – indice de refracţie al mediului obiect,

r – raza dioptrului,
$\tilde{i}$ - unghiul de incidenţă,
$\tilde{i}'$ - unghiul de emergenţă,
$\tilde{\sigma}$ - unghiul dintre raza incidentă şi axa optică,
$\tilde{\sigma}'$ - unghiul dintre raza emergentă şi axa optica

Relaţiile calcul al aberaţiilor de ordinul III sunt redate în tabelul 5.3.

Tabelul 5.3

Aberaţia	Relaţia de calcul
Aberaţia de sfericitate	$S_I = \sum_1^k h_j^4 Q_j^2 \Delta\left(\frac{1}{ns}\right)_j$ (5.26)
Aberaţia de coma	$S_{II} = \sum_1^k h_j^3 \bar{h}_j Q_j \bar{Q}_j \Delta\left(\frac{1}{ns}\right)_j$ (5.27)
Astigmatismul	$S_{III} = \sum_1^k h_j^2 \bar{h}_j^2 Q_j^2 \bar{Q}_j^2 \Delta\left(\frac{1}{ns}\right)_j$ (5.28)
Curbura de câmp	$S_{IV} = -\sum_1^k H^2 \frac{1}{r_j} \Delta\left(\frac{1}{n}\right)_j$ (5.29)
Distorsiunea	$S_V = \sum_1^k h_j \bar{h}_j^3 \left[\bar{Q}_j^2 \Delta\left(\frac{1}{ns}\right)_j - \bar{Q}_j (Q_j - \bar{Q}_j) \Delta\left(\frac{1}{ns_p}\right)\right]$ (5.30)
Cromatismul de poziţie	$C_I = \sum_1^k h_j^2 Q_j \Delta\left(\frac{\Delta n_\lambda}{n}\right)_j$ (5.31)
Cromatismul de mărire	$C_{II} = \sum_1^k h_j \bar{h}_j \bar{Q}_j \Delta\left(\frac{\Delta n_\lambda}{n}\right)_j$ (5.32)

S-au utilizat notaţiile:

- înălţimea de incidenţă a razei obiective pe dioptri:

$$h_1 = \frac{D_{pj}}{2\left(1 - \frac{s_{p1}}{s_1}\right)}; \quad h_{j+1} = h_j \frac{s_{j+1}}{s'_j}, \qquad (5.33)$$

- înălţimea de incidenţă a razei pupilare principale pe dioptri:

$$\bar{h}_1 = \begin{cases} y_o \frac{s_{p1}}{s_{p1} - s_1} & \text{daca } s_1 \neq \infty \\ -s_{p1} tg\sigma_{RPP} & \text{daca } s_1 = \infty \end{cases}; \quad \bar{h}_{j+1} = \bar{h}_j \frac{s_{pj+1}}{s_{p1}}, \qquad (5.34)$$

- invariantul paraxial obiectiv:

$$Q_j = n_j \left(\frac{1}{R_j} - \frac{1}{s_j}\right), \qquad (5.35)$$

- invariantul paraxial pupilar:

$$Q_j = n_j \left(\frac{1}{R_j} - \frac{1}{s_{pj}}\right), \qquad (5.36)$$

- invariantul Lagrange:

$$H = h_1 \bar{h}_1 \left(\frac{1}{s_{p1}} - \frac{1}{s_1} \right), \qquad (5.37)$$

unde s_{p1} este abscisa pupilei de intrare,

s_1 – absacisa obiectului,

D_{pi} – diametrul pupilei de intrare.

La acestea se poate adăuga aberaţia de sfericitate pupilară:

$$SP_1 = \sum \bar{h}_j^4 \bar{Q}_j^2 \Delta \left(\frac{1}{ns_p} \right)_j . \qquad (5.38)$$

Această aberaţie deşi nu afectează direct calitatea imaginii, interacţionează cu aberaţiile de ordinul III, inducând aberaţii de ordin superior.

Se pot totuşi menţiona două efecte ale acestei aberaţii [D1]:

- efect static – interacţiunea cu curbura de câmp induce distorsiunea de ordinul V, care poate fi utilizată pentru compensarea distorsiunii de ordinul III
- efect dinamic – important în cazul modificării distanţei focale a sistemului optic.

5.4.4 Alegerea soluţiilor constructive pentru componentele mecanice ale tubului optic

Având stabilită geometria componentelor optice şi distanţele dintre ele, se poate trece la proiectarea părţii mecanice. Aceasta are un caracter predominant constructiv. Dimensionarea monturilor şi a tuburilor de legătură nu are la bază calcule de rezistenţă, ci numai condiţiile de gabarit radial şi axial impuse de piesele optice.

Fiecare tip de subansamblu optic se fixează într-o montură specifică, având prevăzute elementele mecanice necesare reglării poziţiei, conform condiţiilor impuse de rolul funcţional.

5.5 Calculul de proiectare a lunetei Kepler [D2, G2, N1, N2]

5.5.1 Generalităţi

Luneta este un aparat optic, care are rolul de a mări unghiul sub care se vede un obiect îndepărtat, astfel încât să se distingă mai multe detalii ale acestuia. Este un sistem optic afocal sau telescopic, având distanţa focală infinită şi formează la infinit imagini ale obiectelor situate la infinit.

Luneta este compusă din două subansambluri optice de bază: obiectivul şi ocularul. Acestea sunt montate astfel încât focarul imagine al obiectivului să coincidă cu focarul obiect al ocularului, de unde rezultă caracterul afocal al sistemului.

În funcţie de caracteristicile optice ale componentelor, există două tipuri de lunete: luneta astronomică (Kepler), având convergente atât obiectivul, cât şi ocularul şi luneta terestră (Galilei), având obiectivul convergent şi ocularul divergent.

Schema optică a lunetei Kepler este prezentată în figura 5.7. Acest tip de lunetă formează la infinit imagini virtuale, răsturnate, ale obiectelor situate la infinit.

În schemă este figurat traseul razelor marginale dintr-un fascicul paralel cu axa optică, traseul razei pupilare principale (2) şi al razelor marginale (1) şi (3), dintr-un fascicul incident înclinat cu unghiul ω faţă de axa optică.

Diafragma de deschidere D_d a aparatului este montura obiectivului, care devine şi pupila de intrare P_i. Imaginea reală a acesteia este dată de ocular şi reprezintă pupila de ieşire P_e, care în majoritatea aplicaţiilor se materializează printr-o diafragmă de mărime fixă D'.

Cele trei raze trasate în fasciculul înclinat, se intersectează în planul focal comun, al imaginii intermediare y'_{ob}. Mărimea câmpului obiect $2y'_{ob}$ este determinată de deschiderea diafragmei de câmp D_C.

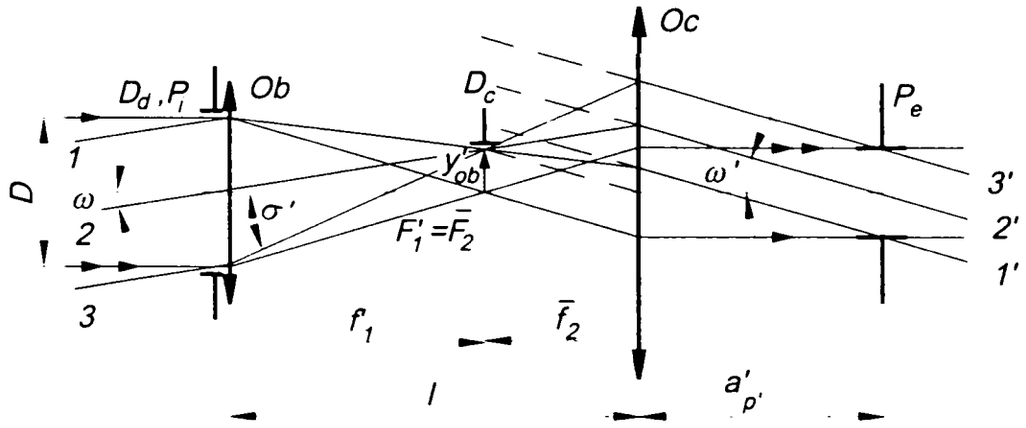


Fig.5.7

Prin definiţie, grosismentul lunetei este dat de raportul:

$$\Gamma = \frac{tg\omega'}{tg\omega}, \qquad (5.39)$$

unde ω' este semiunghiul de câmp imagine, iar ω - semiunghiul de câmp obiect.

Urmărind figura 2, se poate determina expresiile acetor unghiuri:

$$tg\omega = -\frac{y'_{ob}}{f'_1}, \qquad (5.40)$$

$$tg\omega' = -\frac{y'_{ob}}{\bar{f}_2}. \qquad (5.41)$$

Grosismentul devine:

$$\Gamma = \frac{f'_1}{\bar{f}_2} = -\frac{f'_1}{f'_2}. \qquad (5.42)$$

Pentru luneta Kepler, $f'_1>0$, $f'_2>0$ şi rezultă $\Gamma<0$ (imagine răsturnată).

Pentru luneta Galilei, $f'_1>0$,$f'_2<0$ şi rezultă $\Gamma>0$ (imagine dreaptă).

Grosismentul se poate exprima şi sub formă de raport al deschiderii pupilelor:

$$\Gamma = \frac{P_i}{P_e}. \qquad (5.43)$$

Luneta Kepler formează imagini răsturnate şi virtuale. Pentru obţinerea imaginilor drepte, instrumentul poate fi prevăzut cu subansambluri numite redresoare. Redresarea se poate realiza cu sisteme prismatice sau cu sisteme lenticulare.

Redresorul lenticular este constituit din una sau două lentile (de tip simplet sau dublet), care se introduc în schema de bază a lunetei (fig.5.8).

Dezavantajul redresorului lenticular este acela că măreşte gabaritul axial al lunetei.

Soluţia de redresare cu lentile este utilizată la luneta de vânătoare, de ochire, în construcţia teodolitelor, nivelmetrelor etc. Toate aceste lunete au montat în planul imaginii intermediare un reticul cruce sau/şi o scală gradată.

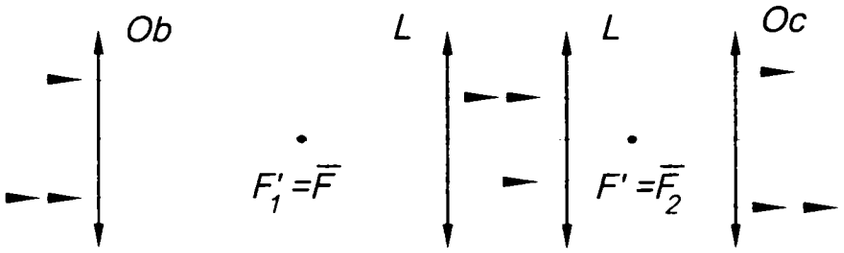


Fig.5.8

Redresarea prismatică (fig.4.9) cea mai cunoscută utilizează două prisme cu feţele corespunzătoare ipotenuzei lipite şi rotite cu 90° (Porro de speţa I).

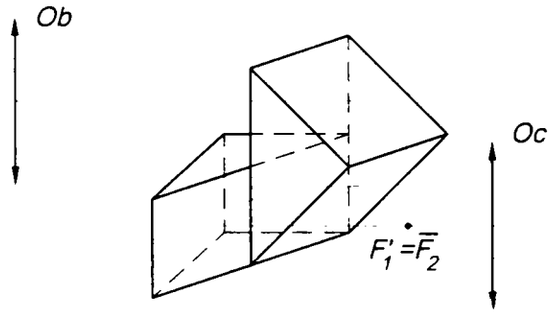


Fig.5.9

Ansamblul de prisme se intercalează între obiectiv şi ocular în apropierea planului focal comun, unde diametrul fasciculului luminos este mic. Redresarea prismatică scurtează mult lungimea tubului mecanic, atât datorită traseului frânt, cât şi mediului optic dens introdus de prisme. Această soluţie este specifică binoclului de teren.

Obiectivele pentru lunete se încadrează în categoria obiectivelor destinate aparatelor afocale, care au rolul de a forma imagini reale ale obiectelor situate la infinit (fig.5.10).

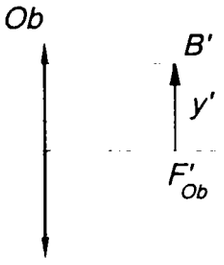


Fig.5.10

Având în vedere faptul că abscisa obiect este infinită, se poate admite că fasciculele incidente pe obiectiv au întotdeauna înclinaţie mică, astfel încât aberaţiile geometrice specifice fasciculelor înclinate sunt neglijabile. Seminificativă este aberaţia sferică pentru că obiectivele lunetelor au deschideri mari. În majoritatea aplicaţiilor alegerea sau proiectarea obiectivului are în vedere numai corectarea aberaţiei sferice şi cromatice. În mod frecvent, subansamblul obiectiv este un dublet acromat sau un triplet apocromat cu deschiderea relativă D/f'=1/10...1/15.

Ocularul utilizat în construcţia lunetelor reprezintă o lupă simplă (o singură lentilă, un dublet) sau compusă (un şir de lentile).

Ocularul compus este constituit din două componente: lentila de câmp şi lentila de ochi. Lentila de câmp se amplasează în apropierea planului imaginii intermediare şi are rolul de a strânge razele luminoase, care sunt trimise spre lentila ochiului sub unghiuri mai mici, astfel încât să se diminueze gabaritul transversal al ocularului. Lentila de ochi se situează în apropierea pupilei de ieşire. Funcţie de puterea lentilelor şi de distanţa dintre ele, se obţin diferite poziţii ale planului focal obiect.

Ocularele pozitive se bazează pe schema ocularului Ramsden (fig.5.11), la care cele două lentile au distanţele focale egale între ele şi egale cu distanţa dintre lentile.

Considerând $f'_1 = f'_2 = f'$ şi $e=f'$, puterea ocularului este:

$$\frac{1}{f'_{oc}} = \frac{1}{f'_1} + \frac{1}{f'_2} - \frac{e}{f'_1 f'_2} = \frac{1}{f'} + \frac{1}{f'} - \frac{f'}{f'f'} = \frac{1}{f'}. \qquad (5.44)$$

Dacă cele două lentile se confecţionează din aceeaşi sticlă se compensează aberaţiile cromatice. Pentru micşorarea aberaţiilor geometrice, cele două lentile plan-convexe se aşază cu suprafeţele curbe una spre cealaltă.

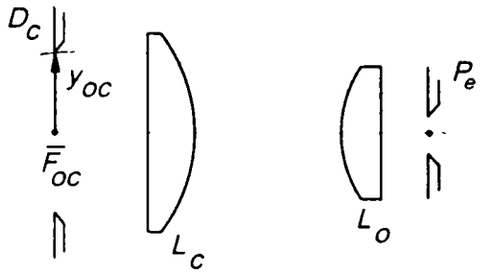


Fig.5.11

Două variante des utilizate ale ocularului Ramsden sunt ocularul Kellner, la care lentila ochiului este un dublet şi ocularul simetric, la care ambele lentile dublete.

Ocularele negative se bazează pe schema ocularului Huygens (fig.5.12).

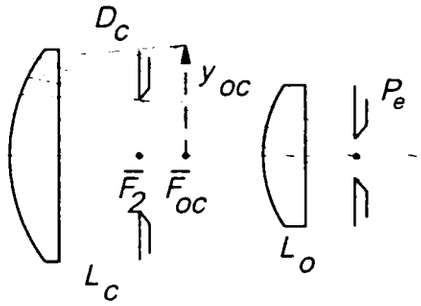


Fig.5.12

Lentila de câmp este plasată în faţa planului imaginii intermediare. Puterile lentilelor şi distanţa între ele se alege astfel încât:

$$f_1' = 2f_2'; \quad e = \frac{f_1' + f_2'}{2}; \quad f' = \frac{4}{3}f_2'. \qquad (5.45)$$

În analiza sau sinteza ocularelor se au în vedere câteva caracteristici principale:

- distanţa focală f'_{oc}; ocularele de uz general au distanţe focale cuprinse în intervalul [20;50]mm;
- grosismentul $\Gamma = 250/f'_{oc}$;
- câmpul unghiular imagine $2\omega'$; ocularele de uz general au câmpul imagine în domeniul [30;70]°;
- mărimea obiectului $2y_{oc}$ (numărul de ocular);
- diametrele utile ale lentilelor;
- depărtarea şi diametrul pupilei de ieşire, $s'_{p'}$, D';
- calitatea imaginii.

Luneta Kepler are largi aplicaţii şi se execută în variante diverse: lunetă astronomică, lunetă de vânătoare, lunetă înălţător, lunetă panoramică, binoclu sau intră în construcţia unor aparate cum ar fi goniometrul, telemetrele,

spectroscoapele etc. Lunetele de măsurare sunt prevăzute, în planul focal comun cu reticule reprezentând mire sau scale gradate.

5.5.2 Calculul de gabarit al lunetei Kepler

Pentru calculul de gabarit este necesară cunoaşterea următoarelor date de intrare:

- grosismentul, Γ
- unghiul de câmp obiect, 2ω
- mărimea pupilei de ieşire, D'
- depărtarea pupilei de ieşire, $s'_{P'}$
- lungimea tubului, l.

Din calcul rezultă următoarele elemente:

- distanţa focală a obiectivului, f'_{ob}
- distanţa focală a ocularului, f'_{oc}
- diametrul pupilei de intrare, D
- diametrul diafragmei de câmp, D_C
- diametrul util al ocularului, $D_{u\,oc}$
- depărtarea efectivă a pupilei de ieşire, $s'_{P'}$
- lungimea efectivă a tubului, l.

Sunt prezentate în continuare etapele de calcul:

1. distanţa focală a obiectivului:
$$f'_{ob} = \frac{\Gamma}{\Gamma - 1} l . \qquad (5.46)$$
2. distanţa focală a ocularului:
$$f'_{oc} = \frac{1}{\Gamma - 1} l . \qquad (5.47)$$
3. unghiul de câmp imagine:
$$2\omega' = 2\mathrm{arctg}(\Gamma \mathrm{tg}\omega) . \qquad (5.48)$$
4. alegerea ocularului funcţie de setul de date (f'_{oc}, $2\omega'$, $s'_{P'}$).
5. diametrul pupilei de intrare:
$$D = \Gamma D' = 2h_{max} . \qquad (5.49)$$
6. deschiderea relativă a obiectivului:
$$q = \frac{D}{f'_{ob}} . \qquad (5.50)$$
7. se alege obiectivul funcţie de grosisment şi deschiderea relativă.
8. distanţa pupilei de ieşire de la focarul imagine al ocularului:

$$z'_{P'} = \frac{f'_{ob}}{\Gamma^2}. \qquad (5.51)$$

9. depărtarea efectivă a pupilei de ieşire:

$$s'_{P'} = s'_{F'oc} + z'_{P'}. \qquad (5.52)$$

10. grosismentul real (funcţie de obiectivul şi ocularul ales):

$$\Gamma = \frac{f'_{ob}}{f'_{oc}}. \qquad (5.53)$$

11. lungimea efectivă a tubului optic:

$$l = f'_{ob} + f'_{oc}. \qquad (5.54)$$

12. diametrul diafragmei de câmp:

$$D_C = 2f'_{ob}\,tg\omega. \qquad (5.55)$$

13. înălţimea de incidenţă pe obiectiv a razei pupilare marginale:

$$h_{1max} = \frac{D}{2}. \qquad (5.56)$$

14. unghiul dintre raza pupilară marginală şi axa optică, după obiectiv:

$$\sigma' = \mathrm{arctg}\left(\frac{h_{1max}}{f'_{ob}} + tg\omega\right). \qquad (5.57)$$

15. înălţimea de incidenţă a razei pupilare marginale pe ocular:

$$h_2 = l tg\sigma' - h_{1max}. \qquad (5.58)$$

16. diametrul util al ocularului:

$$D_{u\,oc} = 2h_2. \qquad (5.59)$$

5.5.3 Evaluarea calităţii imaginii. Calculul aberaţiilor reziduale

Având în vedere particularităţile subansamblurilor optice ale lunetelor (deschidere mare, fascicule puţin înclinate), în general, se verifică nivelul aberaţiei sferice şi a celei cromatice. La grosismente mari, se pot lua în considerare şi alte aberaţii geometrice (coma, distorsiunea, astigmatismul).

Aberaţiile denumesc generic erorile de formare a imaginilor.

Sistemele optice ideale asigură obţinerea unor imagini caracterizate prin stigmatism, planeitate şi ortoscopie, în condiţiile în care fasciculele care intră în sistem au deschidere mică, înclinări mici faţă de axa optică şi sunt monocromatice.

Sistemele optice reale lucrează cu fascicule largi şi înclinate, impuse de necesitatea iluminării corespunzătoare a planului imagine şi de mărimea câmpului obiect impus sistemului. Condiţiile reale de funcţionare determină manifestarea aberaţiilor geometrice. De asemenea, majoritatea aparatelor

lucrează în lumină albă, policromatică, a cărei utilizare introduce aberaţii cromatice.
Aberaţiile au ca efect scăderea calităţii imaginii. Aceasta poate deveni neclară, neasemenea cu obiectul şi colorată pe margini. De aceea, în proiectarea sistemelor optice un loc central este ocupat de corectarea aberaţiilor.

5.5.3.1 Aberaţia de sfericitate

Aberaţia de sfericitate se manifestă în fascicule largi. Datorită acestei aberaţii, conul luminos circular drept, corespunzător fasciculului divergent care porneşte dintr-un punct axial obiect, nu este transformat de sistem într-un con circular drept, corepunzător fasciculului convergent în punctul axial imagine. În realitate, în spaţiul imagine, fasciculul emergent are o formă spaţială cu generatoare neliniară şi o distribuţie energetică neuniformă. În diverse plane perpendiculare pe axa optică se obţin pete circulare de mărimi şi intensităţi luminoase diferite (fig. 5.13). Diametrul şi intensitatea luminoasă sunt invers proporţionale.

Aberaţia de sfericitate nu este generată de forma sferică sau abaterile de la sfericitate ale dioptrilor. Manifestarea aberaţiei sferice se datorează faptului că unghiurile de incidenţă pe suprafeţele refringente cresc pe măsura depărtării razelor de axa optică, iar refracţia este cu atât mai accentuată cu cât unghiul de incidenţă este mai mare. Ca urmare, razele emergente paraxiale se intersectează în punctul imagine cel mai îndepărtat de lentilă, A'. Punctele imagine extraaxiale vor fi cu atât mai apropiate de lentilă, cu cât unghiul razei incidente cu axa optică este mai mare (înălţimea de incidenţă este mai mare).

Înfăşurătoarea neliniară a razelor emergente poartă denumirea de caustică.

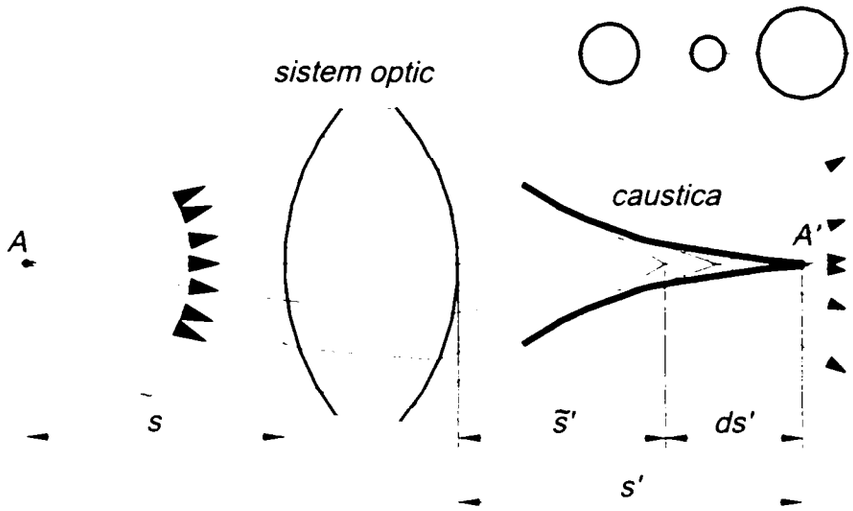
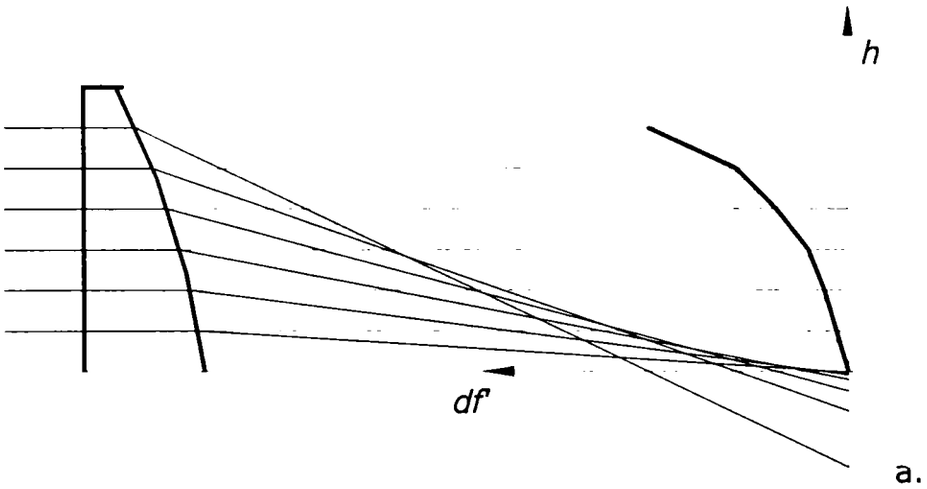


Fig.5.13

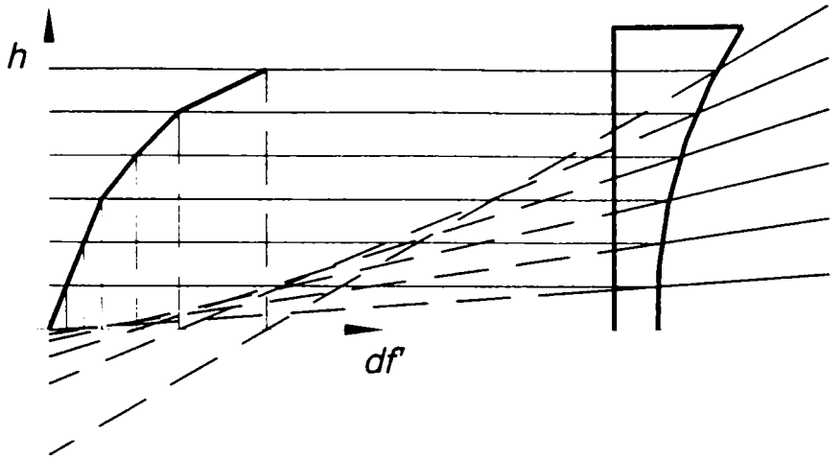
Aberaţia de sfericitate se redă grafic printr-o curbă reprezentând variaţia abscisei imagine sau a distanţei focale imagine (pentru obiect situat la infinit) în funcţie de deschiderea sistemului. Analitic, aberaţia de sfericitate se exprimă prin diferenţa maximă a absciselor sau a distanţelor focale ce provin de la un fascicul incident:

$$ds' = \tilde{s}' - s' \quad \text{sau} \quad ds' = \tilde{f}' - f' . \tag{5.60}$$

În cazul lentilelor convergente, aberaţia este negativă şi lentila se numeşte subcorectată (fig.5.14.a).



a.



b.

Fig.5.14

În cazul lentilelor divergente aberaţia se consideră pozitivă şi lentila se numeşte supracorectată, curba fiind orientată în sensul propagării razelor (fig.5.14.b).

Pentru aceeaşi distanţă focală şi deschidere, aberaţia de sfericitate se schimbă în funcţie de forma lentilei şi de orientarea acesteia faţă de planele conjugate obiect-imagine.

Printre metodele de corectare a aberaţiei sferice se enumeră şi următoarele:

- Asocierea unei lentile convergente (subcorectate) cu o lentilă divergentă (supracorectată), calculate astfel încât aberaţia negativă a lentilei convergente să fie compensată de aberaţia pozitivă a lentilei divergente, cel puţin la înălţimea maximă de incidenţă. Prin alegerea corespunzătoare a indicilor de refracţie ai celor două lentile se poate realiza totodată şi acromatizarea sistemului (corectarea aberaţiilor cromatice). Un astfel de dublet se numeşte acromat (fig. 5.15).

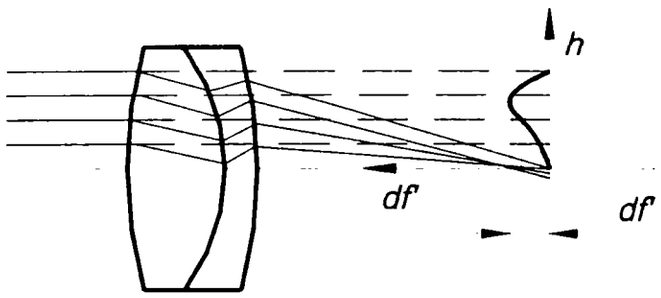


Fig.4.15

Chiar şi în cazul dubletului acromat, aberaţia de sfericitate nu poate fi corectată pentru toate deschiderile fasciculului. O anulare a aberaţiei se poate realiza numai pentru două zone ale lentilei, cea paraxială şi cea marginală. Pentru restul

zonelor se manifestă o aberaţie reziduală, de o valoare considerabil mai mică faţă de cea a lentilei simple necorectate.

- Înlocuirea unei componente unice cu un sistem format din mai multe lentile, având raze de curbură mai mari decât cele ale lentilei echivalente. În acest caz devierea razelor prin sistem fiind distribuită pe mai multe suprafeţe, unghiurile de incidenţă vor fi mai mici iar aberaţia de sfericitate redusă corespunzător.
- Utilizarea lentilelor cu suprafeţe asferice, astfel încât razele luminoase care traversează lentila prin diverse zone să prezinte un focar comun.

Aberaţia de sfericitate se exprimă analitic prin diferenţa ds′ dintre abscisa imagine extraaxială $\tilde{s}'_k$ (fig. 5.16) şi abscisa imagine paraxială s'_k (sau dintre distanţa focală extraaxială şi distanţa focală paraxială):

$$ds' = \tilde{s}'_k - s'_k \quad \text{sau} \quad ds' = \tilde{s}'_{F'} - s'_{F'} = df' . \qquad (5.61)$$

Abscisa imagine extraaxială $\tilde{s}'_k$ şi cea paraxială s'_k sau $s'_{F'}$ se obţin ca rezultat al unei drumuiri extraaxiale, respectiv a unei drumuiri paraxiale.

Mărimea ds', respectiv df' reprezintă aberaţia de sfericitate axială sau longitudinală. Aberaţia de sfericitate transversală sau de mărire, notată cu dy' (fig.5.16) se exprimă prin relaţia:

$$dy' = ds' . tg\tilde{\sigma}'_k . \qquad (5.62)$$

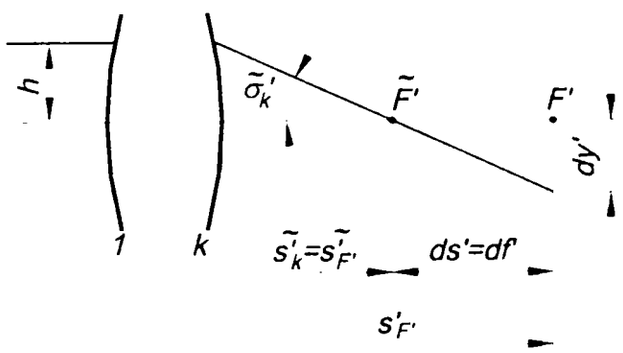


Fig.5.16

Evaluarea aberaţiei de sfericitate se face cu ajutorul setului de valori obţinut pentru diverse înălţimi de incidenţă cuprinse între $(0...h_{max}]$. Numărul intervalelor în care este împărţită înălţimea maximă de incidenţă depinde de mărimea acesteia şi de posibilităţile sistemului de calcul.

Pentru interpretarea rezultatelor, se obişnuieşte, pe lângă determinarea analitică a aberaţiei şi reprezentarea sa grafică, sub forma unei curbe h(ds').

Pentru a pune în evidenţă şi distribuţia luminoasă în planul imagine, se apelează la drumuirea vectorială, aplicată unei mulţimi de raze, care porneşte din punctul obiect şi este incidentă uniform pe pupila de intrare. Diagramele spot, rezultate grafice ale drumuirii vectoriale (fig. 5.12), au un caracter mai intuitiv decât curbele de variaţie în plan meridian şi se utilizează pe scară largă în ultimul timp.

Drumuirea vectorială, care are la bază calculul analitic al punctelor de incidenţă, emergenţă, al lungimilor şi cosinuşilor directori ai razelor, este singura metodă care permite studiul imaginilor pentru obiecte axiale, extraaxiale şi extraaxiale din afara planului meridian. Dezavantajul metodei constă în faptul că rezultă o caracterizare globală a calităţii imaginii, fără punerea în evidenţă în mod explicit a fiecărui tip de aberaţie geometrică. În principiu, nici nu este necesară exprimarea separată a fiecărei aberaţii geometrice, pentru că, în practică, este observabil întotdeauna efectul lor cumulat. În plus, determinarea analitică clasică a comei, astigmatismului, distorsiunii şi curburii câmpului este greoaie şi imprecisă, fiind bazată pe drumuirea trigonometrică aplicată unui punct axial, faţă de care se fac corecţii necesare caracterizării punctelor extraaxiale.

Ca aspect caracteristic pentru punctele axiale, distribuţia iluminării în planul imagine are o formă simetrică circulară, în jurul axei optice (fig.5.17). Densitatea punctelor din reprezentare este direct proporţională cu iluminarea. Se observă că în jurul punctului central, apar cercuri concentrice, de diamteru crescător şi iluminare descrescătoare.

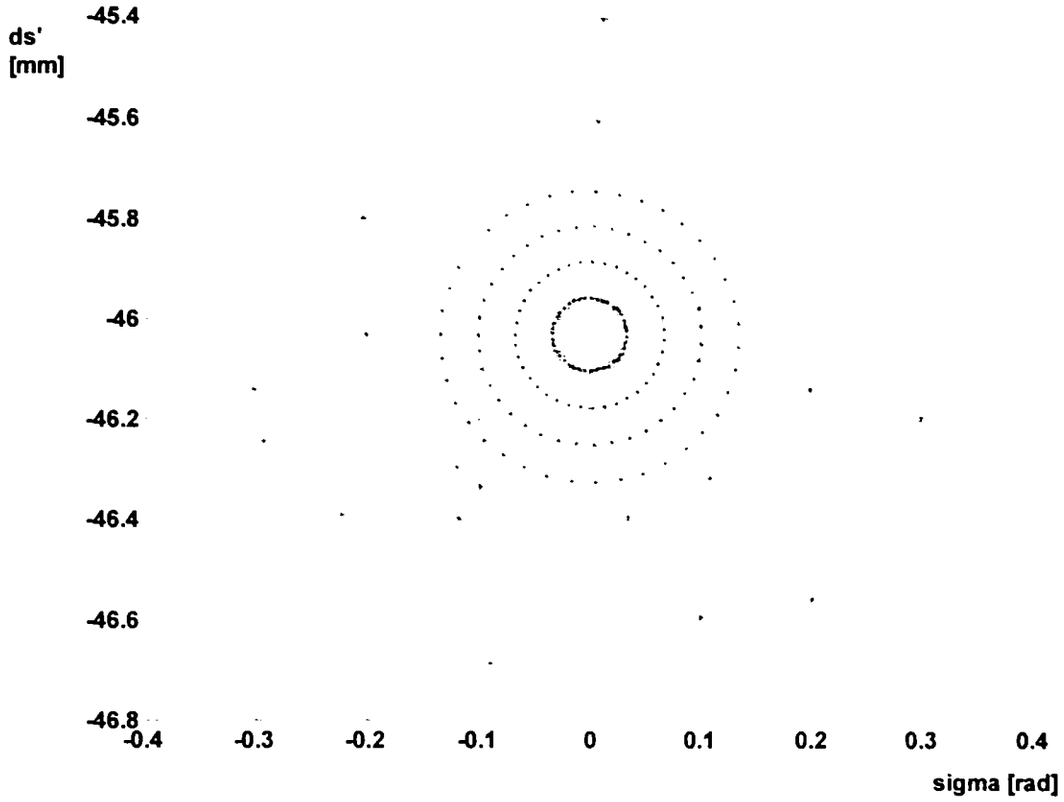


Fig.5.17

Pentru punctele extraaxiale (fig.5.18), în funcţie de poziţia lor în planul obiect, punctele luminoase se află pe curbe închise care nu mai sunt cercuri şi al căror grad şi sens de deformare depind de mărimea coordonatelor x şi y ale punctului obiect, de asemenea, de poziţia mărimea şi depătarea pupilei de intrare.

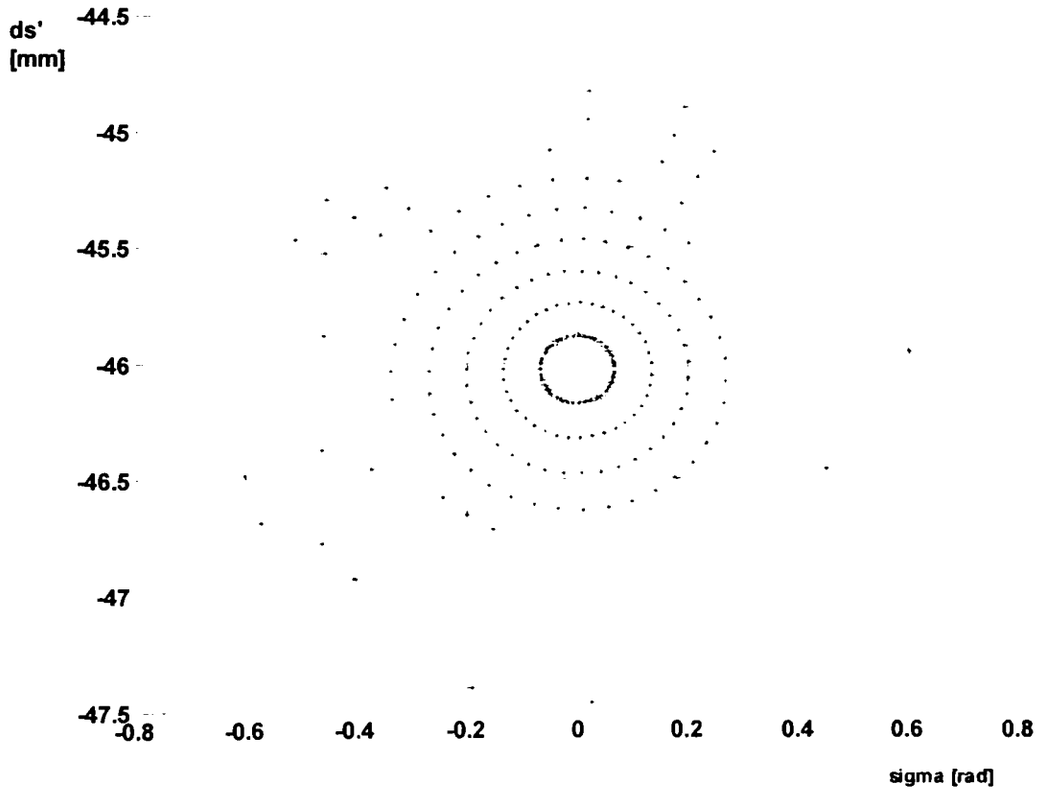


Fig.5.18

5.5.3.2 Aberaţiile cromatice

Aberaţiile cromatice se manifestă în cazul sistemelor traversate de lumină policromatică şi însoţesc aberaţiile geometrice fără a le influenţa sau a fi influenţate de acestea.

Aberaţiile cromatice apar datorită fenomenului de dispersie (variaţia indicelui de refracţie cu lungimea de undă) a sticlelor optice, care face ca, pentru diferite lungimi de undă, să se obţină diferite abscise imagine şi măriri transversale.

Aberaţia cromatică de poziţie, numită şi aberaţia cromatică longitudinală sau axială, are ca efect faptul că abscisele imagine sau focarele imagine pentru diverse lungimi de undă λ nu se suprapun (fig.5.19).

Prin acromazie sau dicromazie se înţelege egalitatea absciselor imagine pentru două radiaţii. Un sistem care satisface condiţia de acromazie se numeşte acromat.

Apocromazia sau tricromazia înseamnă egalitatea absciselor pentru trei radiaţii. Sistemele care satisfac condiţia de apocromazie se numesc apocromate sau tricromate.

Acromatele şi apocromatele sunt utilizate pe scară largă ca elemente singulare sau asociate cu alte componente în construcţia subansamblurilor optice.

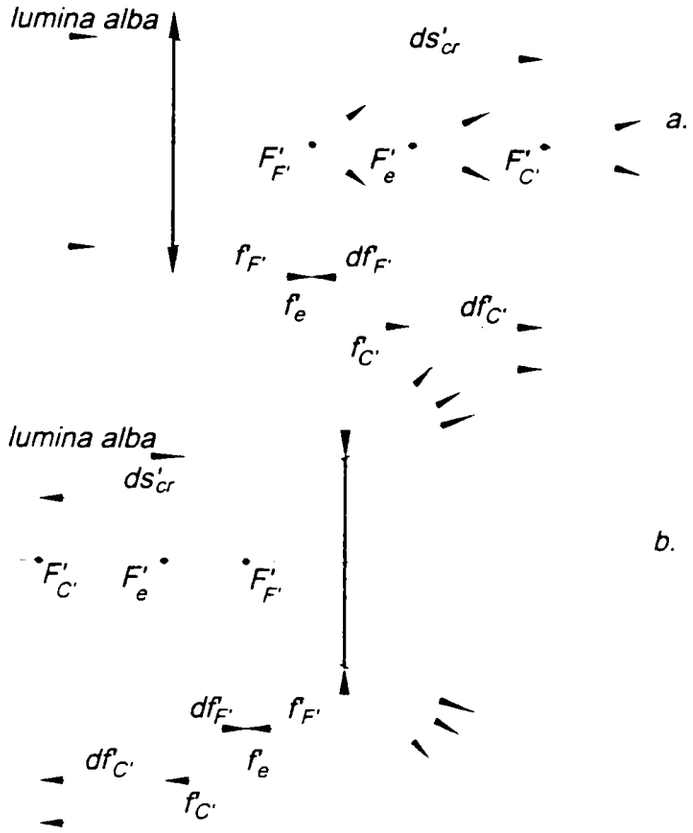


Fig.5.19

În spaţiul extraaxial, aberaţiile cromatice depind de înălţimea de incidenţă. În acest caz, este indicat studiul aberaţiilor cromatice împreună cu cele de sfericitate.

În mod frecvent, se apelează la reprezentarea grafică a sferocromatismului (aberaţia sferică la diverse lungimi de undă).

În general, aberaţia cromatică de poziţie se poate exprima prin relaţia:

$$ds'_{\lambda} = s'_{\lambda_1} - s'_{\lambda_2}, \qquad (5.63)$$

unde s'_{λ_1} este abscisa imagine corespunzătoare radiaţiei cu lungimea de undă λ_1; s'_{λ_2} - abscisa imagine corespunzătoare radiaţiei cu lungimea de undă λ_2.

Cel mai frecvent, sistemele optice lucrează cu ochiul. În acest caz se utilizează drept radiaţii de referinţă liniile F' şi C' (albastră şi roşie), iar relaţia (5.63) devine:

$$ds'_{cr} = s'_{F'} - s'_{C'}. \qquad (5.64)$$

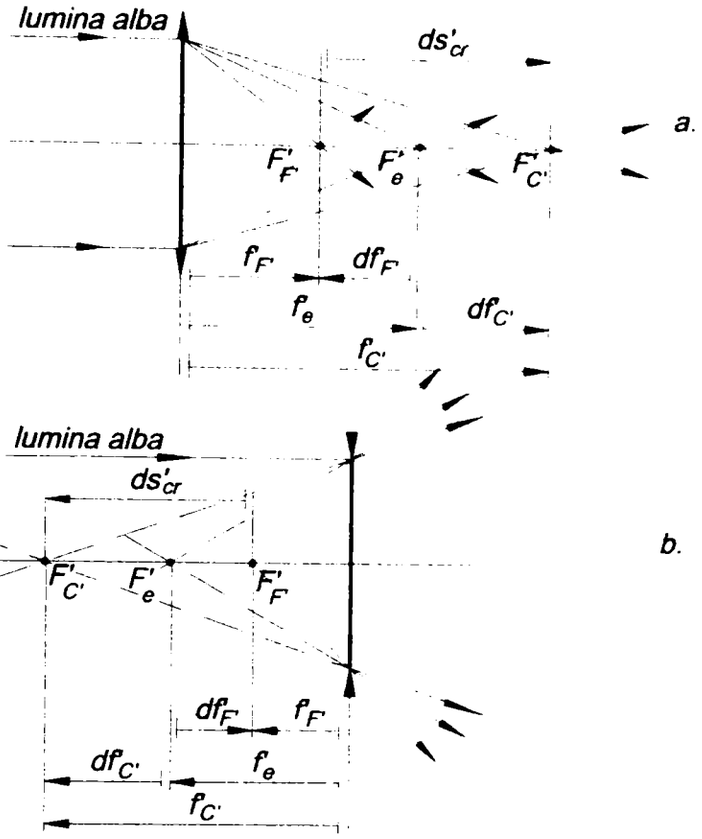


Fig.5.19

În spaţiul extraaxial, aberaţiile cromatice depind de înălţimea de incidenţă. În acest caz, este indicat studiul aberaţiilor cromatice împreună cu cele de sfericitate.

În mod frecvent, se apelează la reprezentarea grafică a sferocromatismului (aberaţia sferică la diverse lungimi de undă).

În general, aberaţia cromatică de poziţie se poate exprima prin relaţia:

$$ds'_{\lambda} = s'_{\lambda_1} - s'_{\lambda_2}, \qquad (5.63)$$

unde s'_{λ_1} este abscisa imagine corespunzătoare radiaţiei cu lungimea de undă λ_1; s'_{λ_2} - abscisa imagine corespunzătoare radiaţiei cu lungimea de undă λ_2.

Cel mai frecvent, sistemele optice lucrează cu ochiul. În acest caz se utilizează drept radiaţii de referinţă liniile F' şi C' (albastră şi roşie), iar relaţia (5.63) devine:

$$ds'_{cr} = s'_{F'} - s'_{C'}. \qquad (5.64)$$

În cazul lunetei, cele două subansambluri optice principale, obiectivul şi ocularul, se verifică la sfericitate şi cromatism. Analiza se face separat, pe subansamblu, apoi
valorile obţinute se compun pentru pentru a evalua imaginea dată de întregul sistem.

5.6 Program pentru proiectarea automată a lunetei Kepler

5.6.1 Organizarea proiectului

Pentru realizarea programului de proiectare automată a lunetei Kepler s-a ales limbajul Visual Basic.

Proiectul a necesitat un formular complex, care conţine controalele Visual Basic pentru introducerea datelor şi citirea rezultatelor şi un modul în care se înscrie codul programului (fig.5.20)

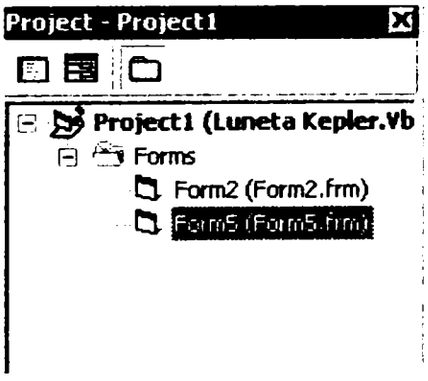


Fig.5.20

5.6.2 Baza de date

Baza de date conţine date referitoare la subansamblurile optice din construcţia lunetei şi caracteristicile sticlelor optice din care sunt confecţionate lentilele care intră în componenţă acestor subansambluri. Baza de date este ordonată în trei tabele, aşa cum rezultă din figura 5.21.

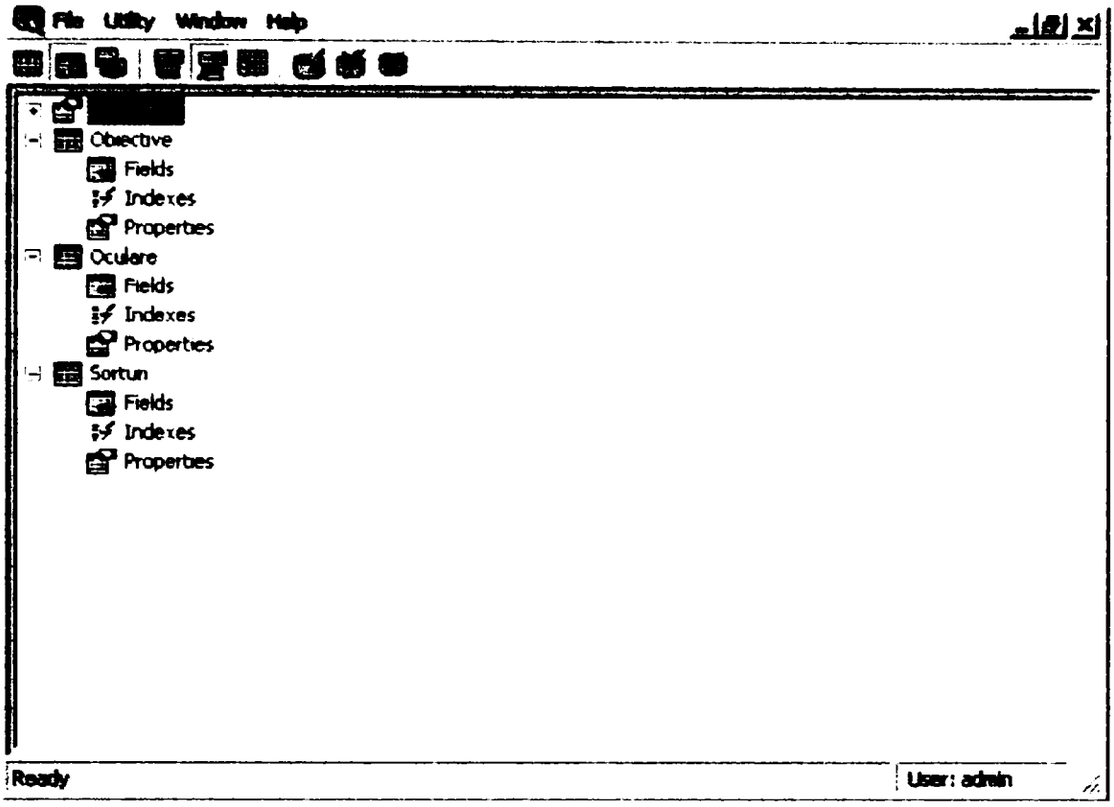


Fig.5.21

Primul tabel cuprinde informaţii complete, necesare alegerii şi verificării obiectivelor pentru lunete (fig.5.22).

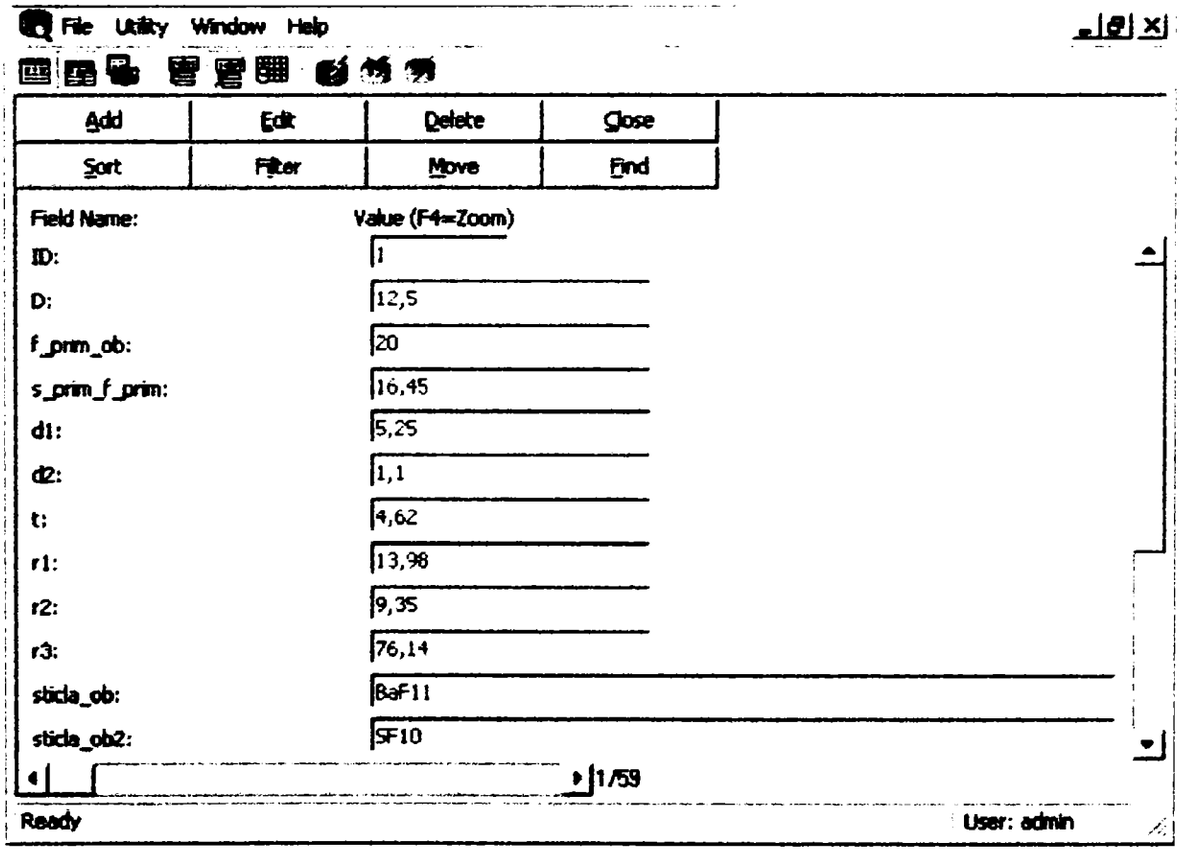


Fig.5.22

Elementul de identificare a obiectivului este distanţa focală, care reprezintă primul criteriu de alegere a subansamblului.

Stabilirea gabaritului axial al subansamblului se face cu ajutorul grosimilor la centru a lentilelor componente. Gabaritul radial este indicat prin deschiderea D.

Datele constructive ale lentilelor (raze şi grosimi) sunt utilizate în analiza aberaţională obiectivului.

Ocularele (fig.5.23), sunt identificate, de asememnea, prin distanţa lor focală, utilizată ca prim criteriu de alegere. Tabelul destinat ocularelor cuprinde aceleaşi date ca şi cel referitor la obiective, la care se adaugă câmpul unghiular imagine şi abscisa pupilei de ieşire, care reprezintă criterii secundare de eligibilitate.

Al treilea tabel din baza de date conţine elementele necesare caracterizării materialelor optice din care sunt confecţionate obiectivele şi ocularele cuprinse în tabelele anterioare (fig.5.23, 5.24).

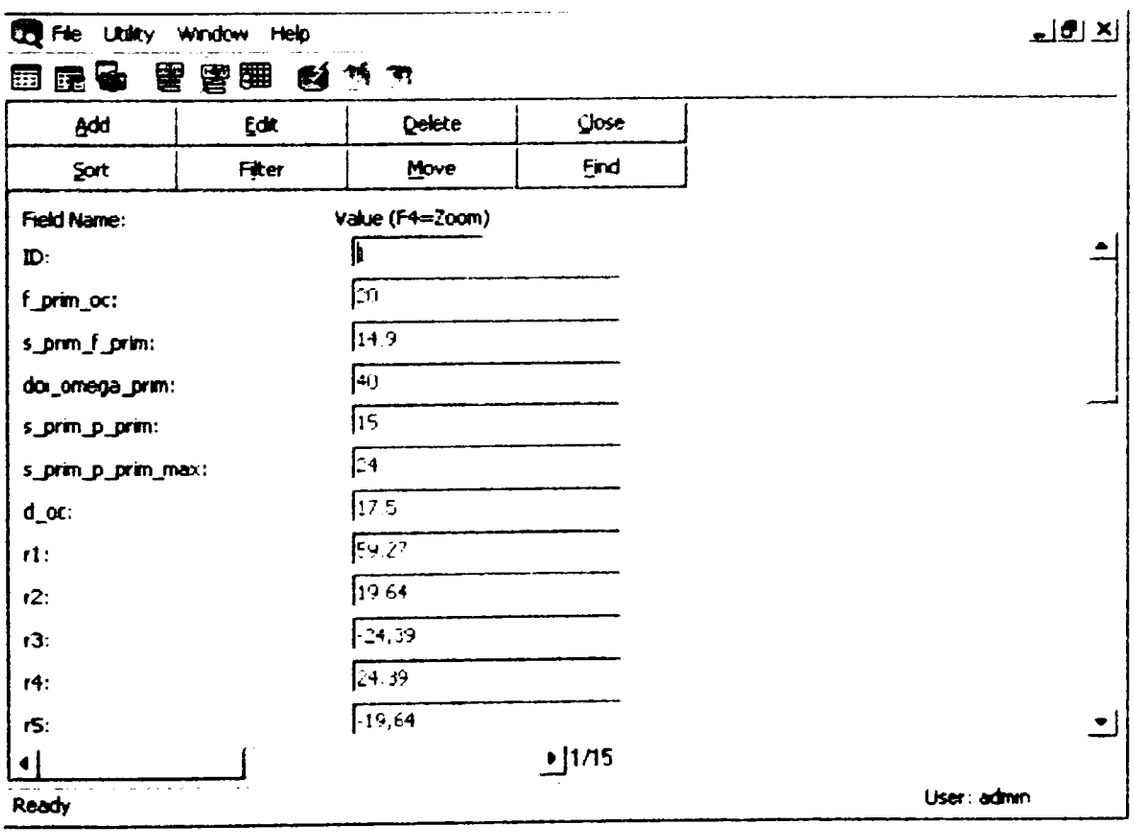


Fig.5.23

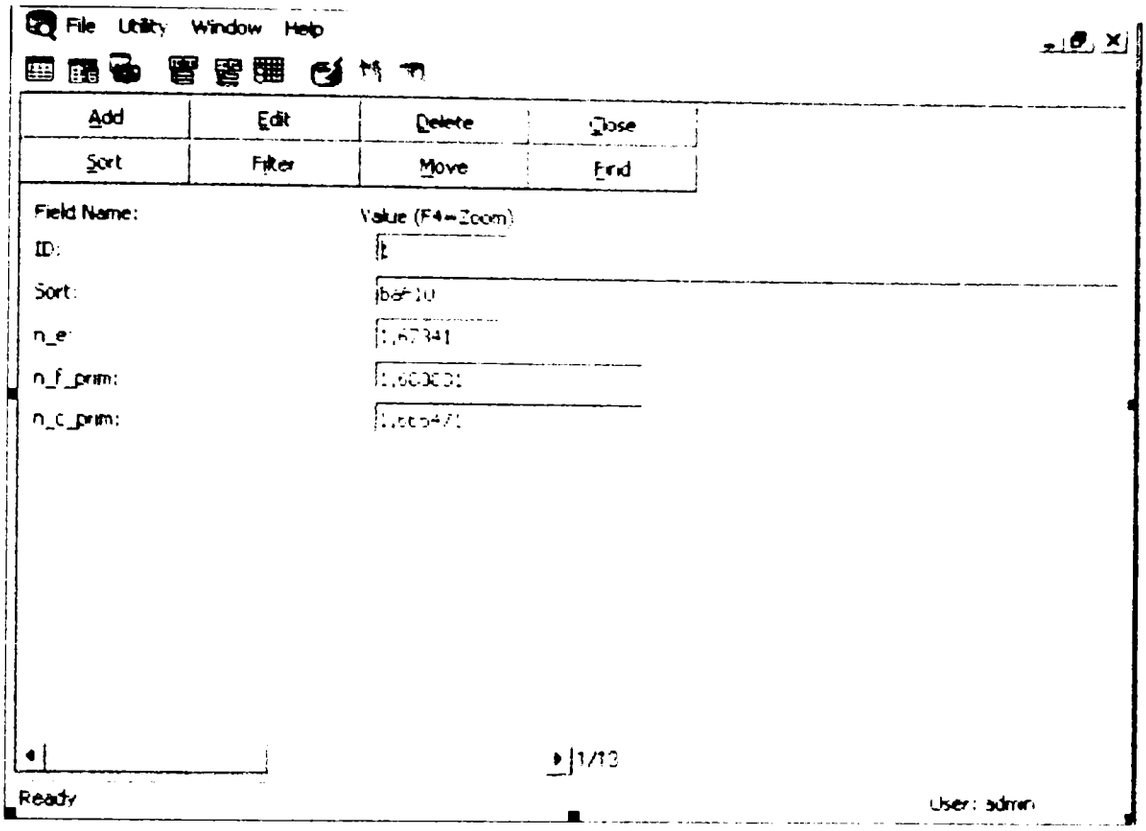


Fig.5.24

5.6.3 Formularul

Formularul are un conţinut complex, organizat sub forma unui şir de Tab-uri alocate unor date specifice (fig.5.25)

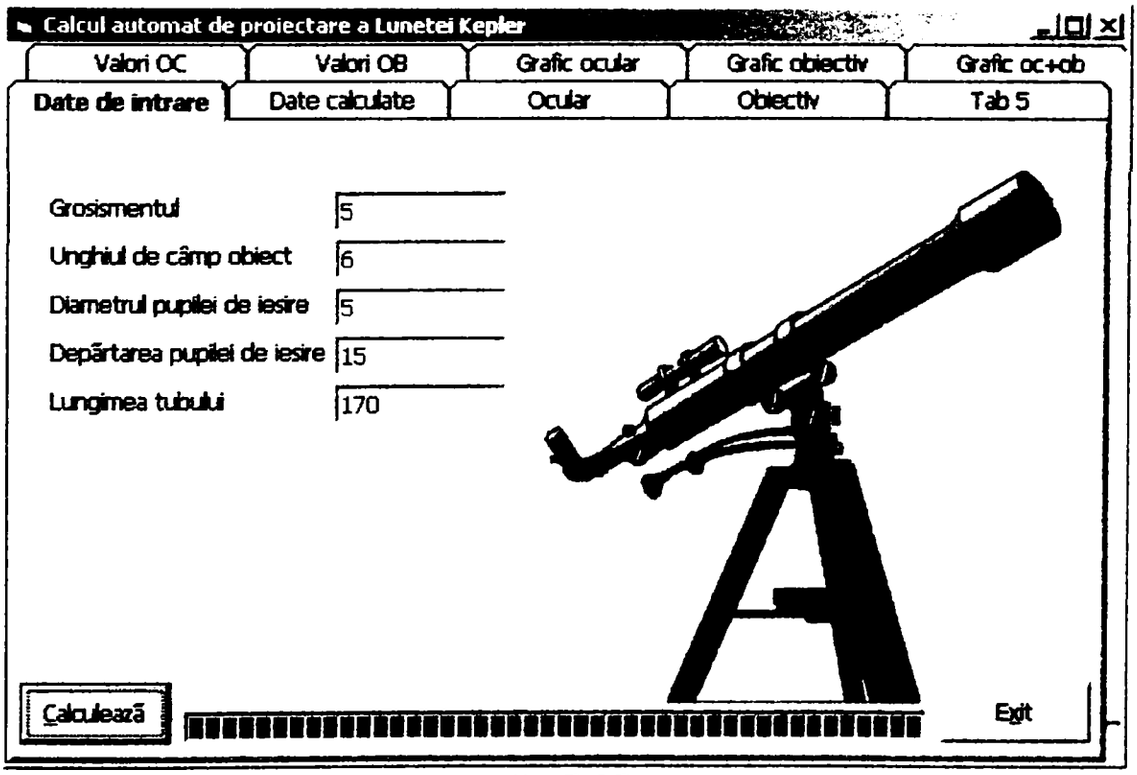


Fig.5.25

Tab-ul "Date de intrare" permite utilizatorului introducerea datelor de proiectare.

Rezultatele calculului pot fi vizualizate în tab-ul "Date calculate" (fig.5.26).

Tab-ul "Ocular" conţine caracterizarea optică şi geometrică completă a ocularului selectat din baza de date conform criteriilor înscrise în codul programului (fig.5.27)

Aceleaşi tipuri de date referitoare la obiectiv sunt afişate în tab-ul "Obiectiv" (fig.5.28).

Tab-urile "Valori OB" şi "Valori OC" permit vizualizarea subansamblurilor obiectiv şi ocular extrase din baza de dare pe baza criteriului distantei focale (fig. 5.29 şi 5.30).

Ultimile trei tab-uri prezintă sub forma grafică variaţia aberaţiilor reziduale sferice şi cromatice pentru ocular (fig.5.31), obiectiv (fig.5.32) şi întregul sistem optic (fig. 5.33).

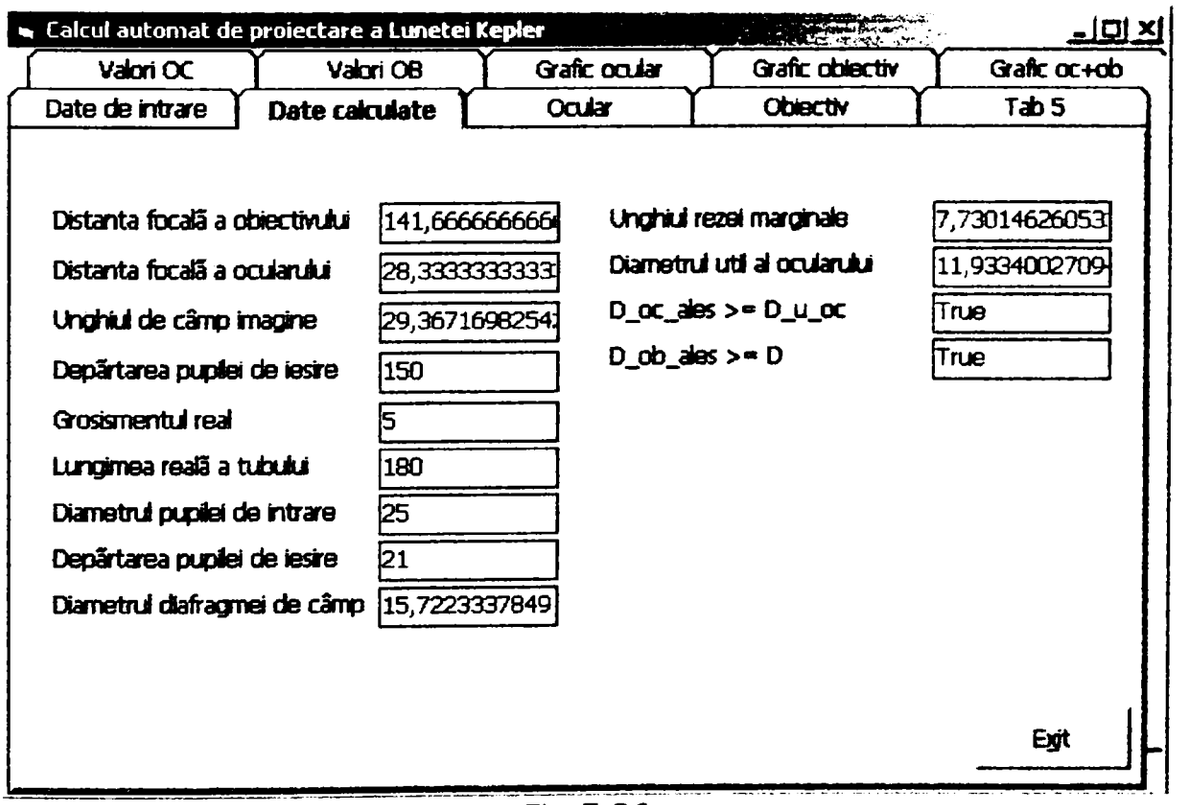


Fig.5.26

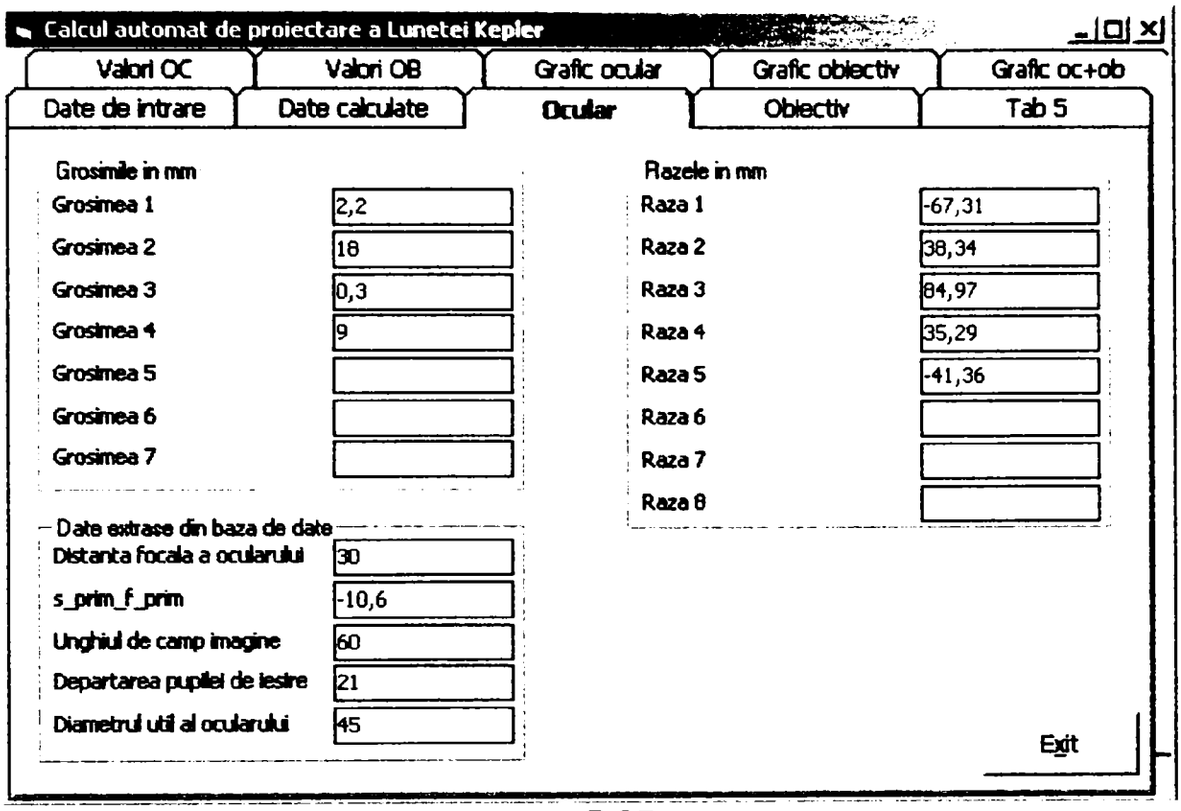


Fig.5.27

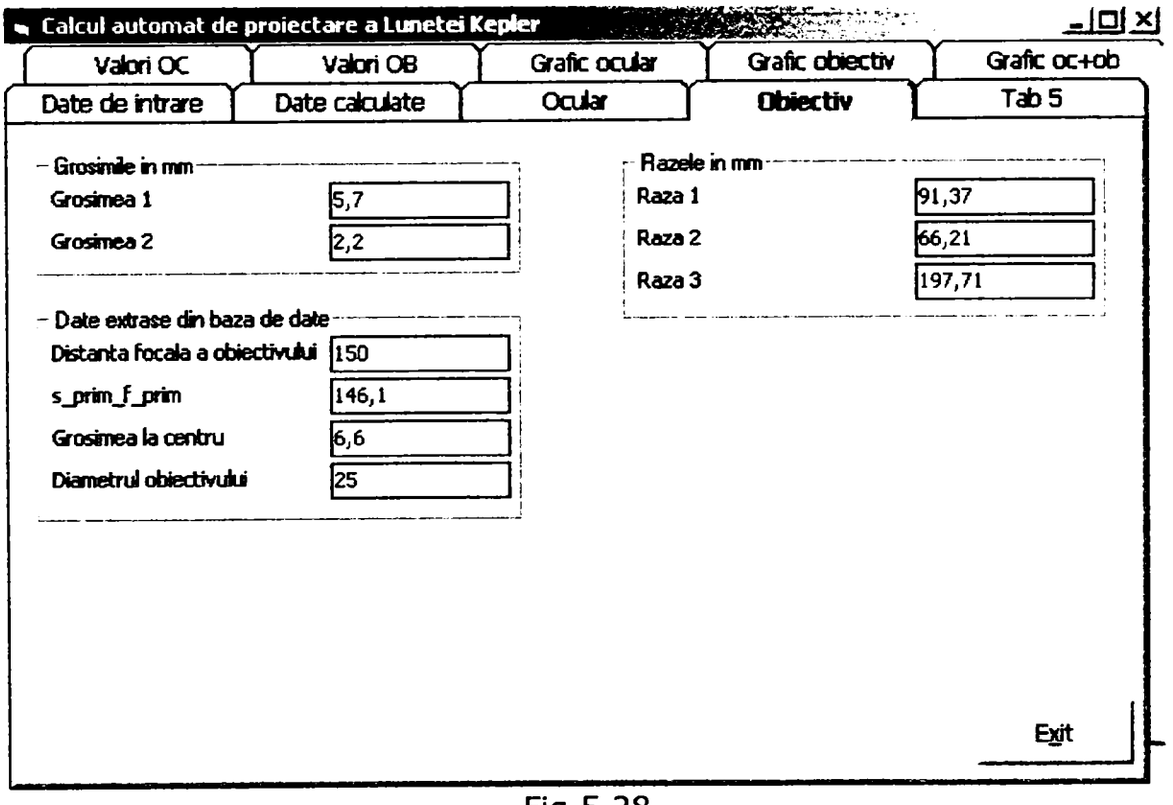


Fig.5.28

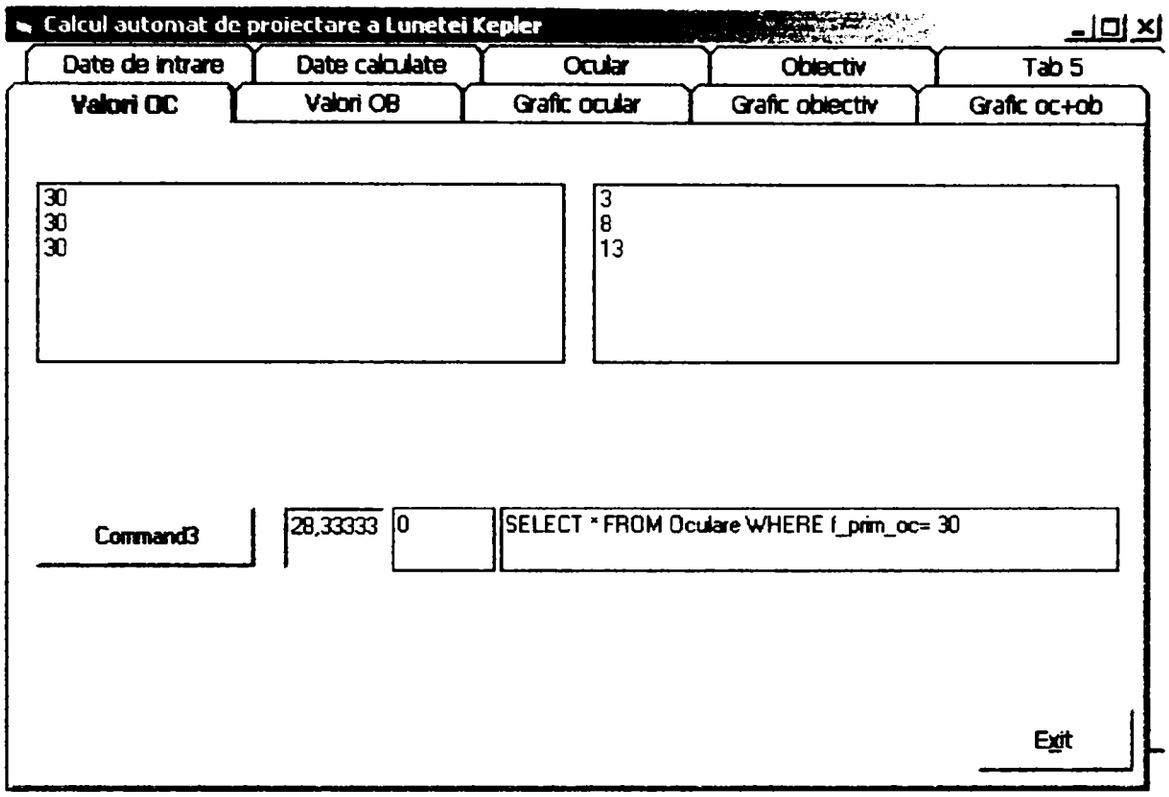


Fig.5.29

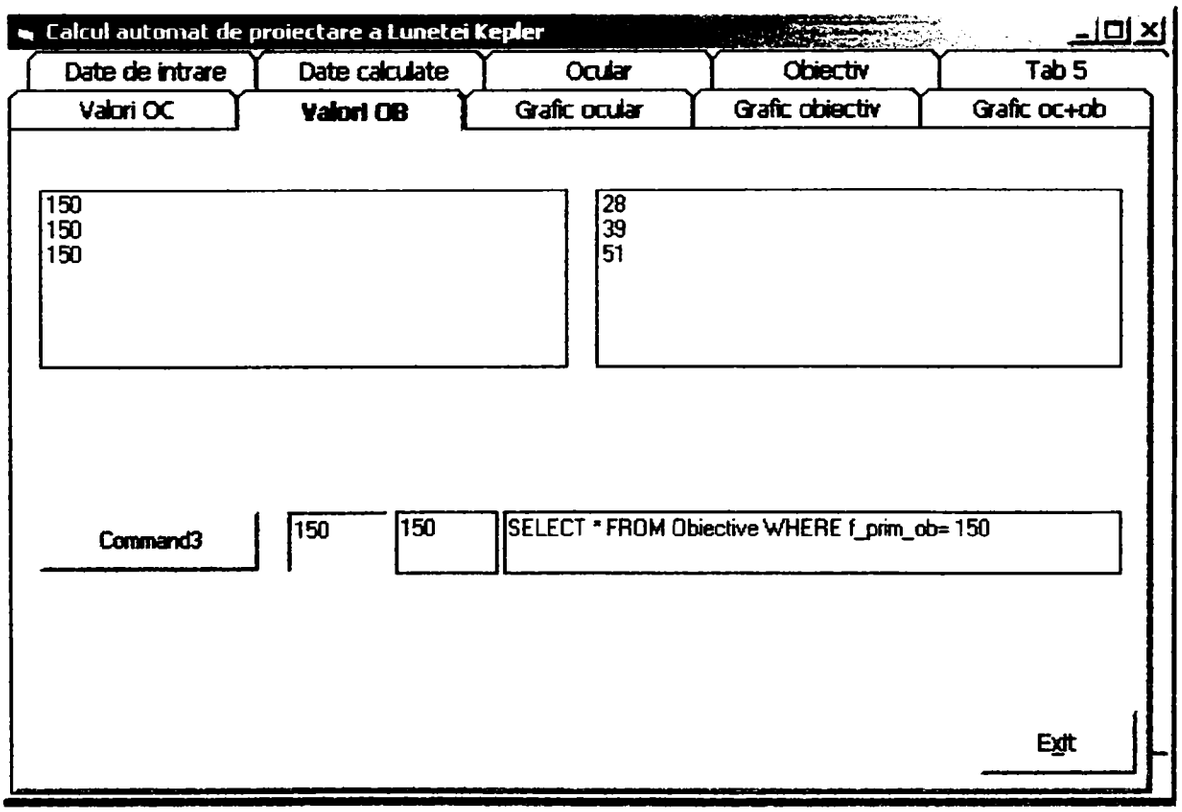


Fig.5.30

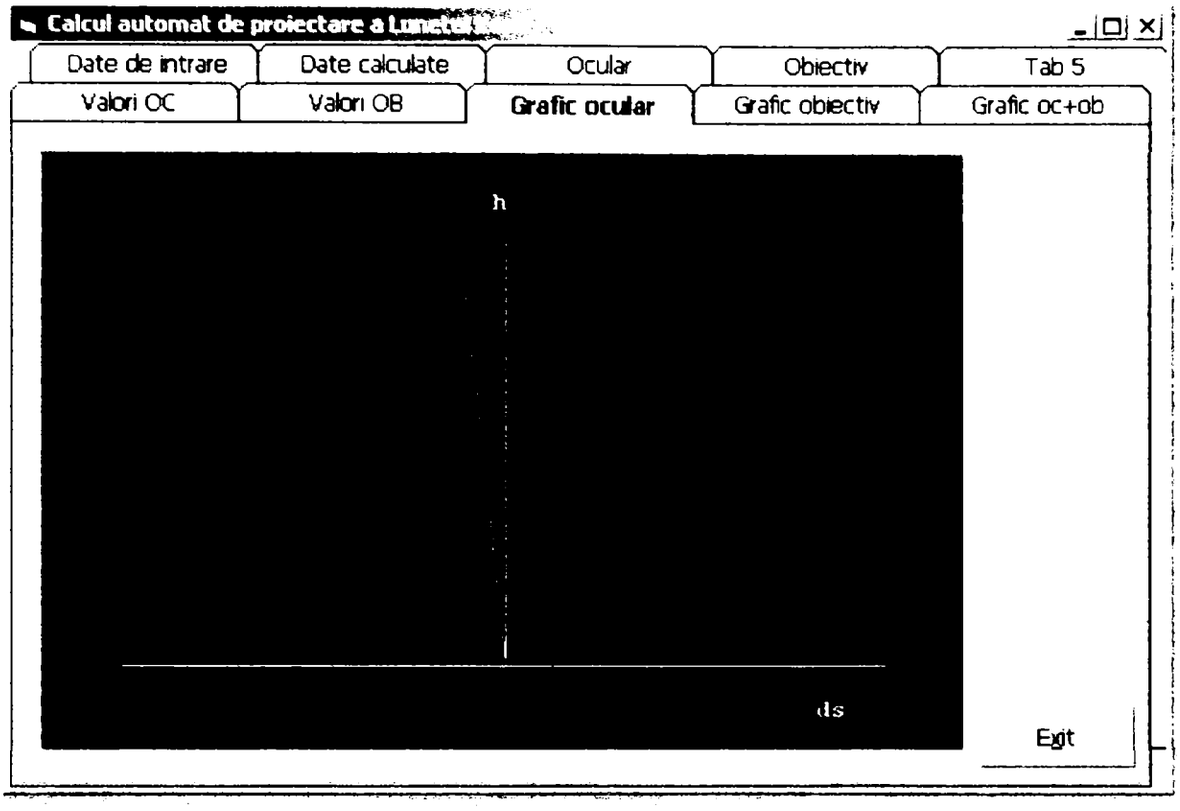


Fig. 5.31

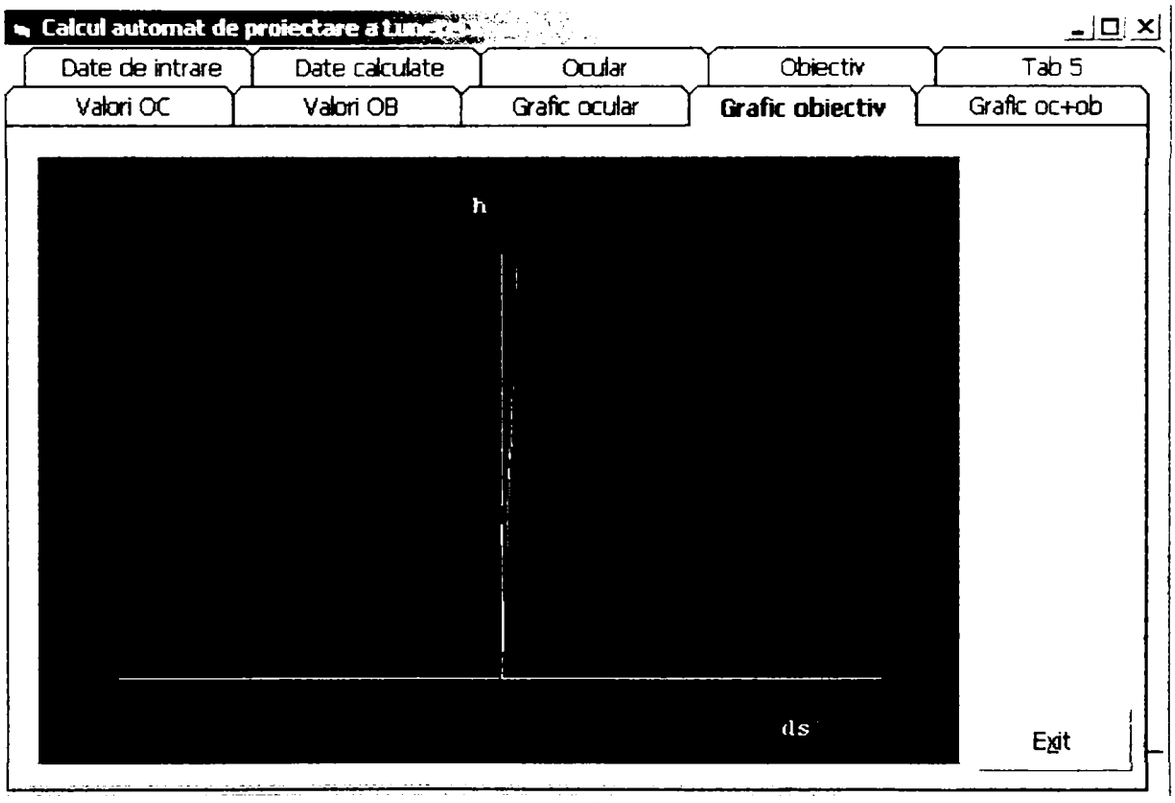


Fig. 5.32

- Calculul laborios şi precizia ridicată necesară impune introducerea automatizării în proiectarea sistemelor optice
- Optimizarea sistemelor optice trebuie să ia în considerare criterii specifice prin care să se exprime calitatea imaginii.

Pe parcursul capitolului au fost introduse elemente cu caracter de originalitate, care pot fi prezentate drept contribuţii ale autoarei. Printre acestea se pot enumera şi următoarele:

- Realizarea unei scheme sintetice şi intuitive privind construcţia modulară a echipamentelor optice
- Elaborarea unei scheme logice cu caracter de maximă generalitate pentru proiectarea sistemelor optice
- Stabilirea unei configuraţii caracteristice pentru o bază de date apelabilă în proiectarea automată a sistemelor optice
- Sintetizarea şi clasificarea cunoştintelor teoretice şi a recomandărilor practice referitoare la criteriile de calitate a imaginii
- Stabilirea etapelor de dimensionare, sinteză şi analiză, necesare proiectării optime a sistemelor optice
- Particularizarea metodologiei generale de proiectare pentru cazul sistemelor optice afocale
- Elaborarea unui program expert complex destinat proiectării automate a sistemelor afocale, în limbajul Visual Basic. Se pot evidenţia următoarele aspecte:

- programul apelează o bază de date originală, care conţine toate datele necesare pentru alegerea şi evaluarea subansamblurilor optice de bază: obiectiv şi ocular
- codul programului transcrie în forma accesibilă sistemului de calcul toate etapele de proiectare a sistemelor optice afocale
- introducerea datelor iniţiale şi a rezultatelor intermediare sau finale este asigurată printr-un formular complex, dar uşor de accesat. Rezultatele pot fi vizualizate atât în foma numerică, cât şi în cea grafică.

6 Concluzii finale

Lucrarea de faţă este pe de o parte o sinteză a experienţei autoarei în domeniul activitatii de proiecte şi o încercare de apropiere a informaticii de domenii, cum ar fi proiectarea asistata de calculator in construcţia de maşini şi echipamente, precum şi de aplicare a unor metode din domeniul dezvoltării software, în proiecte tehnice, cu scopul îmbunătăţirii calitative a procesului de proiectare.

Un rol esenţial se acordă în această lucrare pe de o parte arhitecturii, privită sub mai multe aspecte: arhitectura proceselor, arhitectura de sistem, arhitectura de aplicaţie, precum şi calităţii, pe de alta parte, privită sub aspectul „Total Quality Management".

Principiile teoretice prezentate in lucrare sunt ilustrate pe baza unor exemple din domeniul proiectării asistate de calculator, o parte dintre ele fiind publicate si puse in practica in proiecte la care a participat si participa autoarea in cadrul concernului german HDI unde isi desfasoara activitatea in domeniul conducerii de proiecte si arhitectura sistemelor. Aplicatiile din domeniul aparatelor optice au fost realizate in colaborare cu Facultatea de Mecanica din Universitatea „Politehnica" Timisoara.

Ca o sinteză a concluziilor formulate la sfârşitul fiecărui capitol, putem menţiona următoarele:

Principiile unei arhitecturi moderne centrată pe componente stau la baza sistemelor tehnice flexibile şi adaptabile. Printr-o arhitectură bazată pe componente şi posibilitatea de configurare se pot înlănţui diverse procese de specialitate în cazul proceselor variabile.

Utilizarea tehnicii de calcul în proiectarea şi construcţia de echipamente dobândeşte o pondere crescândă. Metodele de analiză şi design orientate pe obiecte sunt, de asemenea, utile şi în activitatea de proiectare din afara domeniului software, reprezentând un mijloc de comunicare unitar în cadrul unei echipe de proiectare mixte.

Noţiunile de clasă şi instanţă din teoria orientată pe obiecte se pot utiliza cu succes şi la activitatea de proiectare asistată de calculator, dobândind astfel un caracter de generalitate. Această idee stă la baza unei soluţii generalizate (pattern) prezentate în capitolul 4. Conform acestui pattern, activităţile de proiectare se specializează în macrodesign şi microdesign, ceea ce conduce la creşterea eficienţei şi a calităţii în activitatea de proiectare.
Proiectarea sistemelor optice necesită parcurgerea unui traseu bine stabilit, care include etapele: stabilirea schemei optice, calculul de gabarit, sinteza componentelor, evaluarea calităţii imaginii, alegerea soluţiilor constructive pentru componentele mecanice. Calculul laborios şi precizia ridicată necesară impune introducerea automatizării în proiectarea sistemelor optice.

Limbajele de programare orientate pe obiecte reprezintă la ora actuala mijlocul ideal pentru realizarea componentelor software.

In lucrare s-au introdus numeroase aspecte cu caracter de inovaţie, o parte dintre ele fiind finalizate sub formă de sisteme software. Pe lângă analiza şi sinteza materialului bibliografic şi a exemplificării metodei de modelare orientate pe obiecte în analiza, design-ul şi implementarea componentelor software, se pot evidenţia următoarele:

În domeniul arhitecturii de componente, sunt descrise alternative pentru componente reutilizabile. Lucrarea conţine un pattern pentru cuplajul componentelor, inclusiv algoritmul de transformare a unor structuri complexe pentru parametrii de interfaţă. Posibilitatea cuplării componentelor stă la baza realizării unei arhitecturi bazate pe componente şi a modelării proceselor variabile.

In lucrare se propune o optimizare a procesului de proiectare prin specializarea activităţilor de proiectare corespunzător fazelor macro- şi microdesign, fiind descris un pattern pentru soluţionarea acestei probleme (elemente de constructie abstracte si elemente de constructie concrete).

Lucrarea conţine, de asemenea, soluţii generalizate pentru proiectarea de variante. Astfel, este descris un pattern pentru implementarea unui limbaj de

configurare a variantelor. O alta alternativă de soluţie pentru proiectarea variantelor se realizează prin gruparea pattern-ului „elemente de constructie abstracte si elemente de constructie concrete" cu cel pentru cuplajul componentelor, utilizând un algoritm complex de traducere a structurilor.

Construcţia de documente din elemente de constructie şi reguli de construcţie precizate în faza de macrodesign, analog funcţionării unui sistem expert, este soluţionată sub forma combinaţiei unor componente. Aceasta soluţie are, de asemenea, un caracter general şi poate fi privită ca un pattern.

Un alt pattern descris în această lucrare se referă la gestionarea istoriei, permiţând reconstituirea în funcţie de timp a versiunilor de obiecte proiectate.
Lucrarea conţine, de asemenea, o aplicaţie din domeniul proiectării sistemelor optice finalizată printr-un program expert complex destinat proiectării automate a sistemelor afocale, ţinând cont de criteriile de calitate a imaginii.

Anexe

A Fragmente de program (sistemul CAD)

In cele ce urmeaza, se va exemplifica implementarea editorului de expresii conform modelului descris in capitolul al treilea, prin cateva secvente de cod reprezentative, extrase dintr-un proiect realizat de autoare in cadrul concernului german HDI. Pentru implementarea sistemului CAD s-a recurs la limbajul Smalltalk.

```
((CADAusdruckApp createSubApplication: #CADAusdruckView in: 'true') isNil)
    ifTrue: [SubApplication errorCouldNotCreate: #CADAusdruckView]!

CADAusdruckView becomeDefault!
AbtAppBldrView subclass: #CADAusdruckeditorView
    instanceVariableNames: 'leftClass version result ausdruckListe treeViewController
containerDetailsController plausi '
    classVariableNames: ''
    poolDictionaries: ''!

CADAusdruckView becomeDefault!
AbtAppBldrView subclass: #CADWertEditor
    instanceVariableNames: 'result anzeigemodus destinationClass waehrung '
    classVariableNames: ''
    poolDictionaries: ''!

CADAusdruckView becomeDefault!
AbtAppBldrView subclass: #CADWertView
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''!

CADAusdruckView becomeDefault!
SubApplication subclass: #CADAusdruckView
    instanceVariableNames: ''
    classVariableNames: ''
    poolDictionaries: ''!


CADAusdruckView becomeDefault!

!CADAusdruckeditorView class publicMethods !

new

^super new initialize!

open

        | aView |
```

```
	aView := self new.
	"aView initializeParts."
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	!

openOn: aVersion

	| aView |
	aView := self new.
	aView version: aVersion.
	aView installTreeViewController.
	aView installContainerDetailsController.
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openOn: aVersion ausdruck: anAusdruck

	| aView |
	aView := self new.
	aView version: aVersion.
	aView installTreeViewController.
	aView installContainerDetailsController.
	anAusdruck isDataAvailable
		ifTrue:
			[aView ausdruckListe addAll:
				((anAusdruck deepCopyZurOwner: anAusdruck owner)
asKomponentenListe: OrderedCollection new)].
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openOn: aVersion plausi: aSymbol

	| aView |
	aView := self new.
	aView version: aVersion.
	aView plausi: aSymbol.
	aView installTreeViewController.
	aView installContainerDetailsController.
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openOn: aVersion plausi: aSymbol ausdruck: anAusdruck

	| aView |
	aView := self new.
```

```
	aView version: aVersion.
	aView plausi: aSymbol.
	aView installTreeViewController.
	aView installContainerDetailsController.
	anAusdruck isDataAvailable
		ifTrue:
			[aView ausdruckListe  addAll:
				((anAusdruck deepCopyZurOwner: anAusdruck owner)
asKomponentenListe: OrderedCollection new)].
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openRightMenuOn: aVersion

	| aView |
	aView := self new.
	aView version: aVersion.
	aView installTreeViewController.
	aView installContainerDetailsController.
	aView disableLeftMenu.
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openRightMenuOn: aVersion ausdruck: anAusdruck

	| aView |
	aView := self new.
	aView version: aVersion.
	aView installTreeViewController.
	aView installContainerDetailsController.
	aView disableLeftMenu.
	anAusdruck isDataAvailable
		ifTrue:
			[aView ausdruckListe  addAll:
				((anAusdruck deepCopyZurOwner: anAusdruck owner)
asKomponentenListe: OrderedCollection new)].
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.

	^aView result.
	!

openRightMenuOn: aVersion leftClass: aClass

	| aView |
	aView := self new.
	aView version: aVersion.
	aView installTreeViewController.
	aView installContainerDetailsController.
```

```
        aView leftClass: aClass.
        aView disableLeftMenu.
        aView initializeParts.
        aView openApplicationModalWidget.
        aView suspendExecutionUntilRemoved.

        ^aView result.
        !

openRightMenuOn: aVersion leftClass: aClass ausdruck: anAusdruck

        | aView |
        aView := self new.
        aView version: aVersion.
        aView installTreeViewController.
        aView installContainerDetailsController.
        aView leftClass: aClass.
        aView disableLeftMenu.
        anAusdruck isDataAvailable
                ifTrue:
                        [aView ausdruckListe  addAll:
                                ((anAusdruck deepCopyZurOwner: anAusdruck owner)
asKomponentenListe: OrderedCollection new)].
        aView initializeParts.
        aView openApplicationModalWidget.
        aView suspendExecutionUntilRemoved.

        ^aView result.
        !

openRightMenuOn: aVersion leftClass: aClass withVariable: aVariable

        | aView hist aOrderedCollection |
        aView := self new.
        aView version: aVersion.
        aView installTreeViewController.
        aView installContainerDetailsController.
        aView leftClass: aClass.
        aView disableLeftMenu.
        hist := aVariable getZurVersion: aVersion.
        aOrderedCollection := OrderedCollection new.
        aOrderedCollection add: hist wert.
        hist isDataAvailable
                ifTrue:
                        [aView ausdruckListe  addAll: aOrderedCollection].
        aView initializeParts.
        aView openApplicationModalWidget.
        aView suspendExecutionUntilRemoved.
        ^aView result.
        ! !

!CADAusdruckeditorView class privateMethods !
.....

!CADAusdruckeditorView publicMethods !

abbruchButtonClicked
```

```
        self closeTreeViewController.
        self closeContainerDetailsController.
        (self subpartNamed: 'Window1') closeWidgetCommand.!

ausdruckAuswerten

|aWert paramList|
paramList := (self subpartNamed: 'lstAusdruecke') selectedItems.
paramList do:
        [:each|
        aWert := CADWertAlphanumerisch new.
        aWert wert: (each wertFuer: self version) printString.
        self ausdruckListe add: aWert.
        (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe.
        (self subpartNamed: 'lstAusdruecke') deselectAll]


!

ausdruckListe

ausdruckListe isNil
        ifTrue:
                [ausdruckListe := OrderedCollection new].
^ausdruckListe!

ausdruckListe: aCollection

ausdruckListe := aCollection!

ausdruckLoeschen

|paramList k|

paramList := (self subpartNamed: 'lstAusdruecke') selectionIndices.
k := 0.
paramList do: [:each|
        (self ausdruckListe removeAtIndex: (each - k)).
        k := k + 1 ].
(self subpartNamed: 'lstAusdruecke') items: self ausdruckListe.
(self subpartNamed: 'lstAusdruecke') deselectAll


!

cdvLabelBlock

        ^[      :object :index | | res |
                        res := nil.
                        index = 1
                                ifTrue:[res := object variableStamm symbol].
                        index = 2
                                ifTrue: [res := object wertFuer: self version.
                                                (res isKindOf: Boolean)
                                                        ifTrue:
                                                                [res = true
```

```
                                                                    ifTrue: [res :=
'Ja'].
                                                            res = false
                                                                    ifTrue: [res :=
'Nein']]].
                        index = 3
                                ifTrue:[res := (CADWert getClassViaTyp: (object
variableStamm wertTyp)) text].
                        index = 4
                                ifTrue:[(object mussFeldFuer: self version) = true
                                                        ifTrue: [res := 'Ja']
                                                        ifFalse: [res := 'Nein'] ].
                        index = 5
                                ifTrue:[(object tarifmerkmalFuer: self version) = true
                                                        ifTrue: [res := 'Ja']
                                                        ifFalse: [res := 'Nein'] ].
                        res.
                ]!

closeContainerDetailsController

        self containerDetailsController close!

closeTreeViewController

                self treeViewController close!

containerDetailsController

        ^containerDetailsController!

containerDetailsController: aController

        containerDetailsController := aController!

createZugriffAusdruck

|ausdr aCollection|

aCollection :=  (self subpartNamed: 'lstAusdruecke') selectedItems.
(aCollection size = 5)
        ifTrue:
                [ausdr := CADAusdruckZugriff new.
                ausdr symbol: (aCollection at: 3) wert;
                                        symbolTyp: (aCollection at: 2) wert asSymbol;
                                        zugriffTyp: (aCollection first) wert;
                                        zugriffKey: (aCollection at: 4) wert;
                                        merker: (aCollection at: 5) wert;
                                        owner: self version versioniertesObjekt;
                                        ownerTyp: self version versioniertesObjekt class typ.
                        self ausdruckListe add: ausdr.
                        (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen mindestens einen Parameter
selektieren!!']!

disableLeftMenu
```

```
(self subpartNamed: 'MenuZugriffUebernehmen') disable
!

displayTree

self treeViewController display
        !

einfuegenAlphanumeric

self einfuegenKonst: CADWertAlphanumerisch
!

einfuegenAnd

self einfuegenBin: CADAusdruckAnd
!

einfuegenBin: aClass

|ausdr paramList a b c|
paramList := (self subpartNamed: 'lstAusdruecke') selectedItems.
(paramList size = 2)
        ifTrue:
                [ausdr := aClass new.
                ausdr owner: self version versioniertesObjekt;
                                ownerTyp: self version versioniertesObjekt class typ .
                ausdr erstesAusdruck: (paramList first deepCopyZurOwner: ausdr).
                ausdr zweitesAusdruck: (paramList last deepCopyZurOwner: ausdr).
                ausdr isPlausi
                        ifTrue:
                                [self ausdruckListe add: ausdr.
                                (self subpartNamed: 'lstAusdruecke') items: self
ausdruckListe]
                        ifFalse:
                                [CADInfoPrompter promptFor: 'Falsche Definition']
                ]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen zwei Parameter selektieren!!']
!

einfuegenBoolean

self einfuegenKonst: CADWertBoolean
!

einfuegenDate

self einfuegenKonst: CADWertDatum
!

einfuegenDiv

self einfuegenBin: CADAusdruckDiv
!

einfuegenGefuellt
```

```
self einfuegenUn: CADAusdruckGefuellt
!

einfuegenGleich

self einfuegenBin: CADAusdruckGleich !

einfuegenGroesser

self einfuegenBin: CADAusdruckGroesser
!

einfuegenGroesserG

self einfuegenBin: CADAusdruckGroesserGleich
!

einfuegenKleiner

self einfuegenBin: CADAusdruckKleiner
!

einfuegenKleinerG

self einfuegenBin: CADAusdruckKleinerGleich
!

einfuegenKonst: aClass

|konst|
(aClass = CADWertBetrag)
        ifTrue:
                [konst := CADWertEditor openWithWaehrungAuswahlOn: false class:
aClass]
        ifFalse:
                [konst := CADWertEditor openOn: false class: aClass].
konst notNil
        ifTrue:
                [self ausdruckListe add: konst.
                (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe].
!

einfuegenMinus

self einfuegenBin: CADAusdruckMinus

!

einfuegenMoney

self einfuegenKonst: CADWertBetrag
!

einfuegenMul

self einfuegenBin: CADAusdruckMult
!
```

```
einfuegenNot

self einfuegenUn: CADAusdruckNot
!

einfuegenNumeric

self einfuegenKonst: CADWertNumerisch
!

einfuegenOr

self einfuegenBin: CADAusdruckOr
!

einfuegenPlus

self einfuegenBin: CADAusdruckPlus
!

einfuegenProzent

self einfuegenKonst: CADWertProzent
!

einfuegenSelektion: aSymbol

|ausdr paramList anAusdruckZugriff|
paramList := (self subpartNamed: 'lstAusdruecke') selectedItems.
(paramList size = 1)
        ifTrue:
                [(paramList first isBoolean and: [ paramList first isZugriffEindeutig])
                        ifTrue:
                                [anAusdruckZugriff := paramList first detectAusdruckZugriff.
                                anAusdruckZugriff notNil
                                        ifTrue:
                                                [ausdr := CADAusdruckSelektion new.
                                                ausdr owner: self version versioniertesObjekt;
                                                                ownerTyp: self version
versioniertesObjekt class typ.
                                                ausdr bedingung: (paramList first
deepCopyZurOwner: ausdr).
                                                ausdr treffer: aSymbol.
                                                ausdr zugriffTyp: anAusdruckZugriff zugriffTyp.
                                                self ausdruckListe add: ausdr.
                                                (self subpartNamed: 'lstAusdruecke') items:
self ausdruckListe]
                                        ifFalse:
                                                [CADInfoPrompter promptFor: 'Falsche
Definition']]
                        ifFalse:
                                [CADInfoPrompter promptFor: 'Falsche Definition']
                ]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen ein Parameter selektieren!!']!

einfuegenSelektionAlle
```

```
self einfuegenSelektion: #Alle
!

einfuegenSelektionAlleAkt

self einfuegenSelektion: #AlleA
!

einfuegenSelektionErsterTreffer

self einfuegenSelektion: #Erster!

einfuegenSelektionErsterTrefferAkt

self einfuegenSelektion: #ErsterA!

einfuegenSelektionLetzterTreffer

self einfuegenSelektion: #Letzter!

einfuegenSelektionLetzterTrefferAkt

self einfuegenSelektion: #LetzterA!

einfuegenUn: aClass

|ausdr paramList|
paramList := (self subpartNamed: 'lstAusdruecke') selectedItems.
(paramList size = 1)
        ifTrue:
                [ausdr := aClass new.
                ausdr owner: self version versioniertesObjekt;
                                ownerTyp: self version versioniertesObjekt class typ.
                ausdr erstesAusdruck: (paramList first deepCopyZurOwner: ausdr).
                ausdr zweitesAusdruck: nil.
                ausdr isPlausi
                        ifTrue:
                                [self ausdruckListe add: ausdr.
                                (self subpartNamed: 'lstAusdruecke') items: self
ausdruckListe]
                        ifFalse:
                                [CADInfoPrompter promptFor: 'Falsche Definition']
                ]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen ein Parameter selektieren!!']


!

einfuegenUngleich

self einfuegenBin: CADAusdruckUngleich
!

einfuegenWert

|var konst|
```

```
var :=  (self subpartNamed: 'Container Details1') selectedItem object.
konst := var  wertFuer: self version.
(konst isDataAvailable and: [(konst isKindOf: CADWert) not])
        ifTrue: [
                "Damit von VersicherterGefahr und VersichertemObjekt kein deepCopy
ausgeführt wird. Der Vergleich zwischen dem Attribut Vater und dem Wert
                stimmt sonst nicht. Die Kopie wird dann mit == mit dem Original
verglichen"
                (konst isKindOf: CADStrukturVUTEObjekt)
                ifTrue: [konst := var variableTyp new: konst]
                ifFalse: [konst := var variableTyp new: konst deepCopy]
                ].
konst isDataAvailable
        ifTrue:
                [self ausdruckListe add: konst.
                (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe].
!

einfuegenZugriff

self isZielVariable
        ifTrue: [self zielVariableUebernehmen]
        ifFalse: [self einfuegenZugriffAusdruck]!

einfuegenZugriffAusdruck

|ausdr aCollection|
aCollection :=  (self subpartNamed: 'Container Details1') selectedItems.
(aCollection notEmpty)
        ifTrue:
                [aCollection do: [:each|
                        ausdr := each object getAusdruckZugriff.
                        ausdr owner: self version versioniertesObjekt;
                                        ownerTyp: self version versioniertesObjekt class typ.
                        self ausdruckListe add: ausdr.
                        (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe].
                ]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen mindestens einen Parameter
selektieren!!']!

einfuegenZugriffObjekt

        |ausdr obj |
        obj := (self subpartNamed: 'Tree View1') selectedItem object.
        (obj isZugriffObjekt and: [self isZielVariable not])
                ifTrue:
                        [ausdr := CADAusdruckZugriff new.
                        ausdr           symbol: String new;
                                                symbolTyp: 'Objekt';
                                                zugriffTyp: obj class typ;
                                                zugriffKey: obj stammobjekt kurztext;
                                                merker: obj merker.
                        ausdr owner: self version versioniertesObjekt;
                                        ownerTyp: self version versioniertesObjekt class typ.
                        self ausdruckListe add: ausdr.
                        (self subpartNamed: 'lstAusdruecke') items: self ausdruckListe]
                ifFalse:
```

```
                    [CADInfoPrompter promptFor: 'Übernahme nicht erlaubt!!']
!

enableBtnOK

(self subpartNamed: 'lstAusdruecke') selectedItems notEmpty
        ifTrue: [(self subpartNamed: 'btnOK') enable]
        ifFalse: [(self subpartNamed: 'btnOK') disable]
!

enableLeftMenu: anObject

self leftClass notNil
        ifTrue:
                [(anObject isKindOf: self leftClass)
                        ifTrue:
                                [(self subpartNamed: 'MenuZugriffUebernehmen') enable]
                        ifFalse:
                                [(self subpartNamed: 'MenuZugriffUebernehmen') disable]
                ]
!

getSelectedItems
        "Returne eine Collection der im ContainerDetails selektierten Variablen"

        ^(self view subpartNamed: 'Container Details1') selectedItems collect:
                        [:eachItem | eachItem object variable].
!

initialize

super initialize.
self ausdruckListe: OrderedCollection new.!

initializeKonstMenus

(self subpartNamed: 'MenuAlphanumeric')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenAlphanumeric;
                        receiver: self).
(self subpartNamed: 'MenuNumeric')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenNumeric;
                        receiver: self).
(self subpartNamed: 'MenuBoolean')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenBoolean;
                        receiver: self).
(self subpartNamed: 'MenuMoney')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenMoney;
                        receiver: self).
(self subpartNamed: 'MenuProzent')
                abtWhen: #clicked
```

```
                perform: (DirectedMessage new
                        selector: #einfuegenProzent;
                        receiver: self).
(self subpartNamed: 'MenuDate')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenDate;
                        receiver: self).
(self subpartNamed: 'MenuGefuellt')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenGefuellt;
                        receiver: self).
(self subpartNamed: 'MenuAlle')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionAlle;
                        receiver: self).
(self subpartNamed: 'MenuErsterTreffer')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionErsterTreffer;
                        receiver: self).
(self subpartNamed: 'MenuLetzterTreffer')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionLetzterTreffer;
                        receiver: self).
(self subpartNamed: 'MenuAlleAkt')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionAlleAkt;
                        receiver: self).
(self subpartNamed: 'MenuErsterTrefferAkt')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionErsterTrefferAkt;
                        receiver: self).
(self subpartNamed: 'MenuLetzterTrefferAkt')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenSelektionLetzterTrefferAkt;
                        receiver: self).!

initializeOperatorButtons

(self subpartNamed: 'btnPlus')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenPlus;
                        receiver: self).
(self subpartNamed: 'btnMinus')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenMinus;
                        receiver: self).
(self subpartNamed: 'btnMul')
                abtWhen: #clicked
```

```
                perform: (DirectedMessage new
                        selector: #einfuegenMul;
                        receiver: self).
(self subpartNamed: 'btnDiv')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenDiv;
                        receiver: self).
(self subpartNamed: 'btnGleich')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenGleich;
                        receiver: self).
(self subpartNamed: 'btnUngleich')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenUngleich;
                        receiver: self).
(self subpartNamed: 'btnKleiner')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenKleiner;
                        receiver: self).
(self subpartNamed: 'btnGroesser')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenGroesser;
                        receiver: self).
(self subpartNamed: 'btnKleinerG')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenKleinerG;
                        receiver: self).
(self subpartNamed: 'btnGroesserG')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenGroesserG;
                        receiver: self).
(self subpartNamed: 'btnGefuellt')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenGefuellt;
                        receiver: self).
(self subpartNamed: 'btnNot')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenNot;
                        receiver: self).
(self subpartNamed: 'btnAnd')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenAnd;
                        receiver: self).
(self subpartNamed: 'btnOr')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenOr;
                        receiver: self).!
```

```
initializeParts

|aString|
        aString := (self subpartNamed: 'Window1') title.
        aString := aString, ' ', self version versioniertesObjekt langtext.
        (self subpartNamed: 'Window1') title: aString.
        self showAusdruckListe.

        (self subpartNamed: 'Window1')
                abtWhen: #openedWidget
                perform: (DirectedMessage new
                        selector: #displayTree;
                        receiver: self).

        (self subpartNamed: 'btnOK')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #okButtonClicked;
                        receiver: self).

        (self subpartNamed: 'Tree View1')
                abtWhen: #selectedItem
                perform: (DirectedMessage new
                        selector: #selectedTreeViewItemChanged;
                        receiver: self).

        (self subpartNamed: 'MenuUebernehmen')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenZugriff;
                        receiver: self).
        (self subpartNamed: 'MenuWertUebernehmen')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenWert;
                        receiver: self).
        (self subpartNamed: 'MenuZugriffUebernehmen')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #einfuegenZugriffObjekt;
                        receiver: self).

        self initializeOperatorButtons.
        self initializeKonstMenus.

        (self subpartNamed: 'btnCancel')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #abbruchButtonClicked;
                        receiver: self).

(self subpartNamed: 'MenuAuswerten')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #ausdruckAuswerten;
                        receiver: self).
```

```
(self subpartNamed: 'MenuLoeschen')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #ausdruckLoeschen;
                        receiver: self).

(self subpartNamed: 'lstAusdruecke')
                abtWhen: #selectedItems
                perform: (DirectedMessage new
                        selector: #enableBtnOK;
                        receiver: self).

(self subpartNamed: 'MenuAuswertungListe')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #showAuswertungListe;
                        receiver: self).

(self subpartNamed: 'MenuVarRef')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #varReferenzen;
                        receiver: self).

(self subpartNamed: 'MenuObjRef')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #objektReferenzen;
                        receiver: self).

!

installContainerDetailsController

        self containerDetailsController: (CADOldContainerDetailsController

                                                                        newFor:
(self  subpartNamed: 'Container Details1')

                                                                        getData:
[#()]

                                                                        getLabel:
self cdvLabelBlock).
!

installTreeViewController

|aCollection|
aCollection := OrderedCollection new.
aCollection add: self version versioniertesObjekt.
                self treeViewController: (CADOldTreeViewController

        newFor:         (self  subpartNamed: 'Tree View1')

        rootObjects:            aCollection

        getNext:                                [:aVersionObject1 |
```

```
                                                                aVersionObject1
allBrowsedRoots: self version]

        getImage:                               [:aVersionObject2 | CADDllResourceManager
new getBitmapDescriptor: aVersionObject2 imageName. ]

        getString:                      [:aVersionObject3 | aVersionObject3
treeNodeItemLabel]).!

isAktivierbaresObjekt

^self plausi = #AktivierbaresObjekt!

isBoolean

^self plausi = #Boolean!

isVerknuepfung

^self plausi = #Verknuepfung!

isZielVariable

^self plausi = #ZielVariable!

leftClass

^leftClass!

leftClass: aClass

leftClass := aClass!

objektReferenzen

| param aCollection|
param := (self subpartNamed: 'Tree View1') selectedItem object.
param isDataAvailable
        ifTrue:
                [aCollection := self version versioniertesObjekt findReferencesFor: param
version: self version.
                aCollection notEmpty
                        ifTrue:
                                [^CADReferenzView openOn: aCollection title: param
attrAnzeigenMitArt]]




!

okButtonClicked

|anAusdruck aClass|
        self closeTreeViewController.
        self closeContainerDetailsController.
```

```
	(self subpartNamed: 'lstAusdruecke') selectedItems notEmpty
		ifTrue:
			[anAusdruck := (self subpartNamed: 'lstAusdruecke') selectedItems
first.
			self result: anAusdruck.
			(anAusdruck notNil and: [self isAktivierbaresObjekt])
				ifTrue:
					[((anAusdruck isZugriff or: [anAusdruck
isZugriffObjekt]) and: [(aClass := anAusdruck class getClassViaTyp: anAusdruck
zugriffTyp) isDataAvailable and: [aClass isAktivierbaresObjekt]])
						ifFalse:
							[self result: nil.
							^CADInfoPrompter promptFor: 'Kein
aktivierbares Objekt!!']]].
			(anAusdruck notNil and: [self isBoolean])
				ifTrue:
					[anAusdruck isBoolean
						ifFalse:
							[self result: nil.
							^CADInfoPrompter promptFor: 'Kein
logischer Ausdruck!!']].
	(self subpartNamed: 'Window1') closeWidgetCommand.!

plausi

^plausi!

plausi: aSymbol

plausi := aSymbol!

result
	"Return the value of result."

	^result!

result: anObject
	"Save the value of result."

	result := anObject.
	self signalEvent: #result
		with: anObject. !

selectedTreeViewItemChanged
	"Finde das (im TreeView) selektierte Item und bringe seine Kinder im Container
Details zur Anzeige"

	| anItem |

	anItem := (self  subpartNamed: 'Tree View1') selectedItem.
	anItem notNil
		ifTrue:
			[self containerDetailsController clear.
			 self containerDetailsController newData: ([ | aCollection|

						aCollection := OrderedCollection new.
```

```
isDataAvailable not) or:                                ((anItem object variableListeBasis
                                                        [anItem object variableListeBasis isEmpty])
                                                                ifTrue: [anItem object
createVariableListeBasis].
                                                                aCollection addAll: anItem object
variableListeBasis.
                                                                aCollection addAll: anItem object
variableListe.
                                                                aCollection asSortedCollection:
                                                                        [:a :b| a symbol <= b symbol].
                                                                ]).
                                self containerDetailsController display.
                                self enableLeftMenu: anItem object.
                        ]
                ifFalse:
                        [ self containerDetailsController clear].

!

showAusdruckListe

(self subpartNamed: 'lstAusdruecke') items: self ausdruckListe!

showAuswertungListe

|anItem|
anItem := (self subpartNamed: 'Container Details1') selectedItem.
anItem notNil
        ifTrue:
                [(anItem object isKindOf: CADVariable)
                        ifTrue:
                                [CADVariableAuswertungListeView openOn:
                                        ((self subpartNamed: 'Container Details1')
selectedItem object)]
                        ifFalse:
                                [CADInfoPrompter promptFor: 'Der Wert wurde intern
gesetzt']
                ]
        ifFalse:
                [CADInfoPrompter promptFor: 'Sie müssen einen Parameter selektieren!!']
!

treeViewController

        ^treeViewController!

treeViewController: aController

        treeViewController := aController!
```

```
varReferenzen

| param aCollection|
param := (self subpartNamed: 'Container Details1') selectedItem object.
param isDataAvailable
        ifTrue:
                [aCollection := self version versioniertesObjekt findReferencesForVariable:
param version: self version.
                aCollection notEmpty
                        ifTrue:
                                [^CADReferenzView openOn: aCollection title: param
attrAnzeigenMitArt]]



!

version

^version!

version: aVersion

version := aVersion!

zielVariableUebernehmen

|ausdr|
ausdr := (self subpartNamed: 'Container Details1') selectedItem object.
(ausdr isKindOf: CADVariable)
        ifTrue: [self einfuegenZugriffAusdruck]
        ifFalse: [CADInfoPrompter promptFor: 'Übernahme nicht erlaubt!!']!

zugriffReferenzen

| param aCollection|
param := (self subpartNamed: 'lstAusdruecke') selectedItem.
param isDataAvailable
        ifTrue:
                [aCollection := self version versioniertesObjekt findReferencesFor: param
version: self version.
                aCollection notEmpty
                        ifTrue:
                                [^CADReferenzView openOn: aCollection title: param
attrAnzeigenMitArt]]




! !

!CADAusdruckeditorView privateMethods !
.....


!CADWertEditor class publicMethods !
```

```
open

	| aView |
	aView := self new.
	aView initializeParts.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	^aView result!

openOn: aBoolean

	| aView |
	aView := self new.
	aView initializeParts.
	self anzeigemodus: aBoolean.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	^aView result!

openOn: aBoolean class: aClass

	| aView |
	aView := self new.
	aView anzeigemodus: aBoolean.
	aView destinationClass: aClass.
	aView perform: (aView converterDictionary at: aClass).
	aView initializeParts.
	(aView subpartNamed: 'Window') title: (aView subpartNamed: 'Window') title, ' :
', aView destinationClass text.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	^aView result!

openOn: aBoolean class: aClass waehrung: aWaehrung

	| aView |
	aView := self new.
	aView anzeigemodus: aBoolean.
	aView destinationClass: aClass.
	aView waehrung: aWaehrung.
	aView perform: (aView converterDictionary at: aClass).
	aView initializeParts.
	(aView subpartNamed: 'Window') title: (aView subpartNamed: 'Window') title, ' :
', aView destinationClass text.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	^aView result!

openOn: aBoolean form: aForm

	| aView |
	aView := self new.
	aView initializeParts.
	(aView subpartNamed: 'Group Box') subpartNamed: 'FormWert'  put: aForm.
	self anzeigemodus: aBoolean.
	aView openApplicationModalWidget.
	aView suspendExecutionUntilRemoved.
	^aView result!
```

```
openWithWaehrungAuswahlOn: aBoolean class: aClass

        | aView |
        aView := self new.
        aView anzeigemodus: aBoolean.
        aView destinationClass: aClass.
        aView perform: (aView converterDictionary at: aClass).
        aView enableWaehrung.
        aView initializeParts.
        (aView subpartNamed: 'Window') title: (aView subpartNamed: 'Window') title, ' :
', aView destinationClass text.
        aView openApplicationModalWidget.
        aView suspendExecutionUntilRemoved.
        ^aView result! !

!CADWertEditor class privateMethods !
.....

!CADWertEditor publicMethods !

abbruchButtonClicked

        (self subpartNamed: 'Window') closeWidgetCommand.!

anzeigemodus

^anzeigemodus!

anzeigemodus: aBoolean

anzeigemodus := aBoolean!

converterDictionary

|aDictionary|
aDictionary := Dictionary new.
aDictionary at: CADWertAlphanumerisch put: #setConverterString.
aDictionary at: CADWertBoolean put: #setConverterBoolean.
aDictionary at: CADWertBetrag put: #setConverterDecimal.
aDictionary at: CADWertProzent put: #setConverterFloat.
aDictionary at: CADWertNumerisch put: #setConverterInteger.
aDictionary at: CADWertDatum put: #setConverterDate.

^aDictionary!

destinationClass

^destinationClass!

destinationClass: aClass

destinationClass := aClass!

editierenErlauben: aBoolean

(self subpartNamed: 'wert') editable: aBoolean not!
```

```
enableWaehrung

(self subpartNamed: 'drpWaehrung') enable.
!

initializeParts

self initializeWaehrung.
self editierenErlauben: self anzeigemodus.

        (self subpartNamed: 'Window')
                abtWhen: #openedWidget
                perform: (DirectedMessage new
                        selector: #setFocus;
                        receiver: (self subpartNamed: 'wert') ).
        (self subpartNamed: 'drpWaehrung')
                abtWhen: #selectedItem
                perform: (DirectedMessage new
                        selector: #waehrung:;
                        receiver: self;
                        arguments: (Array new:1) ).
        (self subpartNamed: 'btnOK')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #okButtonClicked;
                        receiver: self).
        (self subpartNamed: 'btnCancel')
                abtWhen: #clicked
                perform: (DirectedMessage new
                        selector: #abbruchButtonClicked;
                        receiver: self).
!

initializeWaehrung

(self destinationClass = CADWertBetrag)
        ifTrue:
                [(self subpartNamed: 'drpWaehrung') items: self selectWaehrung.
                self waehrung notNil
                        ifTrue:
                                [(self subpartNamed: 'drpWaehrung') selectedItem: self
waehrung]
                ].!

okButtonClicked

|obj|
self anzeigemodus
        ifFalse:
                [obj :=  (self subpartNamed: 'wert') object.
                self destinationClass = CADWertBetrag
                        ifTrue:
                                [obj := self destinationClass wertTyp new: obj deepCopy
waehrung: self waehrung]
                        ifFalse:
                                [(obj isKindOf: self destinationClass wertTyp)
                                        ifFalse:
```

```
                                                [obj := self destinationClass wertTyp new: obj
deepCopy]
                                    ].
                        self result: (self destinationClass new: obj)].
(self subpartNamed: 'Window') closeWidgetCommand.!

result

^result!

result: anObject

result := anObject!

selectWaehrung

|aCollection|
aCollection := (KEYWaehrungenManagerAusTableSystem new alleWaehrungen )
        asSortedCollection: [:a :b| a beschriftungKurz < b beschriftungKurz].
^aCollection
        !

setConverter: aConverterClass

(self subpartNamed: 'wert')
                    converter: ( aConverterClass abtCreatePart: aConverterClass name
parent:  nil )!

setConverterBoolean

self setConverter:  AbtBooleanConverter.
(self subpartNamed: 'wert')  converter noStr: 'Nein' ;
                                                yesStr: 'Ja'.
(self subpartNamed: 'wert') object: true.!

setConverterDate

self setConverter:  AbtDateConverter.
(self subpartNamed: 'wert') object: Date today!

setConverterDecimal

self setConverter:  AbtDecimalConverter.
!

setConverterFloat

self setConverter: AbtFloatingPointRepresentationConverter .

!

setConverterInteger

self setConverter: AbtIntegerConverter.
!

setConverterString
```

```
self setConverter: AbtStringConverter.
!

waehrung

waehrung isNil
        ifTrue:
                [waehrung := KEYWaehrungenManagerAusTableSystem new
defaultWaehrung].
^waehrung!

waehrung: anObject

waehrung := anObject! !

!CADWertEditor privateMethods !
.....

!CADWertView class privateMethods !
.....

!CADWertView privateMethods !
.....
```

B Fragmente de program (calculul lunetei Kepler)

Codul programului VisualBasic înscris în modulul proiectului conţine secvenţele necesare proiectării lunetei, urmărindu-se etapele de constructie aşa cum au fost prezentate în paragrafele anterioare. Aplicatiile a fost realizata in colaborare cu Facultatea de Mecanica din Universitatea „Politehnica" Timisoara.

Se prezintă, in continuare, secvenţele principale ale codului.

```
Private Sub Text1_GotFocus()
   Text1.SelStart = 0
   Text1.SelLength = 10
End Sub
Private Sub Text2_GotFocus()
   Text2.SelStart = 0
   Text2.SelLength = 10
End Sub
Private Sub Text3_GotFocus()
   Text3.SelStart = 0
   Text3.SelLength = 10
End Sub
Private Sub Text4_GotFocus()
   Text4.SelStart = 0
   Text4.SelLength = 10
End Sub
Private Sub Text5_GotFocus()
   Text5.SelStart = 0
   Text5.SelLength = 10
End Sub
Private Sub Command1_Click()
ProgressBar1.Min = 0
ProgressBar1.Max = 15
ProgressBar1.Value = 0
Dim dbBazaDate As Database
Dim rs As Recordset
Dim Gama As Double
Dim doi_omega As Double
Dim omega As Double
Dim D_prim As Double
Dim s_prim_p_prim As Double
Dim L As Variant

Let PI = Atn(1) * 4

   ' Citeste datele de la intrare
' Citeste grosismentul
Let Gama = Text1
' Citeste unghiul de camp obiect
Let doi_omega = Text2
' Calculeaza pe omega
Let omega# = (doi_omega / 2) * PI / 180
' Citeste diametrul pupilei de iesire
```

```
Let D_prim = Text3
' Citeste departarea pupilei de iesire
Let s_prim_p_prim# = Text4
' Citeste lungimea tubului
Let L = Text5
'-----------------------------------------------------------------
' 1) Distanta focala a obiectivului
ProgressBar1.Value = 1
Dim f_prim_ob As Double
   Let f_prim_ob = Gama / (Gama + 1) * L
   Let Text6 = f_prim_ob
'-----------------------------------------------------------------
' 2) Distanta focala a ocularului
ProgressBar1.Value = 2
Dim f_prim_oc As Double
f_prim_oc = 1 / (Gama + 1) * L
Text7 = f_prim_oc
'-----------------------------------------------------------------
' 3) Unghiul de camp imagine
ProgressBar1.Value = 3
Dim doi_omega_prim As Double
doi_omega_prim = Atn(Gama * Tan(omega)) * 2 / PI * 180
Text8 = doi_omega_prim
'-----------------------------------------------------------------
' 4) Se alege ocularul din baza de date cu conditiile
'   - f_prim_oc ales are valoarea cea mai apropiata de f_prim_oc calculat
'   - doi_omega_prim pentru ocularul ales este >= doi_omega_prim calculat
'   - s_prim_p_prim pentru ocularul ales este >= s_prim_p_prim impus.
ProgressBar1.Value = 4
Dim f_prim_oc_ales As Variant
Dim s_prim_f_prim_ocular_ales As Variant
  ' Diferenta cea mai buna
Dim BestDif As Integer
  ' Cel mai apropiat numar mai mare decat valoarea cautata
Dim ClosestBigger As Integer
  ' Cel mai apropiat numar mai mic decat valoarea cautata
Dim ClosestSmaller As Integer
  ' De cate ori apare cea mai apropiata valoare mai mare
Dim BiggerCount As Integer
  ' De cate ori apare cea mai apropiata valoare mai mica
Dim SmallerCount As Integer
Set dbBazaDate = OpenDatabase("Proiect.Mdb")
SQLQuery = "SELECT * FROM Oculare"
Text39 = SQLQuery
Set rs = dbBazaDate.OpenRecordset(SQLQuery)
Do Until rs.EOF = True
   rs.MoveNext
Loop
Let Text41 = Text7
'  Initializeaza listele
List1.Clear
List2.Clear
Let BiggerCount = 0
Let SmallerCount = 0
Let ClosestBigger = 0
Let ClosestSmaller = 0
Let ClosestCount = 0
Let BestDif = 1000
```

```
' cauta cele mai apropiate valori mai mici
rs.MoveFirst
For poz = 1 To rs.RecordCount
  If (Text41 - rs.Fields("f_prim_oc")) < BestDif _
  And (Text41 - rs.Fields("f_prim_oc")) > 0 Then
    ' o noua valoare mai mica mai apropiata de numarul cautat
    ClosestSmaller = rs.Fields("f_prim_oc")
    SmallerCount = 1
    BestDif = Text41 - rs.Fields("f_prim_oc")
    List1.AddItem rs.Fields("f_prim_oc")
    List2.AddItem rs.Fields("ID")
  ElseIf Text41 - rs.Fields("f_prim_oc") = BestDif Then
    ' inca o valoare mai mica
    SmallerCount = SmallerCount + 1
    List1.AddItem rs.Fields("f_prim_oc")
    List2.AddItem rs.Fields("ID")
  End If
  rs.MoveNext
Next poz
' cauta cele mai apropiate valori mai mari
rs.MoveFirst
For poz = 1 To rs.RecordCount
  If rs.Fields("f_prim_oc") - Text41 < BestDif _
  And rs.Fields("f_prim_oc") - Text41 >= 0 Then
    ' o noua valoare mai mare mai apropiata de numarul cautat
    ClosestBigger = rs.Fields("f_prim_oc")
    BiggerCount = 1
    BestDif = rs.Fields("f_prim_oc") - Text41
    List1.AddItem rs.Fields("f_prim_oc")
    List2.AddItem rs.Fields("ID")
  ElseIf rs.Fields("f_prim_oc") - Text41 = BestDif Then
    ' inca o valoare mai mare
    ClosestBigger = rs.Fields("f_prim_oc")
    BiggerCount = BiggerCount + 1
    List1.AddItem rs.Fields("f_prim_oc")
    List2.AddItem rs.Fields("ID")
  End If
  rs.MoveNext
Next poz
' check to see what really is the closest number
If (Text41 - ClosestSmaller) = (ClosestBigger - Text41) Then ' bigger and smaller bests
are equally close
  Let bestchoice = ClosestBigger
ElseIf BiggerCount = 0 Then ' smaller was closest
  Let bestchoice = ClosestSmaller
ElseIf SmallerCount = 0 Then ' bigger was closest
  Let bestchoice = ClosestBigger
ElseIf (Text41 - ClosestSmaller) < (ClosestBigger - Text41) Then ' smaller was closest
  Let bestchoice = ClosestSmaller
ElseIf (Text41 - ClosestSmaller) > (ClosestBigger - Text41) Then ' bigger was closest
  Let bestchoice = ClosestBigger
End If
' elimina surplusurile din liste
Let elimina = 0
Do While elimina < List1.ListCount
    If List1.List(elimina) - BestDif <> bestchoice - BestDif _
    And List1.List(elimina) + BestDif <> bestchoice - BestDif _
```

```
   Then List1.RemoveItem (elimina): List2.RemoveItem (elimina): Let elimina = elimina
- 1
   Let elimina = elimina + 1
Loop

' elimina cazurile in care doi_omega_prim ales < doi_omega_prim calculat
List1.Clear
List2.Clear
Let SQLQuery = "SELECT * FROM Oculare WHERE f_prim_oc= " & bestchoice
Text39 = SQLQuery
Set rs = dbBazaDate.OpenRecordset(SQLQuery)
Do Until rs.EOF = True

   List1.AddItem rs.Fields("f_prim_oc"): List2.AddItem rs.Fields("ID")
   ' populeaza formularul
   Text20 = rs.Fields("d1")
   Text23 = rs.Fields("d2")
   Text21 = rs.Fields("d3")
   Text22 = rs.Fields("d4")
   If rs.Fields("d5") <> Null Then
      Text25 = rs.Fields("d5")
   Else: Text25.Enabled = False
   End If
   If rs.Fields("d6") <> Null Then
      Text26 = rs.Fields("d6")
   Else: Text26.Enabled = False
   End If
   If rs.Fields("d7") <> Null Then
      Text24 = rs.Fields("d7")
   Else: Text24.Enabled = False
   End If
   Text35 = rs.Fields("f_prim_oc")
   Text38 = rs.Fields("s_prim_f_prim")
   Text36 = rs.Fields("doi_omega_prim")
   Text37 = rs.Fields("s_prim_p_prim")
   Text34 = rs.Fields("r1")
   Text31 = rs.Fields("r2")
   Text33 = rs.Fields("r3")
   Text32 = rs.Fields("r4")
   Text28 = rs.Fields("r5")
   If rs.Fields("r6") <> Null Then
      Text27 = rs.Fields("r6")
   Else: Text27.Enabled = False
   End If
   Text45 = rs.Fields("d_oc")
rs.MoveNext
Loop
f_prim_oc_ales = List1.List(0)
Text43 = List2.List(0)
'-----------------------------------------------------------------
' 5) Se recalculeaza distanta focala obiectiv
ProgressBar1.Value = 5
Let f_prim_ob = Gama * f_prim_oc_ales
Let Text9 = f_prim_ob
'-----------------------------------------------------------------
' 6) Se alege obiectivul din baza de date astfel incat sa fie cat mai apropiat de cel calculat
=> f_prim_ob ales
```

```
ProgressBar1.Value = 6
Dim f_prim_ob_ales As Double
SQLQuery = "SELECT * FROM Obiective"
Text44 = SQLQuery
Set rs = dbBazaDate.OpenRecordset(SQLQuery)
Do Until rs.EOF = True
rs.MoveNext
Loop
Text42 = f_prim_ob
List3.Clear
List4.Clear
BiggerCount = 0
SmallerCount = 0
ClosestBigger = 0
ClosestSmaller = 0
ClosestCount = 0
BestDif = 1000
' cauta cele mai apropiate valori mai mici
rs.MoveFirst
For poz = 1 To rs.RecordCount
  If (Text42 - rs.Fields("f_prim_ob")) < BestDif _
  And (Text42 - rs.Fields("f_prim_ob")) > 0 Then ' new best smaller number
    ClosestSmaller = rs.Fields("f_prim_ob")
    SmallerCount = 1
    BestDif = Text42 - rs.Fields("f_prim_ob")
    List3.AddItem rs.Fields("f_prim_ob")
    List4.AddItem rs.Fields("ID")
  ElseIf Text42 - rs.Fields("f_prim_ob") = BestDif Then ' another best smaller num
    SmallerCount = SmallerCount + 1
    List3.AddItem rs.Fields("f_prim_ob")
    List4.AddItem rs.Fields("ID")
  End If
  rs.MoveNext
Next poz
' cauta cele mai apropiate valori mai mari
rs.MoveFirst
For poz = 1 To rs.RecordCount
  If rs.Fields("f_prim_ob") - Text42 < BestDif _
  And rs.Fields("f_prim_ob") - Text42 >= 0 Then ' new best bigger number
    ClosestBigger = rs.Fields("f_prim_ob")
    BiggerCount = 1
    BestDif = rs.Fields("f_prim_ob") - Text42
    List3.AddItem rs.Fields("f_prim_ob")
    List4.AddItem rs.Fields("ID")
  ElseIf rs.Fields("f_prim_ob") - Text42 = BestDif Then ' another best bigger num
    ClosestBigger = rs.Fields("f_prim_ob")
    BiggerCount = BiggerCount + 1
    List3.AddItem rs.Fields("f_prim_ob")
    List4.AddItem rs.Fields("ID")
  End If
  rs.MoveNext
Next poz
' check to see what really is the closest number
If (Text42 - ClosestSmaller) = (ClosestBigger - Text42) Then ' bigger and smaller bests
are equally close
  Let bestchoice = ClosestBigger
ElseIf BiggerCount = 0 Then ' smaller was closest
  Let bestchoice = ClosestSmaller
```

```
ElseIf SmallerCount = 0 Then ' bigger was closest
  Let bestchoice = ClosestBigger
ElseIf (Text42 - ClosestSmaller) < (ClosestBigger - Text42) Then ' smaller was closest
  Let bestchoice = ClosestSmaller
ElseIf (Text42 - ClosestSmaller) > (ClosestBigger - Text42) Then ' bigger was closest
  Let bestchoice = ClosestBigger
End If
' elimina surplusurile din liste
Let elimina = 0
Do While elimina < List3.ListCount
If List3.List(elimina) - BestDif <> bestchoice - BestDif _
And List3.List(elimina) + BestDif <> bestchoice - BestDif _
Then List3.RemoveItem (elimina): List4.RemoveItem (elimina): Let elimina = elimina - 1
Let elimina = elimina + 1
Loop
Let f_prim_ob_ales = List3.List(0)
Text43 = f_prim_ob_ales
SQLQuery = "SELECT * FROM Obiective WHERE f_prim_ob= " & f_prim_ob_ales
Text44 = SQLQuery
Set rs = dbBazaDate.OpenRecordset(SQLQuery)
   Text19 = rs.Fields("d1")
   Text18 = rs.Fields("d2")
   Text14 = rs.Fields("f_prim_ob")
   Text11 = rs.Fields("s_prim_f_prim")
   Text13 = rs.Fields("t")
   Text12 = rs.Fields("D")
   Text15 = rs.Fields("r1")
   Text17 = rs.Fields("r2")
   Text16 = rs.Fields("r3")
'-----------------------------------------------------------------
' 7) Se recalculeaza grosismentul real
ProgressBar1.Value = 7
Dim Gama_real As Double
Gama_real = f_prim_ob_ales / f_prim_oc_ales
Text10 = Gama_real
'-----------------------------------------------------------------
' 8) Se recalculeaza lungimea reala a tubului
ProgressBar1.Value = 8
Dim L_real As Double
Let L_real = f_prim_oc_ales + f_prim_ob_ales
Text50 = L_real
'-----------------------------------------------------------------
' 9) Diametrul pupilei de intrare
ProgressBar1.Value = 9
Dim d As Variant
Let d = Gama_real * D_prim
Text51 = d
'-----------------------------------------------------------------
' 10) Departarea pupilei de iesire
ProgressBar1.Value = 10
Let s_prim_f_prim_oc_ales = Text4
Let s_prim_p_prim = s_prim_f_prim_oc_ales + f_prim_ob_ales / (Gama_real ^ 2)
Text52 = s_prim_p_prim
'-----------------------------------------------------------------
' 11) Diametrul diafragmei de camp
ProgressBar1.Value = 11
Dim D_dc As Variant
Let D_dc = 2 * f_prim_ob_ales * Tan(omega)
```

```
Text53 = D_dc
'-----------------------------------------------------------------
' 12) Unghiul razei marginale cu axa optica
ProgressBar1.Value = 12
Dim sigma_prim As Variant
Let sigma_prim = Atn(d / (2 * f_prim_ob_ales) + Tan(omega))
Text54 = sigma_prim * 180 / PI
'-----------------------------------------------------------------
' 13) Diametrul util al ocularului
ProgressBar1.Value = 13
Dim d_u_oc As Variant
Let d_u_oc = L_real * Tan(sigma_prim) - d / 2
Text55 = d_u_oc
'-----------------------------------------------------------------
' 14) Se verifica daca D_oc_ales >= d_u_oc
ProgressBar1.Value = 14
   d_oc_ales = Text45
   Text56 = d_oc_ales >= d_u_oc
'-----------------------------------------------------------------
' 15) Se verifica daca D_ob_ales >= D
ProgressBar1.Value = 15
   D_ob_ales = Text12
   Text57 = D_ob_ales >= d
Dim oc(8) As String
Dim sticla(3, 9) As String
Dim h(10) As Variant
cauta "Oculare", "ID", List2.List(0), rs
'Form2.List1.Clear
'Form2.List2.Clear
   Let oc(1) = "aer"
   Let oc(2) = rs.Fields("sticla_oc")
   Let oc(3) = rs.Fields("sticla_oc2")
   Let oc(4) = "aer"
   Let oc(5) = rs.Fields("sticla_oc3")
If List2.List(0) < 6 Then Let n = 6: Let oc(6) = "aer": GoTo skip
   Let oc(6) = rs.Fields("sticla_oc3")
If List2.List(0) < 11 Then Let n = 7:  Let oc(7) = "aer": GoTo skip
   Let oc(6) = "aer"
   Let oc(7) = rs.Fields("sticla_oc4")
   Let oc(8) = rs.Fields("sticla_oc5")
   Let oc(9) = "aer"
   Let n = 9
skip:
'Form2.Show
' calcul trigonometric pt oculare
Dim S1 As Double
Let S1# = 10 ^ 30
Dim r(9) As Variant
Dim ro(9) As Variant
Dim da(7) As Variant
Dim Q(100, 100)
Dim QP(100, 100)
Dim sp(100, 100)
Dim G(100, 100)
Dim SIGP(100, 100)
Dim S(100, 100)
Dim SIG(100, 100)
Dim SINEPS(100, 100)
```

```
Dim SINEPSP(100, 100)
Dim EPS(100, 100)
Dim EPSP(100, 100)
Dim spar(100, 100)
Dim b(100, 100)
Dim spp(100, 100)
Dim u(100, 100)
Dim aber_sfer(10) As Variant
Dim aber_crom_par(10) As Variant
Dim aber_crom(10) As Variant
da(1) = Val(Text20)
da(2) = Val(Text23)
da(3) = Val(Text21)
da(4) = Val(Text22)
da(5) = Val(Text25)
da(6) = Val(Text26)
da(7) = Val(Text24)
r(1) = Val(Text34)
r(2) = Val(Text31)
r(3) = Val(Text33)
r(4) = Val(Text32)
r(5) = Val(Text28)
r(6) = Val(Text27)
r(7) = Val(Text30)
r(8) = Val(Text29)
For i = 1 To n
   cauta_s "sorturi", "sort", oc(i), rs
   'Form2.List1.AddItem rs.Fields("n_e") & "|" & rs.Fields("n_c_prim") & "|" &
rs.Fields("n_f_prim")'
   sticla(1, i) = rs.Fields("n_e")
   sticla(2, i) = rs.Fields("n_c_prim")
   sticla(3, i) = rs.Fields("n_f_prim")
Next i
For repet = 1 To 10
   Let h(repet) = d_u_oc / 2 / 10 * repet
   For j = 1 To n - 1
      If Abs(r(j)) < 10000000000# Then
         Let ro(j) = 1 / r(j)
      Else
         Let ro(j) = 0
      End If
   Next j
   For i = 1 To 3
      If Abs(S1) > 10000000000# Then
         Q(i, 1) = h(repet)
         SIG(i, 1) = 0
      Else
         SIG(i, 1) = Atn(h(repet) / S1)
         Q(i, 1) = S1 * Sin(SIG(1, 1))
      End If

      SINEPS(i, 1) = Sin(SIG(i, 1)) - Q(i, 1) * ro(1)
      EPS(i, 1) = Atn(SINEPS(i, 1) / Sqr(1 - (SINEPS(i, 1)) ^ 2))
      SINEPSP(i, 1) = sticla(i, 1)*(SINEPS(i, 1))/sticla(i, 1 + 1)
      EPSP(i, 1) = Atn(SINEPSP(i, 1)/Sqr(1 - (SINEPSP(i, 1))^ 2))
      SIGP(i, 1) = SIG(i, 1) - EPS(i, 1) + EPSP(i, 1)
      G(i, 1) = Q(i, 1) / (Cos(EPS(i, 1)) + Cos(SIG(i, 1)))
      QP(i, 1) = G(i, 1) * (Cos(EPSP(i, 1)) + Cos(SIGP(i, 1)))
```

```
    Next i
    For i = 1 To 3
       For j = 2 To n - 1
          Q(i, j) = QP(i, j - 1)-(da(j - 1)) * Sin(SIGP(i, j - 1))
          SIG(i, j) = SIGP(i, j - 1)
          S(i, j) = Q(i, j) / Sin(SIG(i, j))
          SINEPS(i, j) = Sin(SIG(i, j)) - Q(i, j) * ro(j)
          EPS(i, j) = Atn(SINEPS(i, j)/Sqr(1 -(SINEPS(i, j)) ^ 2))
          SINEPSP(i,j)=sticla(i, j)*(SINEPS(i, j))/sticla(i,j + 1)
          EPSP(i, j)=Atn(SINEPSP(i, j)/Sqr(1 -(SINEPSP(i, j))^ 2))
          SIGP(i, j) = SIG(i, j) - EPS(i, j) + EPSP(i, j)
          G(i, j) = Q(i, j) / (Cos(EPS(i, j)) + Cos(SIG(i, j)))
          QP(i, j) = G(i, j) * (Cos(EPSP(i, j)) + Cos(SIGP(i, j)))
          sp(i, j) = QP(i, j) / Sin(SIGP(i, j))

       Next j
    Next i
'Form2.Show
' calcul paraxial
     For i = 1 To 3
       spp(i, 1) = sticla(i, 2)/((sticla(i, 2)-sticla(i, 1))/ r(1))
    Next i
    For i = 1 To 3
       For j = 2 To n - 1
          spar(i, j) = spp(i, j - 1) - da(j - 1)
          spp(i,j)=(sticla(i,j+1))/(sticla(i,j)/S(i,j)+(sticla(i,j +1)-sticla(i,j))/r(j))
       Next j
    Next i
' Aberatia sferica
    aber_sfer(repet) = sp(1, n - 1) - spp(1, n - 1) ' pleaca din origine (0,0)
' Aberatia cromatica paraxiala
    aber_crom_par(repet) = spp(3, n - 1) - spp(2, n - 1) ' constant
' Aberatia cromatica
    aber_crom(repet) = sp(3, n - 1) - sp(2, n - 1) 'variaza in functie de h
       Picture1.Cls
    Picture1.BackColor = RGB(0, 0, 0)
    Picture1.ForeColor = RGB(255, 255, 255)
Picture1.AutoRedraw = True
Picture1.CurrentX = 5
Picture1.CurrentY = 30
Picture1.DrawWidth = 1
Picture1.Line -(55, 30)
Picture1.CurrentX = 30
Picture1.CurrentY = 30
Picture1.DrawWidth = 1
Picture1.Line -(30, 5)
Picture1.CurrentX = 30
Picture1.CurrentY = 30
    Picture1.ForeColor = RGB(255, 255, 0)
    Picture1.Line -((30 + aber_sfer(repet)), 30 - 4 * h(repet))
Next repet
Picture1.CurrentX = aber_crom_par(1) + 30
Picture1.CurrentY = 30
Picture1.ForeColor = RGB(255, 0, 0)
For repet = 1 To 10
    Picture1.Line -((30 + aber_crom(repet)), 30 - 4 * h(repet))
Next repet
'Form2.Show
```

```
Dim ob(4) As String
Dim sticla_ob(3, 4) As String
Dim h_ob(10) As Variant
cauta "Obiective", "ID", List4.List(0), rs
'Form2.List1.Clear
'Form2.List2.Clear
   Let ob(1) = "aer"
   Let ob(2) = rs.Fields("sticla_ob")
   Let ob(3) = rs.Fields("sticla_ob2")
   Let ob(4) = "aer"
   Let n = 4
' calcul trigonometric pt obiective
Dim S1_ob As Double
Let S1_ob = 10 ^ 30
Dim r_ob(3) As Variant
Dim ro_ob(3) As Variant
Dim da_ob(2) As Variant
Dim Q_ob(100, 100)
Dim QP_ob(100, 100)
Dim SP_ob(100, 100)
Dim G_ob(100, 100)
Dim SIGP_ob(100, 100)
Dim S_ob(100, 100)
Dim SIG_ob(100, 100)
Dim SINEPS_ob(100, 100)
Dim SINEPSP_ob(100, 100)
Dim EPS_ob(100, 100)
Dim EPSP_ob(100, 100)
Dim a_ob(100, 100)
Dim b_ob(100, 100)
Dim spp_ob(100, 100)
Dim u_ob(100, 100)
Dim aber_sfer_ob(10) As Variant
Dim aber_crom_par_ob(10) As Variant
Dim aber_crom_ob(10) As Variant
da_ob(1) = Val(Text19)
da_ob(2) = Val(Text18)
r_ob(1) = Val(Text15)
r_ob(2) = -1 * Val(Text17)
r_ob(3) = -1 * Val(Text16)
Form2.List1.Clear
For i = 1 To n
   cauta_s "sorturi", "sort", ob(i), rs
   'Form2.List1.AddItem rs.Fields("n_e") & "|" & rs.Fields("n_c_prim") & "|" &
rs.Fields("n_f_prim") '
   sticla_ob(1, i) = rs.Fields("n_e")
   sticla_ob(2, i) = rs.Fields("n_c_prim")
   sticla_ob(3, i) = rs.Fields("n_f_prim")
Next i
For repet = 1 To 10
   Let h_ob(repet) = D_ob_ales / 2 / 10 * repet
   For j = 1 To n - 1
      If Abs(r_ob(j)) < 10000000000# Then
         Let ro_ob(j) = 1 / r_ob(j)
      Else
         Let ro_ob(j) = 0
      End If
   Next j
```

```
   For i = 1 To 3
         Q_ob(i, 1) = h_ob(repet)
         SIG_ob(i, 1) = 0
   SINEPS_ob(i, 1) = Sin(SIG_ob(i, 1)) - Q_ob(i, 1) * ro_ob(1)
   EPS_ob(i, 1)=Atn(SINEPS_ob(i, 1)/Sqr(1-(SINEPS_ob(i,1))^ 2))
   SINEPSP_ob(i,1)=sticla_ob(i,1)*(SINEPS_ob(i,1))/sticla_ob(i,1+1)
      EPSP_ob(i, 1) = Atn(SINEPSP_ob(i, 1) / Sqr(1 - (SINEPSP_ob(i, 1)) ^ 2))
      SIGP_ob(i, 1) = SIG_ob(i, 1) - EPS_ob(i, 1) + EPSP_ob(i, 1)
      G_ob(i, 1) = Q_ob(i, 1)/(Cos(EPS_ob(i,1))+Cos(SIG_ob(i,1)))
      QP_ob(i,1)=G_ob(i,1)*(Cos(EPSP_ob(i,1))+ Cos(SIGP_ob(i, 1)))
   Next i
   For i = 1 To 3
      For j = 2 To n - 1
         Q_ob(i,j)=QP_ob(i,j-1)-(da_ob(j-1))*Sin(SIGP_ob(i,j- 1))
         SIG_ob(i, j) = SIGP_ob(i, j - 1)
         S_ob(i, j) = Q_ob(i, j) / Sin(SIG_ob(i, j))
         SINEPS_ob(i,j)=Sin(SIG_ob(i, j)) - Q_ob(i, j) * ro_ob(j)
       EPS_ob(i,j)=Atn(SINEPS_ob(i,j)/Sqr(1-(SINEPS_ob(i,j))^ 2))
SINEPSP_ob(i,j)=sticla_ob(i,j)*(SINEPS_ob(i,j))/sticla_ob(i, j + 1)
EPSP_ob(i,j)=Atn(SINEPSP_ob(i,j)/Sqr(1-(SINEPSP_ob(i,j))^2))
         SIGP_ob(i, j)=SIG_ob(I,j)- EPS_ob(i, j) + EPSP_ob(i, j)
         G_ob(i,j)=Q_ob(i,j)/(Cos(EPS_ob(i,j))+Cos(SIG_ob(i, j)))
       QP_ob(i,j)=G_ob(i,j)*(Cos(EPSP_ob(i,j))+Cos(SIGP_ob(i,j)))
         SP_ob(i, j) = QP_ob(i, j) / Sin(SIGP_ob(i, j))
      Next j
   Next i
' calcul paraxial
   For i = 1 To 3
     spp_ob(i,1)=sticla_ob(i,2)/((sticla_ob(i,2)-sticla_ob(i,1))/ r_ob(1))
   Next i
   For i = 1 To 3
      For j = 2 To n - 1
         S_ob(i, j) = spp_ob(i, j - 1) - da_ob(j - 1)
         spp_ob(i, j) = (sticla_ob(i, j + 1)) / (sticla_ob(i, j) / S_ob(i, j) + (sticla_ob(i, j +
1) - sticla_ob(i, j)) / r_ob(j))
    Next j
   Next i
' Aberatia sferica
   aber_sfer_ob(repet) = SP_ob(1, n - 1) - spp_ob(1, n - 1) ' pleaca din origine (0,0)
' Aberatia cromatica paraxiala
   aber_crom_par_ob(repet) = spp_ob(3, n - 1) - spp_ob(2, n - 1) ' constant
' Aberatia cromatica
   aber_crom_ob(repet) = SP_ob(3, n - 1) - SP_ob(2, n - 1) 'variaza in functie de h
   Picture2.BackColor = RGB(0, 0, 0)
   Picture2.ForeColor = RGB(255, 255, 255)
Picture2.Cls
Picture2.AutoRedraw = True
Picture2.CurrentX = 5
Picture2.CurrentY = 30
Picture2.DrawWidth = 1
Picture2.Line -(55, 30)
Picture2.CurrentX = 30
Picture2.CurrentY = 30
Picture2.DrawWidth = 1
Picture2.Line -(30, 5)
Picture2.CurrentX = 30
Picture2.CurrentY = 30
   Picture2.ForeColor = RGB(255, 255, 0)
```

```
   Picture2.Line -((30 + 40 * aber_sfer_ob(repet)), 30 - 4 * h(repet))
   'Form2.List2.AddItem aber_sfer_ob(repet) & " | " & aber_crom_ob(repet) & "  cu  " &
h(repet)
Next repet
Picture2.CurrentX = 30 + aber_crom_par_ob(1)
Picture2.CurrentY = 30
Picture2.ForeColor = RGB(255, 0, 0)
For repet = 1 To 10
   Picture2.Line -((30 + 40 * aber_crom_ob(repet)), 30 - 4 * h(repet))
Next repet
For repet = 1 To 10
   Picture3.BackColor = RGB(0, 0, 0)
   Picture3.ForeColor = RGB(255, 255, 255)
Picture3.Cls
Picture3.AutoRedraw = True
Picture3.CurrentX = 5
Picture3.CurrentY = 30
Picture3.DrawWidth = 1
Picture3.Line -(55, 30)
Picture3.CurrentX = 30
Picture3.CurrentY = 30
Picture3.DrawWidth = 1
Picture3.Line -(30, 5)
Picture3.CurrentX = 30
Picture3.CurrentY = 30
   Picture3.ForeColor = RGB(255, 255, 0)
   Picture3.Line -((30 + aber_sfer(repet) + aber_sfer_ob(repet)), 30 - 3 * h(repet))
Next repet
Picture3.CurrentX = 30 + aber_crom_par(1) + aber_crom_par_ob(1)
Picture3.CurrentY = 30
Picture3.ForeColor = RGB(255, 0, 0)

For repet = 1 To 10
Picture3.Line -((30 + aber_crom(repet) + aber_crom_ob(repet)), 30 - 4 * h(repet))
Next repet
End Sub

Private Sub Command4_Click()

   End

End Sub
```

Bibliografie

A1. Altschuller, G. S., Erfinden – Wege zur Lösung technischer Probleme, Technik Verlag, Berlin, 1984

A2. Abeln, O., Die CA...-Techniken in der industriellen Praxis, Carl Hanser Verlag, München, Wien, 1990

B1. Bareket, N., Second moment of the diffraction point spread function as an image quality criterion, JOSA, Nr. 69, 1979, p.1311-1312

B2. Berek, M., Grundlagen der praktischen Optik, Walter de Gruyter, Berlin-Leipzig, 1930

B3. Bockmann, C.J., ş.a., Visual Basic, Biblioteca programatorului, Editura Teora, Bucureşti, 2001

B4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stahl, M., Pattern-oriented Software Architecture Volume 1. A System of Patterns, Wiley, 1996

B5. Beck, K., Smalltalk Praxisnahe Gebrauchsmuster, Prentice Hall, 1997

B6. Buschmann, F., "Multi-Threaded Architekturen mit Entwurfsmustern", OOP 2001

B7. Benchimol, G., Levine, P., Pomerol, J.-C., Sisteme expert in intreprindere, Editura Technica, Bucuresti, 1993

B8. Balzert, H., Lehrbuch der Objektmodellierung: Analyse und Entwurf, Spektrum Akademischer Verlag, Heidelberg, 1999

B9. Booch, G., Object Solutions, Addison Wesley, Menlo Park California, 1996

B10. Booch, G., Jacobson, I., Rumbaugh, J., The Unified Modelling Language for Object-Oriented Development, Rational Software Corporation, Santa Clara California, 1996

B11. Booch, G., Jacobson, I., Rumbaugh, J., The Unified Modelling Language – User Guide, Addison-Wesley, Massachussets, 1999

B12. Booch, G., Jacobson, I., Rumbaugh, J., The Unified Modelling Language – Reference Manual, Addison-Wesley, Massachussets, 1999

C1. Cretu, E., Iftimia, N., Optica laserilor, Ed. Academiei Tehnice Militare, Bucuresti, 1997

C2. Cretu, E., Marzu, M., Campean, M., Aplicatii in calculul si proiectarea sistemelor optice, Bucuresti, 1995

C3. Cretu, E., Nicoara, I. s.a.,Calculul si constructia aparaturii optoelectronice, Ed. Academiei Technice Militare, Bucuresti, 2001

D1. Dodoc, A., Analiza, sinteza şi optimizarea sistemelor variofocale. Teza de doctorat, Universitatea "Politehnica" Timişoara, 2001

D2. Dodoc, P., Calculul şi construcţia aparatelor optice, EDP, Bucureşti, 1983

E1. Ehrlenspiel, K., Kiewert, A., Lindemann, U., Kostengünstig Entwickeln und Konstruieren, Springer Verlag, Berlin, Heidelberg, 1998

G2. Gruescu, C., Pommersheim, A., Optica tehnică, Editura orizonturi Universitare, Timişoara, 1999

G1. Gruescu, C., Elemente de optică tehnică şi aparate optice, Editura orizonturi Universitare Timişoara, 2000

G2. Gamma, E., Helm, R., Johnsos, R., Vlissides, J., Entwurfsmuster, Addison-Wesley, 1996

G3. Goldberg, A., Smalltalk 80

G4. Griffel, F., Componentware: Konzepte und Techniken eines Softwareparadigmas dpunkt-Verlag, Heidelberg, 1998

G5. Goulouris, G., Dollimore, J., Kindberg, T., Distributed Systems, Addison Wesley, Menlo Park, California, 1994

G6. Goldberg, A., Rubin, K. S., Succeeding with Objects, Addison-Wesley, Menlo Park California, 1995

G7. Gheorghe, I.Ghe., s.a., Ingineria instrumentatiei, Editura CEFIN Bucuresti, 1999

H1. Heidinger, D., Zur Bildgute von Kleinobjektiven insbesondere bei ausseraxialer Abbildung, Mitteilungen und Berichte des optischen Instituts der TU Berlin H21,1982

H2. Hesse, S., Fertigungsautomatisierung, Vieweg Verlag, 2000

H3. Hansen, F., Konstruktionssystematik, Technik Verlag, Berlin, 1965

H4. Hansen, F., Konstruktionswissenschaft – Grundlagen und Methoden, Technik Verlag, Berlin, 1974

J1. Josuttis, N. M., "eXtreme Programming: ein Gespräch mit Kent Beck" in ObjektSpektum nr. 4 1999

J2. Jian, I.,Proiectarea bazelor de date, Editura Politehnica, 1983

J3. Jian, I., Baze de date, Editura Mirton, 1998

K1. Krause, W., Gerätekonstruktion in Feinwerktechnik und Elektronik, Carl Hanser Verlag, München, Wien, 2000

K2. Kuhlenkamo, A., Konstruktionslehre der Feinwerktechnik, Carl Hanser Verlag, München, 1971

K3. Krause, W., Konstruktionselemente der Feinmechanik 2 Aufl., Carl Hanser Verlag, München, Wien, 1993

L1. Löwe, M., Evolution Patterns - Eine formale Methode zur Entwicklung langlebiger Softwaresysteme auf der Basis von Graphgrammatiken,1996

L2. van der Lans, R. F., Das SQL Lehrbuch, Addison-Wesley, Bonn, 1987

M1. Mutzke, K., ABC der Optik, Carl Hauser Verlag, Munchen-Wien, 1987

M2. Mansfield, R., Programarea bazelor de date în Visual Basic 6, Editura Tehnică, Bucureşti, 2001

M3. Müller, J., Arbeitsmethoden der Technikwissenschaften, Springer Verlag, Berlin, 1990

M4. Möller, D., Modellbildung, Simulation und Identifikation dynamischer Systeme, Springer Verlag, Berlin, Heidelberg, 1992

M5. Marzu, M., Cretu, E., Nicoara, I., Optica ondulatorie si fouriere, Ed. Academiei Tehnice Militare, Bucuresti, 1996

N1. Nicoară, I., ş.a., Aparate optice, Editura Orizonturi Universitare, Timişoara, 2000

N2. Nicoară, I., Calculul şi construcţia aparatelor optice, vol.I, II, Lito IPT, Timişoara, 1988

O1. Oestereich, B., Objektorientrierte Softwareentwicklung, R. Oldenbourg Verlag, 1998

O2. Otte, R., Patrick, P., Roy, M., Understanding CORBA, Prentice Hall, 1996

O3. Oestereich, B., "Erfolgreich mit Objektorientierung", Oldenbourg Verlag, 1999

P1. Palmer, J., Lens Aberration Data, Adam Hilger, London, 1971

P2. Petkovic, D., INFORMIX: Das relationale Datenbanksystem, Addison-Wesley, Bonn, 1991

R1. Rosenbruch, K.J., Ein Rechnenprogramm zur Berechnung der optischen Übertragungsfunktion aus den Konstruktionsdaten eines optischen Systems, PTB-Mitteilungen, nr.5, 1968, p.384-390

R2. Rosenbruch, K.J., Die Bedeutung der optischen Ubertragungsfunktion bei der Konstruktion optischer Systeme, Optik, Nr.30,1969, p.346- 353

R3. Rosenhauer, K., Rosenbruch, K.j., Ein Diskusionsbeitrag zum Problem der Optimierung optischer Systeme unter Berucksichtigung der optischen Übertragungsfunktion, Optik, nr.25, 1967, p.95ff

R4. Rodenacker, W. G., Methodisches Konstruieren 3. Aufl., Springer Verlag, Berlin, Heidelberg, New York, 1984

R5. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W., Objektorientiertes Modellieren und Entwerfen, Carl Hanser Verlag, München, 1993

R6. Rupp, C., SOPHIST GROUP, Requirements-Engineering und- Management, Carl Hanser Verlag, München, 2001

R7. Reimelt, D.C., Utilizarea conceptelor din teoria programarii orientate pe obiecte intr-un sistem de proiectare asistata de calculator, Simpozionul International "Universitaria" ROPET 2001, Petrosani

S1. Schneider, G., Einfluss von spatialer Teilkoherenz auf die Messung optischer Übertragungsfunktion, Dissertation, TU Braunschweig, 1972

S2. Schmidt, D., Stal, M., Rohnert, H., Buschmann, F.,Pattern-oriented Software Architecture Volume 2. Patterns for Concurrent and Networked Objects, Wiley, 2000

S3. Spur, G., Krause, F.L., CAD-Technik, Carl Hanser Verlag, München, Wien, 1984

S4. Starke, G., Effektive Software-Architekturen: Ein praktischer Leitfaden, Carl Hanser Verlag, München, 2002

S5. Singer, G., Object Technology Strategies and Tactics, SIGS, New York, 1996

V1. Veluewen, A. van, Komponentenarchitekturen, in Objektspektrum Nr.4, 1999

V2. G. Versteegen, P. Kruchten, B. Boehm: "Projektmanagement mit dem Rational Unified Process", Springer Verlag, 2000

V3. G. Versteegen, P. Kruchten: "Vom Wasserfallmodell zum iterativen Lifecycle – ein harter Weg für Projektmanager" in ObjektSpektrum nr. 5 2000

V4. Gesamtverband der Deutschen Versicherungswirtschaft e.V.: VAA Die Anwendungsarchitektur der Versicherungswirtschaft, 1999

W1. Weck, M., Werkzeugmaschinen, Springer Verlag, 2001

W2. Wallmüller, E., "Ganzheitliches Qualitätsmanagement in der Informationsverarbeitung", Carl Hanser Verlag, München Wien, 1995

W3. Wirfs-Brock, R., Wilkerson, B., Wiener, L., Objektorientiertes Software-Design, Carl Hanser Verlag, München, 1993

Z1. Zendler, A., Mehmanesh, H., Mehmaneche, H., Komponentenorientierte Entwicklung im Softwarelebenszyklus: Identifizierung und Spezialisierung, in Objektspektrum Nr. 5, 2000

xxx http:// www.zeiss.de

xxx http://sculptor.as.arizona.edu

xxx http://www.imx.nl

xxx CAD CAM Magazin für Computeranwendung in Design und Engineering, Carl Hanser Verlag, Nr.1, Februar 2003,

xxx IT Industrielle Informationstechnik, Carl Hanser Verlag, Nr.1, März 2003

xxx OOP'97 in München Conference Proceedings

xxx OOP 2001 Kursinhalte

xxx OOP 2002 Kursinhalte

xxx Paradigm 3.0, Computec, Karlsruhe, 1995