**Universitatea Politehnica Bucuresti**

**Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei**

**Studenti: Griparis Andreea**

**Nedeloiu Dumitru-Valentin**

**Patrascu Cosmin-Mihnea**

**Grupa: 441A**

Cuprins

Proiectarea functionala: diagrame de flux de date si diagrame de structura, dictionar de date, sinteza diagramelor de structura.

Capitolul 1.(Patrascu Cosmin-Mihnea)

1.Diagrama fluxului de date(DFD).......................................................3

* 1. **Construirea DFD-urilor prin tehnica SSADM.........................4**
  2. **Realizarea diagramelor de flux de date (DFD)....………........5**
  3. **Conventii folosite in diagramele de reprezentare a DFD......6**

Capitolul 2.(Griparis Andreea)

1. Dictionare de date..................................................................................7
   1. **Concepte generale...................................................................7**
   2. **Dictionarul de date utilizat de ORACLE.................................9**

Capitolul 3. (Nedeloiu Dumitru-Valentin)

3.Diagrame de structura............................................................................11

**3.1. Introducere.......................................................................................11**

**3.2. Evaluarea Diagramelor de Structura..............................................11**

**3.2.1 Reprezentarea diagramelor structura....................................11**

**3.2.2 Formalizarea criteriilor de evaluare…………………………..14** **3.3Concluzii..............................................................................................17**

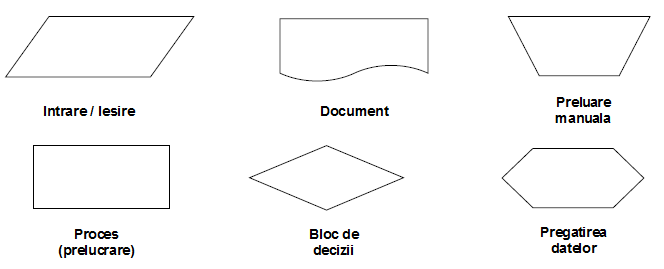
**Capitolul 1.Diagrama fluxului de date(DFD)**

**(Patrascu Cosmin-Mihnea)**

In scopul analizei si proiectarii sistemelor, ingineria software a adoptat o serie de sisteme de reprezentare, dintre care unele pot fi aplicate atat modelului descendent, cat si celui ascendent. Una dintre acestea este diagrama fluxului de date, in care accentul nu cade pe procedurile sau algoritmii ce urmeaza a fi executati, ci pe datele care vor circula prin sistemul propus. Aceasta abordare a aparut in stransa legatura cu modelul de proiectare imperativa, in ideea de a se urmari drumul datelor in cadrul sistemului si punctele in care ele sunt modificate. Pentru ca in aceste puncte sunt necesare activitatile de efectuare a calculelor, activitatile respective, sau grupari de activitati, vor forma modulele sistemului. Urmarindu-se fluxul de date se poate releva astfel o structura modulara a sistemului, fara a mai fi necesar ca acesta sa fie descompus intuitiv in componente.

Desi dezvoltata in contextul modelului de programare imperativa, analiza fluxului de date poate fi utilizata si in mediile orientate spre obiecte, unde poate ajuta la identificarea obiectelor necesare si a activitatilor pe care trebuie sa le efectueze acestea.

Diagramele de flux reprezinta o abordare invechita folosita ca instrument de proiectare grafica prin care se descrie mediul fizic si etapele unui proces in cadrul unui S.I. Prin aceste diagrame se descriu procesele care se desfasoara intr-un program din aplicatie / sistem, in succesiunea in care se executa. Acest sistem este utilizat pentru a face documentatia specificatiilor in proiectarea fizica, mai ales a procedurilor manuale din S.I., deoarece se pot evidentia intrarile, fisierele, prelucrarile si iesirile,folosind simboluri speciale si linii de flux.



Exemple de simboluri folosite in diagramele de flux

Diagrama fluxului de date este o reprezentare grafica a rutelor circulate de date in cadrul sistemului. Simbolurile utilizate in diagrama au semnificatii bine stabilite: sagetile reprezinta itinerariul datelor, dreptunghiurile arata locurile in care datele sunt prelucrate, iar liniile groase reprezinta stocarea datelor. Fiecare simbol are o eticheta care specifica numele obiectului reprezentat.

Descrieri ale obiectelor DFD se regasesc in asa-zisele dictionare ale proiectelor sau depozitele CASE .

Diagramele fluxului de date DFD au ca obiectiv urmarirea modului de transfer al

datelor intre procesele de prelucrare a lor, astfel de diagrame se mai numesc si modele ale proceselor de prelucrare, iar operatiunea se numeste modelarea proceselor.

DFD reprezinta doar una din tehnicile de analiza structurata.

Diagrame fizice ale fluxurilor de date (DFDF): prezinta atributele fizice ale procesului. Nu prezinta atributele logice ale procesului.Sunt niste reprezentari ale sistemului prin care sint scoase in evidenta entitatile interne si externe ale sistemului, precum si fluxurile datelor in si din aceste entitati. Ele se pot modifica relativ des si nu pot fi descompuse.

O entitate interna poate fi o persoana, un loc de munca, un sector, un echipament, un calculator de sistem care contribuie la prelucrarea automata a datelor. Din aceasta cauza DFDF specifica unde, cum si de catre cine este realizat un proces al sistemului. Ea specifica ce realizeaza sistemul.

Diagrame logice ale fluxurilor de date (DFDL) trebuie sa raspunda la intrebarea “CE”face orice proces din DFDF .

DFDL prezinta reprezentarea logica a procesului. Sunt descrieri abstracte care nu specifica nici cine efectueaza o anumita activitate si nici locul in care se afla anumite elemente. Daca subprocesul nu este elementar, acesta poate fi descompus sau partitionat. Astfel de diagrame se modifica foarte rar.

Tehnica de redare a proceselor de prelucrare prin intermediul diagramelor fluxurilor de date a capatat noi acceptiuni prin incorporarea ei in instrumentele de analiza si proiectare cu ajutorul calculatorului, adica in instrumente CASE.

Diagramele fluxului de date au fost propuse de Larry Constantine, primul dezvoltator al design-ului structurat, bazat pe modelul lui Martin si Estrin.

**1.1.Construirea DFD-urilor prin tehnica SSADM (Structured Systems Analysis and Design Methodology)**

Pentru analiza sistemelor se folosesc frecvent reprezentari grafice (diagrame). O diagrama a fluxului de date este un graf care arata sensul in care sunt transformate (“curg”) datele, fluxul valorilor datelor incepand de la sursa lor din obiecte, trecand prin prelucrarile (procesele) care le transforma, spre destinatia lor in alte obiecte.

Scopul diagramelor de date DFD pentru o anumita componenta organizatorica sau functionala la care se refera (sectie, birou, compartiment, intreaga unitate, o anumita activitate – vanzari, cumparari, incasari, plati, s.a) este de a scoate in relief, intr-o maniera cat mai sugestiva, urmatoarele aspecte:

-sursa datelor de prelucrare;

-operatiunile de prelucrare prin care trec datele;

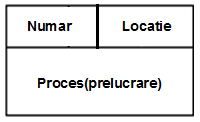
-destinatia datelor prelucrate;

-legatura existenta intre prelucrari si activitatea de stocare a datelor.

**1.2.Realizarea diagramelor de flux de date (DFD)**

DFD este o reprezentare grafica a transformarii datelor de intrare in date de iesire   
folosind un set de simboluri de reprezentare si un set de reguli de completare si validare.

Simboluri folosite in diagramele realizate cu SSADM

****

Proces (prelucrare): -procesele transforma valorile datelor, prelucrarile de cel mai jos nivel sunt functii pure (care doar calculeaza) fara efecte laterale.

Procesul se simbolizeaza printr-un dreptunghi continand o descriere a prelucrarilor si sunt identificate printr-un numar(descriere a functiei procesului de prelucrare, incepand cu un verb, urmat de o descriere a obiectului functiei de prelucrare). In DFD fizica pentru sistemul existent, se va preciza si locatia (compartiment / persoana) procesului.



Flux de date*: -*este cantitatea de date transmisa intre entitatile modelate. Fluxul de date este simbolizat printr-un substantiv ce sugereaza informatia sau pachetul de informatii transmise.



Entitate externa (terminator): -se simbolizeaza printr-o elipsa (sau un cerc)si este o sursa sau un receptor de date. Poate fi un alt sistem (organizatie, compartiment).



Stoc de date (colectie): -este un obiect pasiv in interiorul unei diagrame a fluxului de date care stocheaza date pentru un acces ulterior. Un stoc de date nu angajeaza operatii din proprie initiativa, dar raspunde cererilor de stocare sau accesare a datelor. Colectiile de date permit accesul la valori intr-o ordine diferita de cea in care au fost produse. Un stoc de date poate fi un depozit temporar sau permanent de date.

Liniile care intra indica informatiile si operatiile care modifica datele stocate (stergere, adaugare, modificare), iar liniile care ies indica informatiile preluate din colectie.

Un stoc de date poate fi:

* manual: registre, dosare, arhiva de documente
* pe suport magnetic: fisiere.

**1.3.Conventii folosite in diagramele de reprezentare a DFD:**

* procesele si stocurile de date sunt numerotate secvential, pentru a putea fi identificate. Numerele asociate proceselor nu semnifica ordinea de executie a acestora;
* pentru a evita fluxurile de date intretaiate si aspectul de “paienjenis” al diagramei, entitatile externe si stocurile de date pot fi duplicate. O entitate externa duplicata se reprezinta prin trasarea unei linii oblice, iar un stoc duplicat printr-o linie suplimentara verticala in partea stanga a cutiei;
* pentru a face diagramele mai lizibile, entitatile externe sunt plasate, pe cat posibil, in jurul diagramei iar stocurile de date, in partea centrala a diagramei;
* fluxurile de date de la - catre stocurile de date sunt unidirectionale (fie de adaugare, fie de consultare) si nu sunt etichetate.

Bibliografie: - The Practical Guide to Structured Systems Design - M. Page-Jones

- Proiectarea sistemelor informatice - Nicolae Morariu

- Revista Informatica Economica

- <http://en.wikipedia.org>

**Capitolul 2.Dictionare de date**

**(Griparis Andreea)**

**2.1 Concepte generale**

Un alt instrument folosit în procesul de dezvoltare a sistemelor software îl constituie dictionarul de date**,** care este centralizatorul tuturor informatiilor referitoare la datele utilizate în cadrul sistemului. Aceste informatii cuprind: identificatorul utilizat pentru fiecare element, valorile valide (tip numeric sau alfanumeric, interval permis de valori etc.), locul de stocare (fisier sau baza de date, numele acesteia), locurile în care elementul respectiv este folosit de catre program (ce module contin referiri la el).

Dezvoltarea unui astfel de dictionar de date are mai multe obiective, în primul rând, existenta lui poate îmbunatati comunicarea între viitorul utilizator al sistemului si analistul care transforma solicitarile acestuia în cerinte si specificatii.

Alt obiectiv este acela de a se asigura o abordare unitara a sistemului.

Un dictionar de date este utilizat pentru a indica mult mai general un software utilitar decat un catalog. Un catalog este strans legat de de sistemul de gestiune al bazei de date (SGBD); acesta furnizeaza informatiile stocate in acesta si administratorul bazei de date (DBA), dar este accesat mai ales de diferite module software, cum ar fi DDL (data description language) si compilatoare DML (data manipulation language), optimizatorul interogarilor, procesorul de tranzactii, generatotul de rapoarte si aplicatorul constrangerilor. Pe de alta parte dictionarul de date este o structura ce contine meta-date. Pachetul software pentru un dictionar de date de sine stator poate interactiona cu modulele software ale SGBD-ului, dar este utilizat mai ales de proiectantii, utilizatorii si administratorii unui sistem informatic pentru gestionarea resurselor de informatii. Aceste sisteme sunt utilizate pentru a mentine informatiile despre sistemul hardware, configuratia software, documentatie, aplicatii si utilizatori la fel ca si alte informatii relevante pentru administrarea sistemului.

Daca dictionarul de date este utilizat doar de proiectanti, utilizatori si administratori si nu de software-ul SGBD, acesta se numeste Dictionar de Date Pasiv; altfel se numeste Dictionar de Date Activ sau Dictionar de Date. Un Dictionar de Date Activ este actualizat automat cu schimbatile aparute in baza de date. Un Dictionar de Date Pasiv trebuie actualizat manual.

Un dictionar de date este o parte integranta a unei baze de date. El detine informatii despre baza de date si despre datele stocate de aceasta. Orice baza de date ine proiectata va include, cu siguranta, un dictionar de date, deoarece ofera administratorilor si utilizatorilor de BD (baza de date) acces usor tipului de date pe care ei se asteapta sa il gaseasca in fiecare tabel, rand si coloana a BD, fara accesarea propriu-zisa a acesteia.

Cateva din componentele tipice ale unui dictionar de date:

* Numele tabelei;
* Numele campurilor din fiecare tabela;
* Tipul datei campului(intreg, data, text…);
* Lungimea campului;
* Valoarea implicita a acelui camp;
* Posibilitatea campului de a fi nul sau nu;
* Constrangerile aplicabile pentru fiecare camp.

Nu vom aplica toate aceste campuri (si multe altele) fiecarei intrari din dictionarul de date. De exemplu, daca intrarea a fost cu privire la descrierea radacinii tabelului, este posibil sa nu fie necesara nicio informatie cu privire la campuri. Deasemenea unele dictionare de date includ detalii privind locatia, cum ar fi localizarea curenta a fiecarui camp si detalii privind locatia fizica cum ar fi adresa IP sau DNS a server-ului.

**Formatul si stocarea**

Nu exista niciun format standard pentru a crea un ditionar de date. Meta-datele difera de la o tabela la alta. Unii administratori de BD prefera sa creeze fisiere de text simple, in timp ce altii folosesc diagrame de flux pentru a afisa toate informatiile lor. Singura conditie impusa unui dictionar de date este de a fi usor de cautat.

Singura regula ce trebuie aplicata unui dictionar de date este sa fie intr-o locatie convenabila, care este usor accesibila tuturor utilizatorilor BD. Tipurile de date ale campurile difera de la fisiere text, fisiere xml, foi de calcul, un tabel suplimentar in BD la note scrise de mana. Este de datoria administratorului de BD sa se asigure ca acest document este intotdeauna actualizat, precis si usor de accesat.

**Crearea unui dictionar de date**

In primul rand, trebuie indentificate si inregistrate in documentele de proiectare toate informatiile necesare pentru a crea un dictionar de date. Daca documentele de proiectare sunt intr-un format compatibil, ar trebui sa fie posibil exportul direct al acestor date, in formatul dorit, in dictionarul de date. De exemplu, aplicatii ca Microsoft Visio permit crearea BD direct din structura de proiect, ceea ce face crearea dictionarului de date mai simple. Chiar si fara utilizarea acestor instrumente, script-urile pot fi utilizate in exportarea datelor din BD in document. Exista intotdeauna posibilitatea crearii manuale a acestui document.

**Avantajele dictionarului de date**

* Da claritate restul documentatie bazei de date;
* Cand este introdus in sistem un utilizator sau administrator nou, acestuia ii este mai usor sa se familiarizeze cu structurile si tipurile tabelelor existente in BD;
* Este crucial in cazul scenariilor care implica baze de date foarte mari, deoarece administratorului ii este imposibil sa-si aminteasca bitii specifici de informatie despre miile de domenii;
* Multe persoane pot inventa nume pentru entitati si relatii cand dezvolta un model larg de sistem. Aceste nume trebuie utilizate consecvent si nu trebuie sa intre in conflict. Software-ul de dictionar de date poate verifica, daca este necesar, unicitatea numelui si avertiza in cazul numelor duplicat.

**2.2. Dictionarul de date utilizat de ORACLE**

Ca si in alte tipuri de BD la cele dezvoltate de ORACLE dictionarul de date este una din cele mai importante parti ale unei BD, acesta consta dintr-un set de tabele read-only care furnizeaza informatii despre BD asociata. Dictionarul de date cuprinde: definitia tuturor obiectelor schemei BD, cat spatiu s-a locat pentru obiectele schemei si cat este utilizat, valorile implicite ale coloanelor, informatii despre restrictiile de integritate, numele utilizatorilor Oracle, privilegiile si rolurile acordate fiecarui utilizator, informatii de audit, etichetele tuturor utilizatorilor si obiectelor schemei.

Dictionarul este structurat in table si view-uri ca oricare alta baza de date. Toate tabelele dictionarului sint mentinute in spatiul de tabela SYSTEM.

Dictionarul este un instrument pentru toti utilizatorii, de la end user la administratorii bazei de date.

Pentru accesarea dictionarului, se folosesc directive SQL. Deoarece dictionarul este read only, se lanseaza numai interogari (SELECT) asupra tabelelor si view-urilor dictionarului.

**Structura dictionarului**

Dictionarul bazei de date contine:

* tabele de baza - tabele fundamentale care memoreaza informatii despre baza de date asociata.

Numai Oracle poate scrie si citi aceste tabele.Utilizatorii acceseaza rar aceste tabele direct deoarece acestea sint nominalizate si de cele mai multe ori criptate.

* view-uri accesibile utilizatorilor - view-uri care sumarizeaza si afiseaza informatiile memorate in tabele de baza ale dictionarului. Aceste view-uri decodifica tabelele de baza in informatii utile, cum ar fi numele utilizatorului sau a tabelei utilizind joinari si clauza WHERE pentru a simplifica informatiile. Utilizatorii au acces la view-uri mai mult decit la tabelele de baza.

Bibliografie: - Software engineering de Ian Sommerville

- <http://en.wikipedia.org/wiki/Data_dictionary>

- <http://www.tech-faq.com/data-dictionary.html>

- econ.ubbcluj.ro

**Capitolul 3.** **Diagrame de structura  
(Nedeloiu Dumitru-Valentin)**

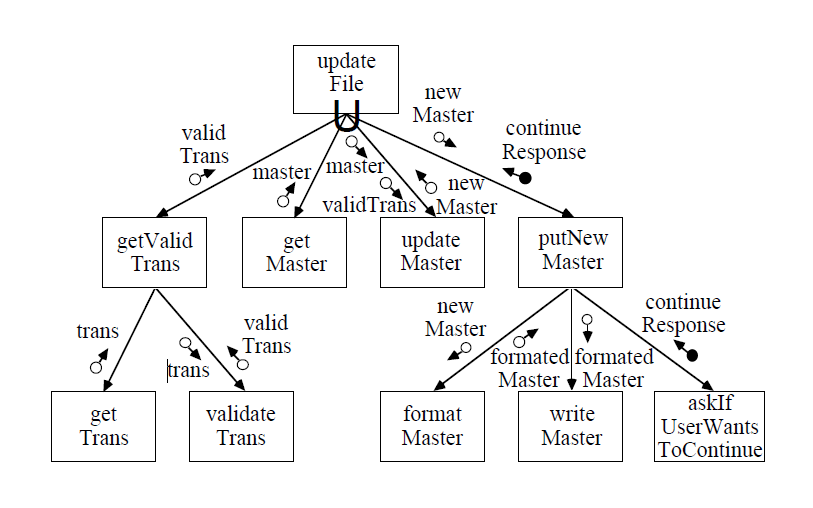
**3.1. Introducere**

Sisteme structurate de metodologii de dezvoltare au fost recunoscute ca fiind unele dintre cele mai populare metode în inginerie software Ei au avut probleme in tehnici de rezolvare şi procese de decizie, care pot fi efectuate numai de către ingineri software cu experienţă. Deşi o serie de instrumente CASE cu privire la metodologiile structurate au fost deja propuse, acestea nu sunt complet satisfăcătoare,deoarece este extrem de dificil de a avea expertize umane formalizate în termeni conventionali şi puse în aplicare prin programe algoritmice.

**3.2. Evaluarea Diagramelor de Structura** În evaluarea  diagramelor de structua , ne preocupă două probleme: cum se reprezinta o diagramă de structura şi modul de a formaliza criteriile de evaluare a diagramelor .Prima problemă este un studiu de reprezentare a cunoştinţelor. A doua problemă este puţin mai complexă, deoarece de cele mai multe ori criteriile propuse în literatura de specialitate sunt mai degrabă vagi şi imprecise.

**3.2.1 Reprezentarea diagramelor structura**

O caracteristica unica de programare logică este că se poate utiliza un set de relaţii pentru a reprezenta o structură de date. Noi nu vom discuta argumentele pro şi contra  pe bază de reprezentare.Controversa noastră este că o diagramă de structură este de obicei mare, şi, prin urmare ar fi mai degrabă greoaie pentru a codifica diagrama ca un termen imens.



**Figura 1 Exemplu de diagrama de structura**

Avem nevoie să ştim (a) toate modulele din diagramă, (b) toate elementele de date din diagramă, (c) organizarea de module, şi (d) toate comunicările între module.  Ca un exemplu,pentru ilustrare, luaţi în considerare figura 1, care a fost adaptata de la Page-Jones[12]. Acesta prezinta o parte a unui sistem interactiv pentru actualizarea unui fişier. Mai întâi de toate, vom identifica toate modulele din  sistemu prin definirea predicatului *is\_module*:

is\_module (updateFile).

is\_module (getValidTrans).

is\_module (getTrans).

is\_module (validateTrans).

is\_module (getMaster).

. . .

Apoi, vom identifica toate elementele de date prin intermediul predicatului *is\_data.*Primul argument al is\_data este numele unui element de date. Al doilea argument ne spune dacă acesta este un element simplu (atomic), un element compozit (înregistrare) sau un steag (de control).

is\_data (trans, record).

is\_data (validTrans, record).

is\_data (continueResponse, control).

. . .

Apoi vom specifica organizarea modulelor. Predicatul structure (ParentModule, Tip, ChildModules) descrie conexiunea între un ParentModule şi o listă de ChildModules .Tipul de conexiune poate fi secvenţă, de selectare sau iteraţie, care sunt notate de către următoarele constante seq, sel sau itr respectiv.

structure (updateFile, itr, [getValidTrans, getMaster,

updateMaster, putNewMaster]).

structure (getValidTrans, seq, [getTrans,

validateTrans] ).

structure (putNewMaster, seq, [formatMaster, writeMaster,

askIfUserWantsToContinue] ).

În cele din urmă, vom specifica comunicarea între module. Predicatul *coupling (Data,  SourceModule, TargetModule*),  atunci când ar deţine fluxuri de date de la *SourceModule* la *TargetModule*. astfel:

coupling (validTrans, getValidTrans, updateFile).

coupling (trans, getTrans, getValidTrans).

coupling (validTrans, validateTrans, getValidTrans).

. . .

Aceasta completează caietul de sarcini al unei diagrame de structură.

În ciuda simplităţii metodei de reprezentare, informaţiile utile pot fi extrase prin mecanismul de deducere care stau la baza, care sunt transparente pentru utilizator. De exemplu, este usor sa gasesti toate datele de intrare şi de ieşire  elementele date de *Module* prin definirea predicatelor următoare:

input\_data (Module, DataItems) :-

is\_module (Module),

findall (Data, coupling (Data, \_, Module), DataItemsB),

delete\_dup (DataItemsB, DataItems).

output\_data (Module, DataItems) :-

is\_module (Module),

findall (Data, coupling (Data, Module, \_), DataItemsB),

delete\_dup (DataItemsB, DataItems).

Predicatul *coupling (data, \_, Module)* ar deţine în cazul în care modul dat acceptă niste de date de la unele modul arbitrar  (notate de către ''\_''). Predicatul *findall  (data,  Goal, DataItems),* atunci când ar deţine *DataItems*este o listăde cazuri de date, astfel încât este urmat de *Goal.* Predicatul  *delete\_dup  (DataItems, Rezultat)* ar deţine atunci când Result este o listă obţinuat prin eliminarea tuturor elementelor duplicate din listă de DataItems.

Mai mult, putem decide dacă un Module dat este un modul get, un modul put sau un modul transform prin simpla adăugare de următoarele reguli:

module\_type (Module, get) :-

/\* nothing flows into it, but something flows out \*/

input\_data (Module, []),

output\_data (Module, [\_|\_]).

module\_type (Module, put) :-

/\* nothing flows out of it, but something flows in \*/

input\_data (Module, [\_|\_]),

output\_data (Module, []).

module\_type (Module, transform) :-

/\* something flows in and something flows out \*/

input\_data (Module, [\_|\_]),

output\_data (Module, [\_|\_]).

3.2.2 Formalizarea criteriilor de evaluare

a)de cuplare şi de coeziune

Cuplare este o măsură de interdependenţă între module diferite. Modulele ar trebui să să fie slab cuplate, sau relativ independente. Există cinci tipuri majore de cuplare. de date de cuplare şi ştampila de cuplare, înseamnă că două module comunică prin elemente de date compozite respectiv atomice . Acesta este  cel mai bun tip de cuplare. Cuplarea de control  înseamnă că două module comunică  prin  steaguri de control. Cuplare comuna înseamnă că două module împart datele. Primele trei tipuri de cuplare pot fi uşor de detectat la reprezentarea noastră a diagramei de structura. De exemplu, putem defini un  predicat *data\_coupling  (Module1,Module2)*, care ar deţine  modulele  *Module1*şi *Module2*.

data\_coupling (Module1, Module2) :-

(coupling (Data, Module1, Module2);

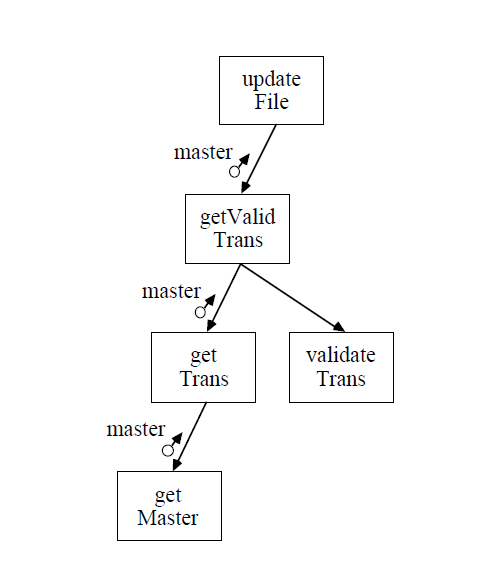
coupling (Data, Module2, Module1)),

is\_data (Data, atomic).

Coeziune este o măsură a puterii de asociere a elementelor într-un modul. Acesta este recomandă pentru elementele  extrem de coezive, sau puternic legate între ele. Există şapte niveluri majore de coeziune. Coeziunii funcţionale înseamnă că modulul efectuează o singură funcţia identifiabila, şi este cel mai bun tip de  coeziune. Coeziune secvenţială înseamnă că datele produse într-o parte anterioară a modulului vor fi folosite într-o parte mai târziu din acelaşi modul. In de coeziune de comunicare, elementele  modulului sunt în acelaşi fişier, dar nu neapărat în ordinea  specificata. În coeziune procedurala, elementele din modulul sunt legate de algoritmi  de control al programului, cum ar fi de selectare sau iteraţie. In coeziunea de timp,  elementele sunt grupate sub un singur modul, deoarece acestea sunt legate de timp. în  coeziunea logică, elementele sunt grupate sub un singur modul, deoarece se presupune ca au similare comportamentul, dar prezintă de fapt diferenţe minore. Coeziunea de coincidenţă înseamnă că elemente din modulul sunt grupate  împreună  pentru nici un motiv special, şi este cel mai rău tip de de coeziune.

b) date Tramp

Potrivit Page-Jones , un tramp ''este o bucată de informaţii care se amestecă fără ţintă în jurul valorii unui sistem, nedorit - şi lipsita de sens - in cele mai multe dintre modulele prin care trece.''Această definiţie este destul de vagă, ca termenii''sens''şi ''cele mai multe sunt doar'' concepte intuitive şi niciodată în mod oficial definite. Este destul de dificil de a converti o astfel definiţie într-un algoritm de programare convenţionale. Cu toate acestea, designeri cu experienţă sunt în măsură să recunosca date cu de tip tramp, în cele mai multe cazuri. În figura 2, de exemplu, elementul de date master apare lipsit de sens la modulul getTrans şi getValidTrans . Acest tip de cunoaştere poate fi manevrate mai uşor de programarea logică decât de programarea convenţionala.



Am putea încerca să se precizeze noţiunea de sens, utilizând predicatul simplu indicat mai jos. Se verifică dacă o bucată din Data passes în şi în afara unui modul direct.

meaningless (Data, Module) :-

coupling (Data, Module1, Module),

coupling (Data, Module, Module2).

Inainte de a sari la concluzii, cu toate acestea, să ne aruncăm o privire la modulul getValidTrans în figura 1. În conformitate cu definiţia noastră simplă, poziţia de date trans-ar filipsită de sens la modul, şi totuşi cele mai multe practicanţii uman nu-l privesc ca pe unvagabond. Motivul este faptul că acest proiect urmează de fapt o altă directoare cu privire la captarea datelor. Aceasta este, getValidTrans modul ascunde caracteristicile fizice ale trans din module mai mari în diagramă. Acest lucru este util mai ales pentru obţine şi pune module. O mai elaborata definiţia de sens ar trebui, prin urmare, implica verificarea dacă atat sursa cat si module de destinaţie de un element de date sunt copii ale aceluiaşi modul(sau frati):

meaningless (Data, Module) :-

coupling (Data, Module1, Module),

coupling (Data, Module, Module2),

not siblings(Module1, Module2).

siblings (Module1, Module2) :-

parent\_child (ParentModule, Module1),

parent\_child (ParentModule, Module2).

parent\_child (ParentModule, Module1) :-

structure (ParentModule, \_, ChildModules),

member (Module1, ChildModules).

Un alt termen problematic, în definiţia de tramp este cel “mai''. Ar trebui să fie clar faptul că aceasta este menit să fie o măsură relativă, mai degrabă decât una  absolută. Acest lucru poate fi reprezentat cu uşurinţă în Prolog, după cum urmează:

tramp (Data, UserDefinedRatio) :-

input\_output (Data, ModuleList),

length (ModuleList, NoOfInputOutput),

NoOfInputOutput > 0,

sub\_list (ModuleList, Data, TrampModuleList),

length (TrampModuleList, NoOfTrampInputOutput),

Ratio is NoOfTrampInputOutput / NoOfInputOutput,

Ratio > UserDefinedRatio.

sub\_list ([ ], Data, []).

sub\_list ([Module | ModuleList], Data,

[Module | TrampModuleList]) :-

sub\_list (ModuleList, Data, TrampModuleList),

meaningless (Data, Module).

sub\_list ([Module | ModuleList], Data, TrampModuleList):-

sub\_list (ModuleList, Data, TrampModuleList).

**c)Subdiagramele plate**

Un alt ghid pentru evaluarea diagramelor de  structură este că modulele de mai sus trebuie să prelucreze date abstracte'''', sau date cu caracteristici mai puţin''fizice”.  Astfel, aceste date , sunt denumite *validTrans* care  este mai abstract decât un alt element de date numit *trans*, deoarece a suferit probabil un proces de validare.

Dar de unde ştim, în general, dacă un element de date este mult mai abstract decât altul? O posibilitate este de a imita modul în care un cititor uman ar determina nivelul de abstractizare -uitanduse la numele de elemente de date. Putem încerca să definim un predicat more\_abstract\_than (A, B), care ar deţine date atunci când A este mai abstract decât B, astfel:

more\_abstract\_than (A, B) :-

abstract\_ prefix (Prefix),

append\_terms (Prefix, B, A).

în cazul în care abstract\_ prefix este definit de o colecţie de afirmaţii, cum ar fi

abstract\_ prefix (edited).

abstract\_ prefix (valid).

abstract\_ prefix (invalid).

abstract\_ prefix (matched).

abstract\_ prefix (unmatched).

Acest lucru este mai degrabă nesatisfăcătoare, deoarece nu există nici o flexibilitate în regulă.  Orice prefix utilizat în afara de vocabularul de date nu vor fi recunoscute. În plus, nu ar funcţiona în cazul în care proiectantul sistemului utilizeaza o altă convenţie de denumire pentru elemente de date. De exemplu, un modul care pregăteşte pentru facturi de ieşire se poate accepta ca intrare şiinvoice\_information  
produc *edited\_invoice\_information.*Deşi elementul de date doua îşi desfăşoară cu ea un prefix de editat, acesta este de fapt mai mult fizică decât *invoice\_information*.

**3.3Concluzii**

Aceasta lucrare este parte a unui efort pe termen lung in a aplica tehnici de programare logică la sisteme structurate de dezvoltare. Găsim că programarea logică  este un instrument util în acest sens. Noi putem reprezenta diagrame de structura in mod natural, şi din care provin informaţiile semnificative într-o simplă manieră.Tehnicile standard de evaluarea  a diagramelor de  structură poate fi formalizată, şi câteva probleme anterioare pot fi rezolvate cu uşurinţă.

Găsim că Prolog este un instrument foarte natural pentru detectarea de anomalii în diagramele de structură. Există mai mulţi factori care contribuie la acest lucru: în primul rând, nu putem fi pe deplin conştienţi de toate formele de anomalii la început, deoarece cele mai multe dintre ele sunt bazate pe experienţa individuală şi nu sunt formal definite în literatura de specialitate. Astfel de forme de anomalii pot fi  
specificate într-un program Prolog treptat. În al doilea rând, suntem mai interesaţi de ceea ce o anomalie arata si nu foarte mult în cauză cu privire la algoritmul pentru a le detecta. Dacă vom folosi Prolog, putem formula  proceduri pentru detectarea  diagrame  problematice de structura prin simpla traducerea a anomaliilor  în predicate, şi se lasă mecanismul de detectare a sistemului. În al treilea rând, elementele de identificare  în diagramele de  structură va avea acelaşi nume dacă şi numai dacă se referă la acelaşi obiect . Prin urmare, ele sunt mult mai natural definite fata de  variabilele logice  din Prolog decât variabile într-un limbaj de programare convenţional pe bază de atribuire.

Bibliografie:

1.M. Page-Jones, *The Practical Guide to Structured Systems Design*, Yourdon Press Computing Series, Prentice Hall, Englewood Cliffs, New Jersey (1988).

2. R.A. Ko walski, *Logic for Problem Solving*, North-Holland, Amsterdam (1979).

3.R.A. Ko walski, ‘‘Software engineering and artificial intelligence in new generation

computing’’, *Future Generation Computer Systems* **1** (1): 39−49 (1984).