PROIECTAREA SISTEMELOR INFORMATICE

CUPRINS

1. Introducere

Etapele clasice ale dezvoltarii unui sistem informatic (SI) sunt urmatoarele:

1. Studiul de fazabilitate
2. Analiza de sistem
3. Proiectarea
4. Codificarea
5. Testarea integrarii

6. Implementarea

Pentru fiecare din aceste etape au fost dezvoltate diverse unelte si metodologii.

Pentru analiza de sistem exista ***SSA (Structured Systems Analysis)*** care utilizeaza diagrame pentru a descrie un sistem informatic. Pentru proiectare se poate folosi ***SD (Structured Design)*** prin folosirea careia se obtine o arhitectura de sistem bazata pe fluxul datelor. In etapa de codificare, folosirea ***SP (structured Programming)*** poate duce la o anumita ordonare a surselor sistemului informatic.

Toate aceste metode au insa un caracter prescriptiv, enuntand reguli care guverneaza etapa respectiva dar nu sunt suficient de formalizate pentru a garanta consistenta intre intrarea intr-o etapa de proiectare si iesirea acesteia. Verificarea consistentei se poate face in doua moduri:

1. Testarea manuala a consistentei: aceasta metoda duce la scaderea numarului de erori dar nu garanteaza consistenta.
2. Existenta unei metode de specificare formala care sa permita un lant de transformari de asemenea formale si care pot duce eventual chiar la generare de cod. Aceasta metoda necesita insa aparatul formal care sta la baza specificarii si transformarilor. Avantajul este ca se poate demonstra formal consistenta intre intrarea si iesirea etapei de proiectare respective.

Acesta prezentare contine dezvoltarea celei de-a doua metode, cu aplicabilitate mai ales la sistemele informatice interactive. Scopul ei este de a prezenta o metodologie care poate produce automat o implementare exacta a unui sistem informatic. O implementare se spune ca este exacta daca exista o demonstratie matematica a afirmatiei: comportarea sistemului implementat este aceeasi cu a specificatiilor sale. Acesta este insa un scop cu o rezolvare completa in urmatorii ani; in abordarea de fata se da o rezolvare partiala a problemei.

In prezentarea de fata se va prezenta metoda ***ADISSA (Architectural Design of Information Systems based on Structured Analysis)*** care este o extensie a SSA. In contextul celor spuse mai sus vom vedea ca se poate obtine o demonstratie matematica a consistentei intre specificatia si proiectul partii de control a tranzactiilor ADISSA. Pentru modelarea acestor tranzactii se folosesc ***automatele finite (AF)*** din cauza faptului ca:

1. AF sunt bine definite matematic, permitand demonstrarea consistentei.
2. SI interactive sunt usor de modelat si realizat folosind AF.
3. Codul pentru un AF este scurt si simplu.

Prezentarea metodologiei se va pace in mai multi pasi:

Capitolul 2 contine o trecere in revista a procesului de dezvoltare a unui SI pe baza metodologiei prezentate.
Capitolul 3, orientat pe analiza de sistem, contine prezentarea metodei ADISSA. Este introdus aici conceptul de tranzactie (in sens ADISSA).
Capitolul 4 contine formalizarea specificarii tranzactiilor, introducandu-se conceptul de tranzactie AF si reguli de transformare a tranzactiilor ADISSA in tranzactii AF. ***Capitolul 5*** prezinta etapa de proiectare. Sunt introduse automatele finite si algoritmi de realizare a lor. Capitolul 6 contine teoremele mai importante care duc la demonstrarea consistentei intre specificatia formala a unei tranzactii si realizarea sa prin AF.
Capitolul 7 ultimul, contine asertiuni si restrictii legate de metoda prezentata si sunt mentionate directiile de dezvoltare viitoare.

2. Procesul de dezvoltare a unui sistem informatic

In figura 1 sunt prezentate etapele de dezvoltare ale unui SI, etape impuse de metoda pe care o descriem.

Obiectul metodei il formeaza etapele de analiza, formalizare, proiectare si verificarea consistentei (deci fara etapa de codificare). Scopul urmarit este formalizarea intrarilor si iesirilor in/din fiecare etapa pentru a face posibila demonstrarea consistentei lor.

Fiecare dintre aceste etape este o rafinare a precedentei, marind precizia si gradul de detaliere dar introducand in acelasi timp restrictii care sunt consecinte ale deciziilor care se iau pe parcursul acelei etape. Aceste decizii vor influenta corespunzator etapele ulterioare. Este bine de precizat ca aceste restrictii trebuie introduse cu grija pentru a nu restringe posibilitatile de alegere in etapele urmatoare.

Dupa cum se vede si in figura, primul pas al dezvoltarii este analiza de sistem, care se va formaliza cu ajutorul metodei ADISSA prezentata in cap. 3. Aceasta metoda este orientata catre specificarea prelucrarilor in sisteme informatice interactive. Conceptul de tranzactie care se va introduce aici semnifica o prelucrare independenta definita de utilizator. Rezultatele aplicarii ADISSA sunt specificatiile semi-formale ale SI. Ela nu au un caracter total formal deoarece pentru fiecare tranzactie se pot da mai multe interpretari, inclusiv unele care nu sunt necesare utilizatorului SI.

Pasul al doilea, formalizarea, descris in cap.4., este bazat pe abordarea transformationala.
El cuprinde selectarea interpretarilor relevante pentru fiecare tranzactie si descrierea metodei folosite pentru obtinerea tranzactiilor AF. Sunt enumerate regulile pe baza carora o tranzactie ADISSA se transforma in tr-o tranzactie AF.

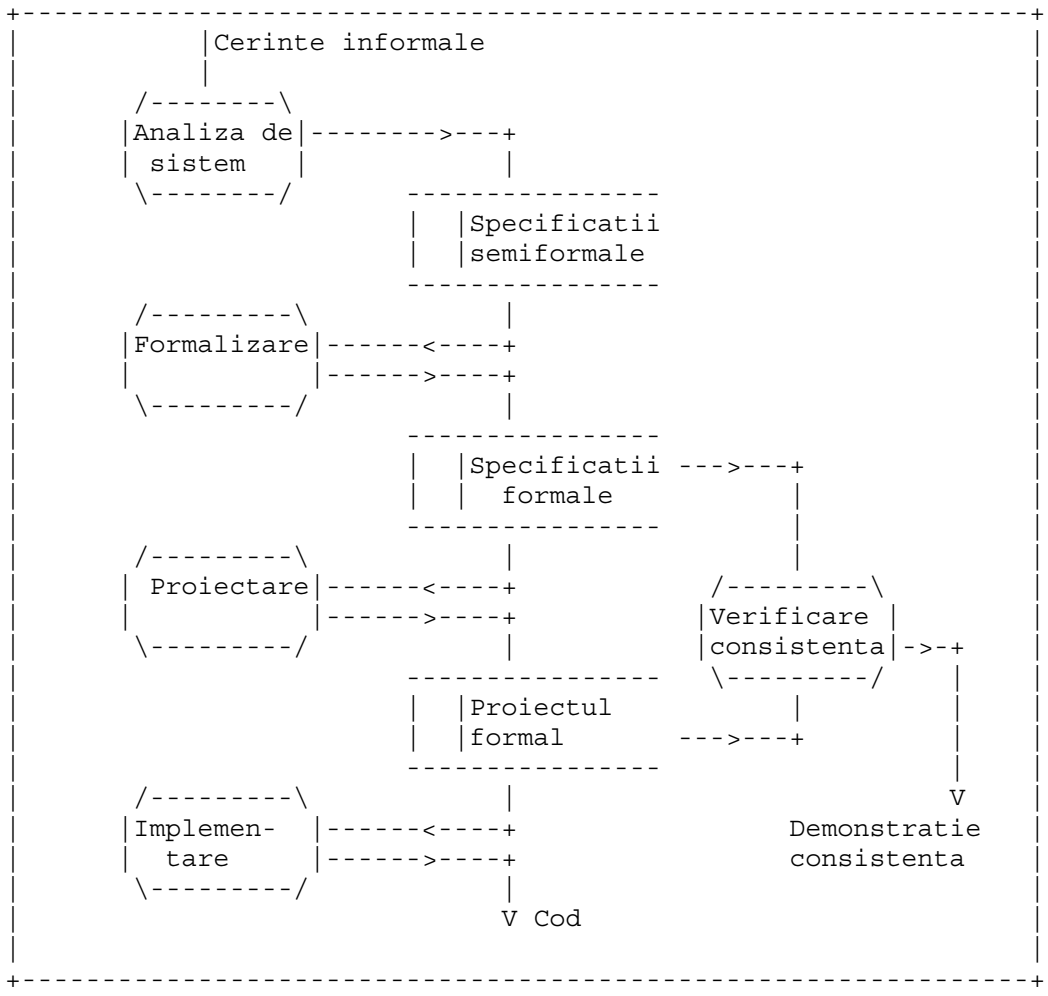


Fig. 1. Etapele dezvoltarii unui sistem informatic (SI)

Pasul al treilea, proiectarea, descris in cap.5., arata cum se implementeaza o tranzactie AF prin intermediul unei multimi de automate finite.
O tranzactie AF este descompusa in mai multe subtranzactii simple, continand elementele

tranzactiei initiale la care se adauga functii "administrative" pentru conectarea subtranzactiilor. Fiecare subtranzactie este implementata printr-un AF.

Verificarea consistentei (pasul patru) intre o tranzactie AF si implementarea sa prin AF este facuta folosind asertiunile pasilor precedenti si un set de teoreme dintre care cele mai importante sunt prezentate in cap.6.

3. Analiza de sistem folosind metoda ADISSA

Metoda ADISSA este o rafinare a metodei SSA si utilizeaza diagrame de flux de data (DFD) nu numai pentru analiza de sistem ci si pentru proiectarea acestuia.

Rezultatele principale ale aplicarii ei sunt
1. Interfata utilizator, formata dintr-un arbore de meniuri care poate fi considerat ca fiind arhitectura externa a sistemului.
2. Arhitectura interna, formata dintr-un set de tranzactii activate ca urmare a diferitelor evenimente sau a unor cereri ale utilizatorilor.

Pe linga acestea, metoda produce si schema bazei de date si schema de intrari iesiri ale SI.

Pasii principali ai aplicarii ADISSA sunt:
1. Analiza functionala si constructia DFD
2. Identificarea tranzactiilor
3. Proiectarea arborelui de meniuri
4. Descrierea structurala a tranzactiilor
5. Proiectarea schemei bazei de date
6. Proiectarea schemei de intrari - iesiri

si sunt obiectul subcapitolelor urmatoare.

3.1. Analiza functionala si constructia DFD

Analiza functionala este bazata pe folosirea unei ierarhii de diagrame de flux de date (DFD). Acestea definesc si descriu:
a. Functiile (prelucrarile) efectuate de sistem
b. Datele memorate care formeaza suportul permanent de date al sistemului.
c. Entitatile externe care se interfateaza cu sistemul
d. Fluxurile de date (FD) care conecteaza elementele de mai sus.

Acestea sunt cele patru categorii de elemente care formeaza o DFD. Prezentam caracteristicile lor si reprezentarea grafica folosita pentru acestea, urmate de definirea conceptului de tranzactie (in sens ADISSA):

Functiile specifica prelucrari efectuate in sistem. Ele sunt de doua feluri: functiile generale sunt acelea care pot fi detaliate ulterior prin DFD asociate lor; functiile elementare specifica prelucrari elementare care nu se pot detalia prin DFD.
Reprezentarea grafica normala pentru functii este cercul: dublu pentru functiile generale si simplu pentru cele elementare.
Pe parcursul prezentarii de fata, in interiorul ovalului unei functii acestora este inclus un identificator al functiei (o secventa de numere care deriva din ierarhia de DFD), ele fiind explicitate in legenda asociata figurilor. Codul lor este de tip FGx sau FEx.

Entitatile externe (numite in continuare entitati) specifica elementele externe cu care se

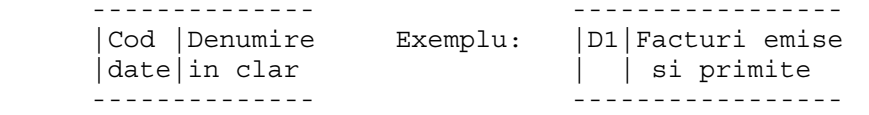
interfateaza sistemul informatic si care reprezinta declansatori si terminatori ai tranzactiilor. Entitatile sunt de trei feluri:

a. Entitate utilizator (EU) specifica interfata SI cu utilizatorii (umani) ai acestuia. Ele modeleaza tipuri de utilizatori (analog cu entitatile din modelul ELE, descris in
partea a doua a cursului).
Sunt atit declansatori de tranzactii (cand utilizatorul lanseaza o optiune a sistemului de meniuri al SI) cit si terminatori de tranzactii (cand datele rezultate din prelucrari sunt destinate prezentarii catre aceea categorie de utilizatori).
Reprezentarea grafica va fi un dreptunghi in care este scris codul acelei entitati (ex. EU3), explicitarea lor facandu-se in legenda atasata figurii. Ele sunt prezentate in exteriorul casetei ce inconjoara portiunea DFD care modeleaza prelucrarile din SI (deci nu se confunda cu functiile, care sunt in interiorul acesteia si nu au in codificare sirul 'EU').

b. Entitate timp (ET) modeleaza activarea unor tranzactii la momente de timp prestabilite (exemplu: sfirsit de zi, sfirsit de luna, sfirsit de an, etc).
Reprezentarea grafica va fi un triunghi in care este inscris codul entitatii (ex. ET sau ET1).

c. Entitate timp real (ETR) modeleaza activarea unor tranzactii de catre factori externi, diferiti de utilizatorul uman, conectati la acel SI (de exemplu diverse procese desfasurate in timp real care se interfateaza cu SI).
Reprezentarea grafica este similara cu cea a ET.

Datele memorate (DM) modeleaza partea "statica" a SI, suportul de date care este actualizat sau consultat de catre functiile prezente in DFD. Fiecare astfel de element al diagramei se poate implementa conceptual si fizic prin una sau mai multe relatii (fisiere de date) utilizate de SI. Reprezentarea grafica este urmatoarea:



Din motive de prezentare, in cele ce urmeaza figurile vor contine doar codul datelor, denumirea in clar fiind in legenda figurii.

Fluxurile de date modeleaza circulatia datelor intre entitati, functii si date memorate. Ele se reprezinta grafic prin arce orientate in sensul de circulatie al datelor. Din motive de prezentare, etichetarea acestor arce se va face printr-ul cod de FD care va fi explicitat in legenda atasata figurii.

Observatie: Daca intr-o DFD este prezenta de mai multe ori acelasi element (deci cu acelasi cod), el reprezinta aceeasi categorie de obiecte in ipostaze diferite.

Conceptul de tranzactie (ADISSA) este un element de baza al metodei si poate fi definit din doua puncte de vedere:

1. Din punct de vedere al utilizatorului, o tranzactie este o prelucrare independenta care:
a. are un inceput (start)
b. efectueaza o prelucrare definita de utilizator
c. are un sfirsit (stop, terminare) care lasa sistemul intr-o stare coerenta (in sensul in care este definita coerenta in teoria bazelor de date)

2. Din punct de vedere al DFD, o tranzactie consta in:
a. una sau mai multe entitati care declanseaza tranzactia

b. una sau mai multe functii elementare legate intre ele prin fluxuri de date

c. toate celelalte elemente (entitati si date memorate) conectate cu aceste functii si dintre care o parte formeaza terminatorii acelei tranzactii.

Pentru exemplificarea folosirii metodei ADISSA prezentam trei diagrame:

- prima (DFD-0) reprezinta diagrama sistemului informatic al unei firme de constructii si contine trei functii generale;
- urmatoarele doua diagrame (DFD-1 si DFD-2) reprezinta detalierea sub forma de DFD a primelor doua functii generale din DFD-0.

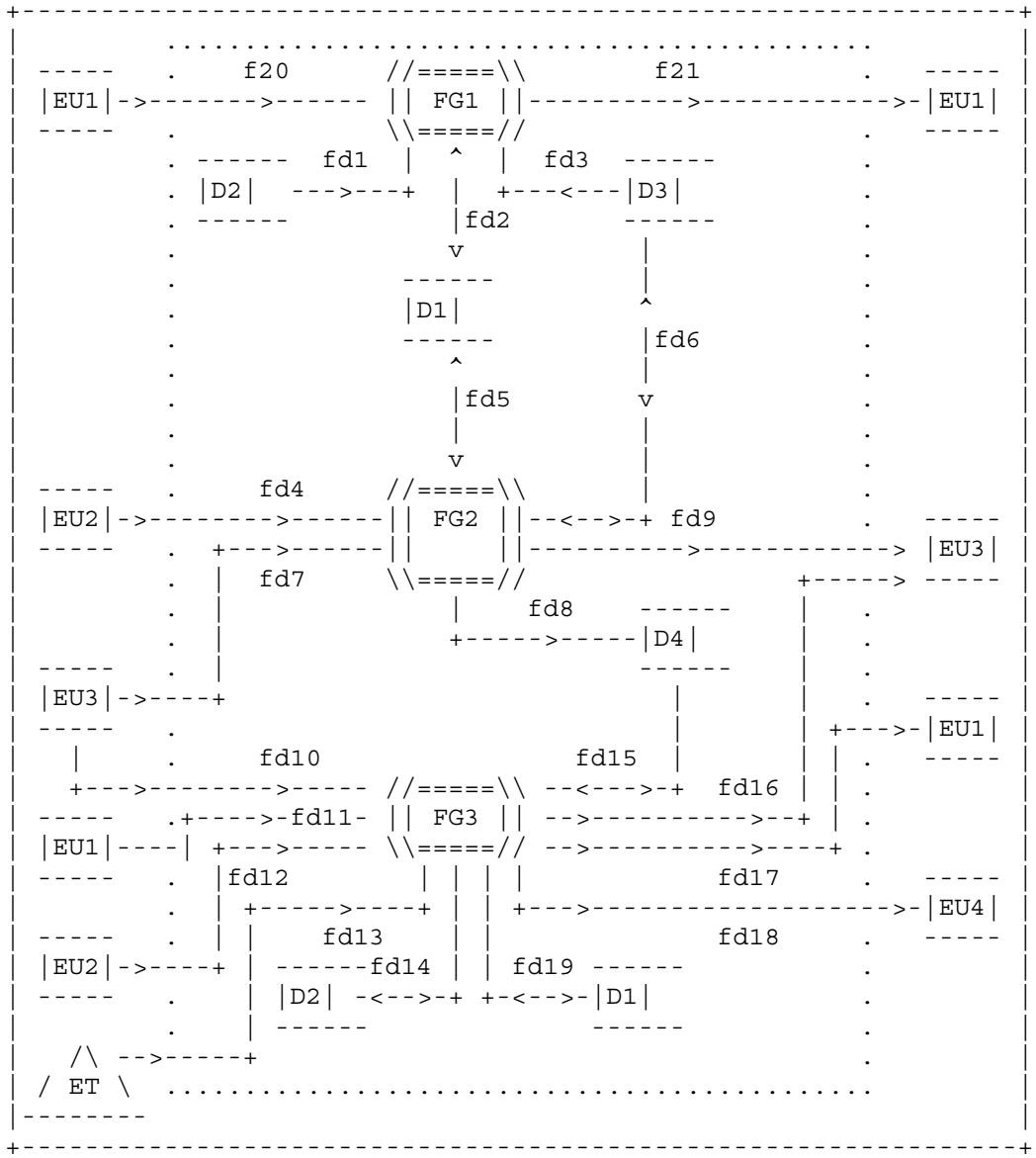


Fig. 2. DFD-0 Sistem informatic al unei firme de constructii

LEGENDA

Entitati:
EU1: Furnizori
EU2: Constructori
EU3: Clienti
EU4: Contabili
ET: Calendar de evenimente

Functii:
FG1: Aprovizionare de la furnizori
FG2: Cereri de lucrari si ordine de lucrari
FG3: Inregistrari contabile si rapoarte

Date memorate:
D1: Materiale in stoc
D2: Conturi furnizori
D3: Dereri de lucrari si ordine de lucrari
D4: Conturi clienti

Fluxuri de date:

fd1, fd20: Facturi si receptii	fd11: Adrese de la furnizori
fd2: Stoc	fd12: Costuri
fd3: Ordine in lucru	fd13: Sfirsit de luna
fd4: Materiale necesare	fd14: Plati, balante/furnizori
fd5: Preturi mater.din stoc	fd15: Plati, balante/clienti
fd6: Acceptare/refuz	fd16: Receptii,adrese,facturi
fd7: Acceptare/refuz	fd17: Plati de efectuat
fd8: Detalii client, preturi	fd18: Rapoarte
fd9: Deviz propus (pt.acont)	fd19: Tranzactii
fd10: Incasari	fd21: Ordine de cumparare

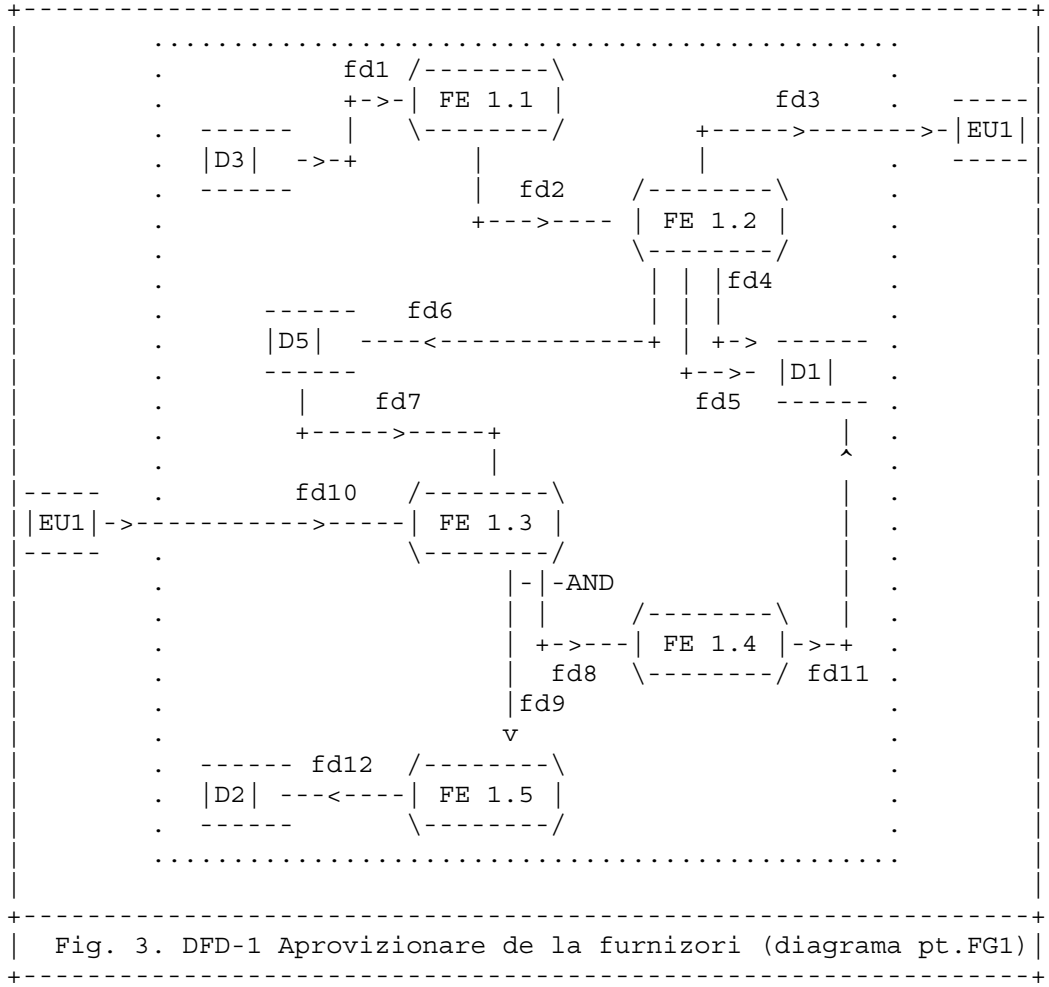


Fig. 3. DFD-1 Aprovizionare de la furnizori (diagrama pt.FG1)

LEGENDA

Entitati: cu aceeasi semnificatie ca la DFD-0

Date memorate:
D1, D2, D3:cu aceeasi semnificatie ca la DFD-0
D5: Ordine de cumparare

Functii:
FE1.1: Cautare cereri de lucrari
FE1.2: Pregatire lista de materiale necesare
FE1.3: Inregistrare factura primita (de la furnizor)
FE1.4: Actualizare stoc
FE1.5: Actualizare cont furnizor

Fluxuri de date:

fd1: Ordine in lucru	fd7: Ordine de cumparare
fd2: Detalii ordine active	fd8: Materiale cumparate
fd3: Ordine de cumparare	fd9: Facturi si receptii
fd4: Materiale comandate	fd10: Facturi si receptii
fd5: Stoc	fd11: Materiale cumparate
fd6: Ordine de cumparare	fd12: Facturi

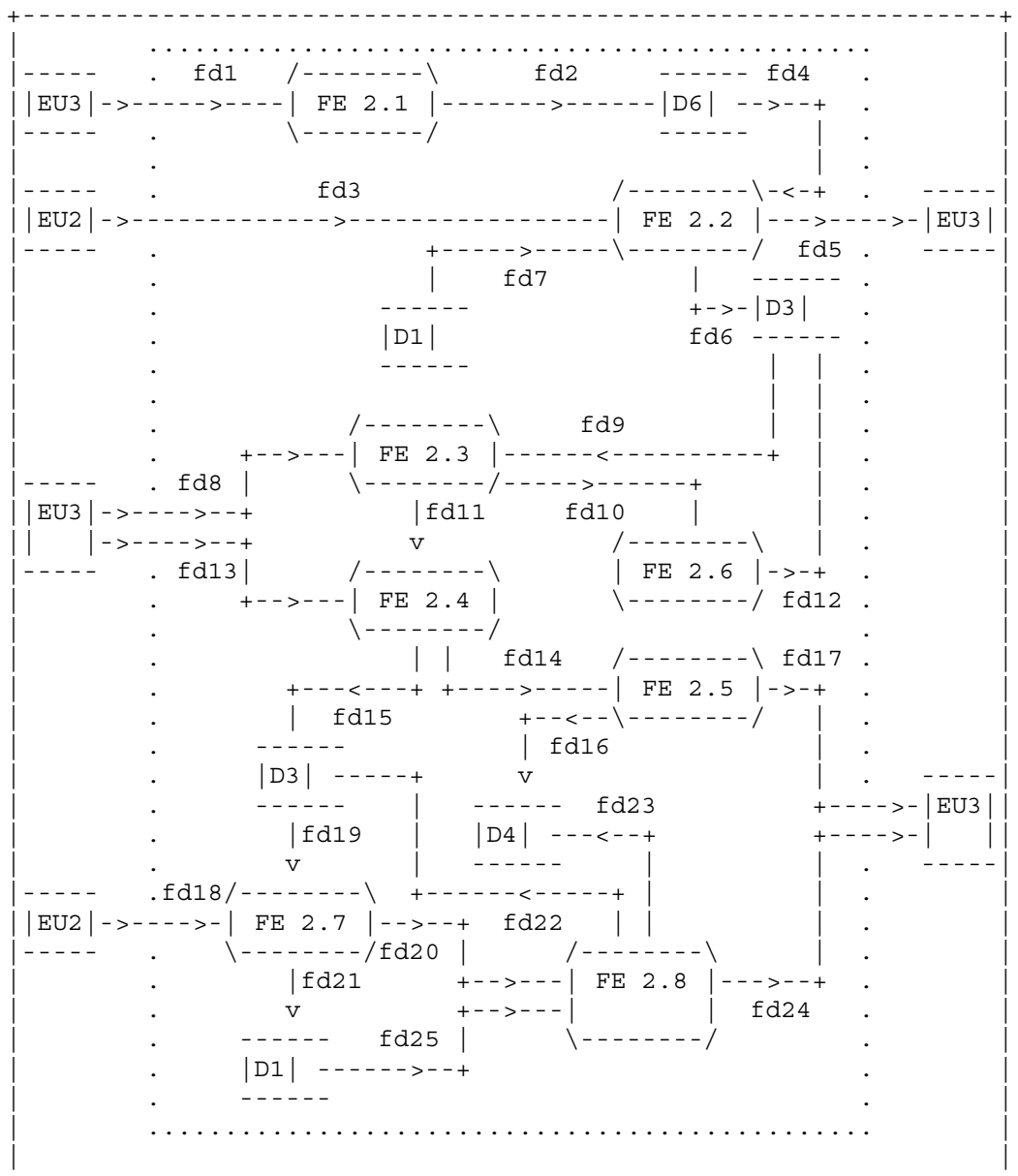


Fig. 3. DFD-2 Cereri de lucrari si ordine de lucrari (pt.FG2)

LEGENDA

Entitati: cu aceeasi semnificatie ca la DFD-0

Date memorate:
D1, D2, D3, D4:cu aceeasi semnificatie ca la DFD-0
D6: Cereri de lucrari

Functii:
FE2.1: Inregistrare cerere de lucrari
FE2.2: Pregatire deviz
FE2.3: Inregistrare raspuns client
FE2.4: Inregistrare prima plata
FE2.5: Deschidere cont client
FE2.6: Inchidere lucrare
FE2.7: Inregistrare consumuri
FE2.8: Actualizari si rapoarte

Fluxuri de date:
fd1: Cereri de lucrari
fd2: Cereri de lucrari
fd3: Necesar de materiale
fd4: Cereri de lucrari
fd13: Prima plata
fd14: Detalii plata
fd15: Acceptare
fd16: Detalii plata

```
fd5: Propuneri lucrari (deviz) fd17: Factura
fd6: Propuneri de lucrari      fd18: Consumuri
fd7: Materiale si preturi      fd19: Detalii lucrare
fd8: Acceptare/refuz           fd20: Materiale cons.+det.lucrare
fd9: Propuneri de lucrari      fd21: Materiale consumate
fd10: Refuz                    fd22: Sfirsit lucrare
fd11: Acceptare                fd23: Cost lucrare
fd12: Propuneri de sters       fd24: Rapoarte de lucru
                 fd25: Materiale si preturi
```

3.2. Identificarea tranzactiilor

Acest pas al metodei include identificarea tranzactiilor si a elementelor componente ale acestora, in principal a functiilor elementare si a datelor memorate continute. Se face prin inspectarea DFD si impartirea lor in tranzactii, conform definitiei din subcapitolul anterior.

Exemple:

In DFD-1 (fig.3) pot fi identificate doua tranzactii:
1. prima contine functiile 1.1 si 1.2, entitatea EU1 si datele memorate D1, D3 si D5.
2. a doua contine functiile 1.3, 1.4 si 1.5 si toate celelalte elemente conectate la acestea.

In DFD-2 (fig.4) identificam patru tranzactii, care contin:
1. functia 2.1
2. functia 2.2
3. functiile 2.3-2.6
4. functiile 2.7, 2.8

} + celelalte elemente conectate la aceste functii

DFD nu contin si semnificatia prelucrarilor efectuate de tranzactii. Aceasta se va inregistra separat. Deoarece DFD nu contin informatii despre fluxul controlului (ci doar al datelor), o tranzactie poate fi interpretata in mai multe feluri (are mai multe semnificatii posibile). Detectarea interpretarilor relevante (necesare utilizatorilor) este obiectul capitolului 4.

3.3. Proiectarea arborelui de meniuri

Scopul acestui pas este crearea unei ierarhii de meniuri care reprezinta, asa cum am mentionat, arhitectura esterna a SI.
Aceste meniuri se deduc din DFD. Meniul initial consta din functiile conectate direct la entitatile utilizator si contin cite o linie pentru fiecare astfel de conexiune.
Optiunile ("liniile") din fiecare meniu sunt de doua feluri:

a. linie terminala daca conexiunea este spre o functie elementara. Selectarea ei din meniu duce la activarea acestei functii. Acest tip de linii sunt frunzele arborelui de meniuri.

b. linie de selectie daca conexiunea este spre o functie generala. Selectarea ei duce la afisarea submeniului asociat acelei functii generale (dedus din DFD al acelei FG). Acest tip de linii sunt nodurile intermediare ale arborelui de meniuri.

Rafinarea arborelui de meniuri se face astfel incit fiecare linie terminala sa poata fi atinsa printr-o cale care porneste din meniul initial si selectarea ei duce la declansarea unei tranzactii.

3.4. Descrierea structurala a tranzactiilor

In metoda ADISSA, aceasta descriere se face la doua nivele:

1. La nivelul superior, fiecare tranzactie este descrisa specificand:
   - elementele care compun tranzactia (functii, entitati, date memorate si fluxurile de date care le conecteaza)
   - fluxul controlului, care poate fi opus fluxului datelor.

Specificarea fluxului de control se face in limbaj natural (vom vedea in capitolul 4 ca in momentul transformarii in tranzactii AF fluxul controlului se va putea specifica pe diagrama)

2. La nivelul inferior sunt detaliate componentele tranzactiei intr-o forma structurata (de exemplu functiile se pot detalia folosind pseudocodul)

Observatie: descrierea unei tranzactii (asa cum este definita in acest subcapitol) este exprimarea uneia sau mai multor interpretari valide ale semanticii sale. Aceasta descriere nu este suficient de formala pentru o transformare automata (de altfel am mentionat ca rezultatul analizei functionale sunt niste specificatii semiformale)

3.5. Proiectarea schemei bazei de date

Se poate folosi orice metoda valida, inclusiv metoda bazata pe modelul entitate asociere extins (ELE) prezentat anterior in cadrul cursului. Rezultatul acestui pas este schema bazei de date in forma normala dorita. Nu insistam asupra acestui pas, fiind tratat in extenso anterior.

3.6. Proiectarea schemei de intrari - iesiri

Acest pas consta din specificarea tipului si dispozitivului pentru fiecare intrare si iesire in/din fiecare tranzactie.
Rezultatul acestui pas contine in principal:

a. machetele ecranelor de introducere a datelor
b. machetele ecranelor de prezentare a datelor (vizualizare)
c. machetele rapoartelor produse de catre SI.

Toate acestea formeaza ceea ce se numeste schema de intrari-iesiri a sistemului informatic.

4. Formalizarea transformarii tranzactiilor

Aceasta parte contine:

- problema interpretarilor multiple ale unei tranzactii
- definirea tranzactiilor AF
- reguli de transformare a tranzactiilor ADISSA in tranzactii AF

4.1. Interpretarile multiple ale unei tranzactii

Unul din lipsurile majore ale modelului ADISSA este lipsa posibilitatii indicarii explicite a fluxului controlului, deci a ordinii in care sunt activate elementele componente ale unei tranzactii. Fluxurile de date (FD) specifica circulatia datelor dar ele pot fi si semnale care duc la declansarea unor functii.
DFD, asa cum au fost descrise pina acum, nu dau nici o indicatie privind care FD sunt si declansatori si, in cazul ca sunt, in ce directie. Din aceasta cauza, pentru fiecare FD pot fi date mai multe interpretari posibile.

Utilizatorul este cel care trebuie sa indice care FD sunt declansatori si in ce directie. Pentru aceasta, el trebuie sa descrie ce prelucrare executa fiecare tranzactie. Descrierea se face desemnand:

a. un element de pornire (start)
b. unul sau mai multe elemente de oprire (stop, terminare)
c. fluxul controlului care uneste elementul de start cu elementele de stop, trecand prin diverse functii elementare.

Fiecare astfel de descriere a unei prelucrari efectuate de o tranzactie se numeste interpretare relevanta a acelei tranzactii.
De mentionat ca pentru fiecare tranzactie ADISSA sunt posibile mai multe interpretari. Doar o parte a acestora au sens si doar o parte a celor cu sens sunt relevante (deci necesare utilizatorului si cerute de catre acesta). Doar acestea din urma vor fi cele descrise la acest pas si implementate mai departe.

Exemplu:

Tranzactia continand functiile 2.7. si 2.8. din DFD-2 (fig.4) are trei interpretari cu sens:

Interpretarea nr. 1: Constructorul (EU2) introduce datele despre materialele consumate, functia FE 2.7 actualizeaza D1 (materiale in stoc) in functie de datele introduse si de detaliile lucrarii prelevate din D3 (propuneri de lucrari si ordine de lucrari). In continuare este activata functia FE 2.8 care actualizeaza D4 (conturi clienti) si tipareste un raport care va fi trimis clientului (EU3).

Interpretarea nr. 2: Clientul (EU3) cere raportul de lucru. Pentru aceasta el activeaza functia FE 2.8 care, pentru a putea tipari raportul, are nevoie de datele pe care i le furnizeaza functia FE 2.7. Deci FE 2.8 va activa FE 2.7 care livreaza datele (materiale consumate si detalii lucrare - manopera, etc) dupa care tipareste raportul de lucru.

Interpretarea nr. 3: Constructorul (EU2) introduce datele privind consumurile de materiale, activand FE 2.7. Aceasta actualizeaza D1 si cu aceasta prelucrarea ia sfirsit.

Din aceste trei interpretari, utilizatorul decide de exemplu ca doar primele doua sunt reprezinta prelucrari pe care le doreste implementate de catre SI. Deci doar aceste doua interpretari sunt relevante in sensul definitiei de mai sus si doar ele vor fi formalizate utilizand tranzactii AF.

4.2. Tranzactii AF

O tranzactie AF este o descriere formala a fluxului de control pentru o tranzactie ADISSA. Functiile elementare, entitatile si datele memorate, asa cum au fost definite in ADISSA, sunt componente si ale tranzactiei AF si vor fi numite in continuare obiecte.

Ambiguitatea privind fluxul controlului este ridicata introducand conceptul de ***flux de date si control (FDC).***
Un FDC este un FD care in acelasi timp este si declansator al unei functii elementare sau entitati sau care indica sfirsitul unei tranzactii.

Un FDC este orientat de la obiectul declansator la obiectul declansat. Acele FD care nu sunt si de control raman in continuare fluxuri de date, semnificand doar circulatia datelor si nu si a controlului.

Deci o tranzactie ADISSA contine:

1. functii elementare 2. entitati externe 3. date memorate	elemente comune cu tranzactiile ADISSA
4. fluxuri de date 5. fluxuri de date si control	in acceptiunea definitiei de mai sus

4.3. Sintaxa tranzactiilor AF

Inainte de a defini ce este o tranzactie AF corecta (deci sintaxa unei tranzactii AF) trebuie sa definim doua din caracteristicile unei astfel de tranzactii:

a. ***Multimea de start*** (pornire, declansare) este formata din entitatile care actioneaza ca decllansatori pentru acea tranzactie.

b. ***Multimea de stop*** (oprire, terminare) este formata din toate entitatile si datele memorate care au cel putin un FDC care intra si nici unul care iese.

Definitia tranzactiei AF:

O multime de obiecte, FD si FDC formeaza o tranzactie AF daca este construita dintr-o tranzactie valida ADISSA si satisface urmatoarele restrictii:

1. Functiile si entitatile sunt legate prin FDC. Daca o functie sau entitate nu este conectata prin FDC de restul obiectelor, ea este inutila.
2. Orice functie poate fi atinsa prin cel putin o cale orientata formata din FDC, cale care porneste dintr-un element al multimii de start. In caz contrar, acea functie nu va fi niciodata activata in caz de executie a tranzactiei.
3. Orice cale formata din FDC trebuie sa se termine intr-un element al multimii de stop, altfel tranzactia are un ciclu infinit. De remarcat ca ciclurile sunt permise in masura in care ele demonstreaza ca au o cale de terminare a lor (in genul punctului fix in cazul recursivitatii).
4. O entitate are cel mult un FDC care iese. Daca are mai multe, ar exista o ambiguitate privind care dintre aceste FDC va fi activat de catre ea.
5. FDC si FD nu pot lega mai mult de doua obiecte (fluxurile nu se divid in sine).
6. Doar FDC care ies dintr-o functie pot participa la un AND logic. In acest caz ele sunt activate impreuna de catre functia din care pleaca. Acesta este singurul tip de AND intre FDC care exista in cazul unei tranzactii AF,

O tranzactie AF poate fi reprezentata printr-o diagrama formata din aceeasi simboli ca in cazul diagramelor ADISSA. In cele ce urmeaza vom numerota elementele unei diagrame. Daca doua elemente au acelasi numar, inseamna ca ele reprezinta unul si acelasi obiect in ipostaze diferite (se va folosi aceasta tehnica pentru simplitatea diagramelor). Obiectele vor avea o simbolizare care incepe cu litera "o" (de exemplu o1, o2, o3, ...), FDC cu "fdc" iar FD cu "fd".

4.4. Semantica executiei unei tranzactii AF

Acest subcapitol defineste semnificatia termenului de "executie a unei tranzactii AF". Accentul cade pe partea de control a unei astfel de tranzactii.

Pentru inceput, vom face doua asertiuni fundamentale:

Asertiunea 1: Pentru fiecare functie elementara exista o *implementare exacta* a specificatiilor acelei functii.
Asertiunea 2: Daca o entitate are un FDC care iese, ea va emite un semnal se activare in timp finit.

In plus, se va presupune (o restriuctie suplimentara deci) ca o tranzactie nu poate fi declansata de mai multe semnale simultan (deci eliminarea concurentei in declansarea tranzactiilor).

Prezentam in cele ce urmeaza o serie de definitii care duc la o definire formala a executiei unei tranzactii AF.

Definitia 1. *Executia unui obiect* = Este in functie de tipul obiectului:
a. ***Executia unei functii***: executia implementarii asociate acesteia (conform primei asertiuni, exista o implementare exacta pentru fiecare functie).
b. ***Executia unei entitati***: consta in emiterea unui semnal pe FDC care iese (daca are) sau emiterea unui semnal vid/sfirsit de tranzactie (daca nu are FDC care iese).
c. ***Executia unor date memorate***: emiterea unui semnal vid. Practic executia unor date memorate inseamna actualizarea sau citirea acestora - deci un accces la date - dupa care se emite un semnal vid - deci practic fluxul se intrerupe. Semnalul vid este introdus si in acest caz pentru a respecta principiul ca executia fiecarui obiect duce la emiterea unui semnal, chiar si vid.

Definitia 2. *Semantica FDC* = Daca mai multe FDC care ies dintr-o functie nu sunt legate prin AND logic, functia activeaza doar pe unul dintre ele (deci se considera automat SAU EXCLUSIV - XOR).
Daca sunt legate cu AND, functia le activeaza concurent pe toate dupa terminarea executiei sale.

Definitia 3. *Eveniment* = Un eveniment este executia unui obiect al unei tranzactii AF, impreuna cu intrarile si iesirile sale.
Deci executia unui obiect (sau eveniment) este un triplet:

(i, A, r)

unde:

- i = FDC care intra in acel obiect
- A = executia acelui obiect (actiune)
- r = multimea FDC care ies din obiect si sunt activate dupa executia sa in cazul intrarii i si executiei actiunii A.

Deci, daca A activeaza un singur FDC, r va contine un singur element. In cazul in care A activeaza mai multe FDC conectate printr-un AND, r va contine mai multe elemente.

Definitia 3. *Eveniment initial* = Se spune ca E = (i, A, r) este un eveniment initial daca:
- obiectul corespunzator lui A este un element din multimea de start (deci o entitate).
- i simbolizeaza activarea acestei entitati de catre o cauza externa tranzactiei (deci i este un fel de pseudo-FDC. Multimea pseudo-FDC se va nota in cele ce urmeaza cu λ).

Definitia 4. *Activarea unui eveniment* = Se spune ca evenimentul E1 = (i1, A1, r1) este activat de evenimentul E = (i, A, r) (notatie: $E \rightarrow E1$) daca si numai daca i1 apartine lui r. In cazul unui eveniment initial, se spune ca acesta este activat de un eveniment extern.
Ca un corolar, se spune ca evenimentul E = (i, A, r) activeaza evenimentele E1 = (i1, A1, r1), ..., En = (in, An, rn) daca si numai daca $r = i1 \cup i2 \cup ... \cup in$.

Definitia 5. *Executia tranzactiei* = Un lant finit de evenimente (E1, E2, ...) activate de un eveniment initial (E1→E2→E3..., unde E1 este un eveniment initial) se numeste o executie a tranzactiei.

Definitia 6. *Implementarea tranzactiei* = Multimea tuturor executiilor unei tranzactii.

4.5. Transformarea tranzactiilor ADISSA in tranzactii AF

Aceasta transformare se face in doi pasi:
1. Definirea interpretarilor relevante
2. Ajustari pentru respectarea sintaxei tranzactiilor AF descrisi in continuare.

4.5.1. Definirea interpretarilor relevante

Pentru fiecare interpretare relevanta a unei tranzactii, analistul construieste o diagrama continand elementele diagramei tranzactiei ADISSA care iau parte la acea interpretare. Pentru aceasta, el trebuie sa executa urmatoarele operatii:

1. Definirea multimii de start.
2. Definirea multimii de stop.
3. Marcarea explicita a relatiei dintre fluxuri (AND, OR, XOR).
4. Specificarea tipului fiecarui flux (FD sau FDC)
5. Introducerea de FDC aditionale cand fluxul controlului este opus fluxului datelor.

Pentru realizarea acestor operatii, se folosesc regulile urmatoare:

Regula 1. ***Multimea de start:*** Daca o entitate declanseaza tranzactia, ea apartine multimii de start.

Regula 2. ***Multimea de stop:*** Daca o tranzactie se termina prin activarea unei entitati sau a unor date memorate, entitatea sau datele memorate apartin multimii de stop.

Regula 3. ***Relatia dintre fluxuri:***
a. Daca, cf. afirmatiilor utilizatorului, mai multe fluxuri care ies dintr-un obiect sunt intotdeauna activate impreuna, ele vor fi marcate ca legate printr-un AND.
b. Daca ele sunt activate impreuna doar in unele executii, vor fi marcate ca legate prin OR.
c. Daca in toate executiile doar unul este activat (nu neaparat acelasi, bineinteles), ele vor fi marcate ca legate prin XOR.

Regula 4. ***Clasificarea in FD si FDC:***
a. Orice flux intre doua functii sau intre o functie si o entitate este FDC.
b. Daca un flux este de la o functie la date memorate, atunci:
- daca nu e implicata in nici un AND, este FDC.
- daca este implicata intr-un AND doar cu fluxuri care pleaca spre date memorate, este FDC
c. Toate celelalte fluxuri sunt FD.

Observatie:
Daca dupa aceste transformari un AND contine doar FD, este sters marcajul acestei legaturi (nu si fluxurile!); in cazul in care contine si FD si FDC, AND-ul este modificat pentru a lega doar FDC (din el se elimima FD).
Ca rezultat, AND-urile vor contine doar FDC.

Regula 5. ***FDC aditionale:*** Daca, cf.afirmatiilor utilizatorului, fluxul controlului este opus

fluxului de date, se adauga un FDC in sensul fluxului de control.

4.5.2. Ajustari pentru respectarea sintaxei tranzactiilor AF

Urmare a aplicarii regulilor de mai sus, se obtine o diagrama pentru fiecare interpretare relevanta a tranzactiei ADISSA. In continuare trebuiesc efectuate doua activitati:

a. Punerea de acord cu sintaza tranzactiilor AF (definita anterior)
b. Fuzionarea acestor diagrame intr-una singura

Aceste activitati se efectueaza folosind regulile urmatoare:

Regula 6. ***Multiplicare entitati:*** Daca o entitate are mai mult de un FDC care iese, este multiplicata in tot atitea copii cite FDC ies din ea, fiecare mostenind unul dintre aceste fluxuri. In acest fel, cerinta de sintaxa conform careia o entitate poate avea doar un singur FDC care iese este indeplinita.
De asemenea, daca are cel putin un FDC care iese si este si in multimea de stop, este multiplicata cu inca un exemplar suplimentar a.i. o copie a sa sa nu aiba nici un FDC care iese.
In ambele cazuri, copiile pot mosteni toate FDC care intra si numele original.

Regula 7. ***Fuzionare:*** Fuzionarea diagramelor (cite una pentru fiecare interpretare relevanta) intr-una singura se face astfel:
7.1. *Datele memorate*: raman intr-un singur exemplar, colectand toate fluxurile (care intra si ies).
7.2. *Functiile*: raman intr-un singur exemplar, colectand toate fluxurile (care intra si ies).
7.3. *Entitatile*: vor fi deocamdata copiate din fiecare interpretare.
7.4. *Fluxurile*: se copiaza cite un exemplar din fiecare flux.
Daca in cel putin o interpretare este FDC, acesta va fi FDC si in diagrama fuzionata; celelalte raman FD. Fiecare flux va fi etichetat cu eticheta sa originala (toate copiile trebuie sa aiba aceeasi eticheta in diversele diagrame de interpretari relevante, inclusiv cele aditionale).

Regula 8. ***Eliminare OR:*** Eliminarea lui OR se face prin descompunerea si multiplicarea fluxurilor in numarul de exemplare necesar expresiei rezultate.

Exemplu: daca fluxurile a, b, c sunt marcate ca formand un OR si:
- in unele executii sunt activate toate trei
- in alte executii doar b si c
- in toate celelalte executii doar b

rezulta expresia (a AND b AND c) XOR (b AND c) XOR b, ceea ce inseamna ca avem nevoie de 1 copie a lui a, 3 copii ale lui b si doua copii ale lui c. OR intre cele trei fluxuri va fi deci inlocuit cu un AND intre 3 fluxuri, un AND intre alte doua fluxuri si un flux neconectat logic cu altele (obs.: XOR este implicit si nu se marcheaza pe diagrama)

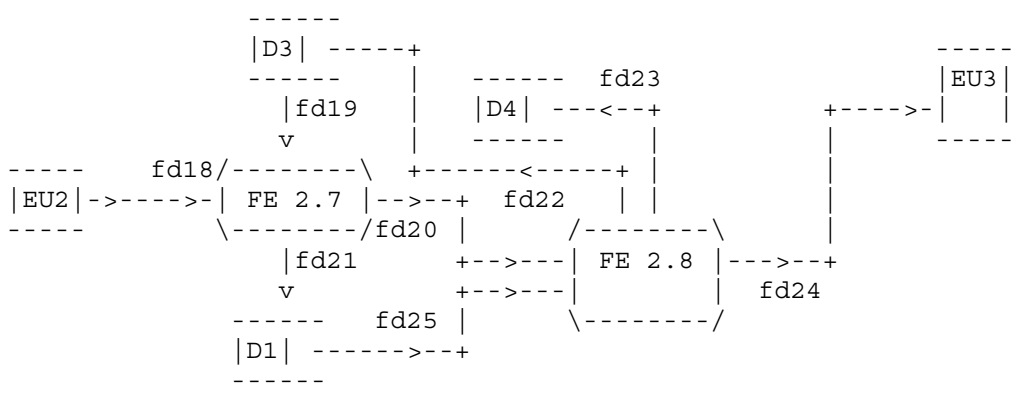
Regula 9. ***Fuziune entitati:*** Aparitiile multiple ale aceleiasi entitati (care pot fi rezultate si din multiplicare, vezi regula 6) pot fuziona daca au aceleasi FDC care ies, in care caz colecteaza toate fluxurile care intra. Daca nu au aceleasi FDC care ies, raman distincte si se redenumesc, astfel incit sa nu avem doua entitati cu acelasi nume.

Regula 10. ***Specificare functii:*** Se face specificarea prelucrarilor efectuate de fiecare functie elementara (in orice mod: pseudocod, limbaj natural, etc). Aceasta regula nu are nici o influenta in prezentarea metodologiei in cele ce urmeaza. Ea se aplica doar cand aceasta se aplica pe un caz real.

4.5.3. Exemple

4.5.3.1. Exemple de aplicare a regulilor 1 - 5

Tranzactia continand functiile 2.7. si 2.8. din DFD-2 (fig.4) are trei interpretari cu sens, enuntate in capitolul 4.1.




```
FE2.7: Inregistrare consumuri
FE2.8: Actualizari si rapoarte

Fluxuri de date
fd18: Consumuri
fd19: Detalii lucrare
fd20: Materiale cons.+detalii lucrare
fd21: Materiale consumate
fd22: Sfirsit lucrare
fd23: Cost lucrare
fd24: Rapoarte de lucru
fd25: Materiale si preturi
```

Interpretarea nr. 1: Constructorul (EU2) introduce datele despre materialele consumate, functia FE 2.7 actualizeaza D1 (materiale in stoc) in functie de datele introduse si de detaliile lucrarii prelevate din D3 (propuneri de lucrari si ordine de lucrari). In continuare este activata functia FE 2.8 care actualizeaza D4 (conturi clienti) si tipareste un raport care va fi trimis clientului (EU3).

Regula 1: Multimea de start: { EU2 }

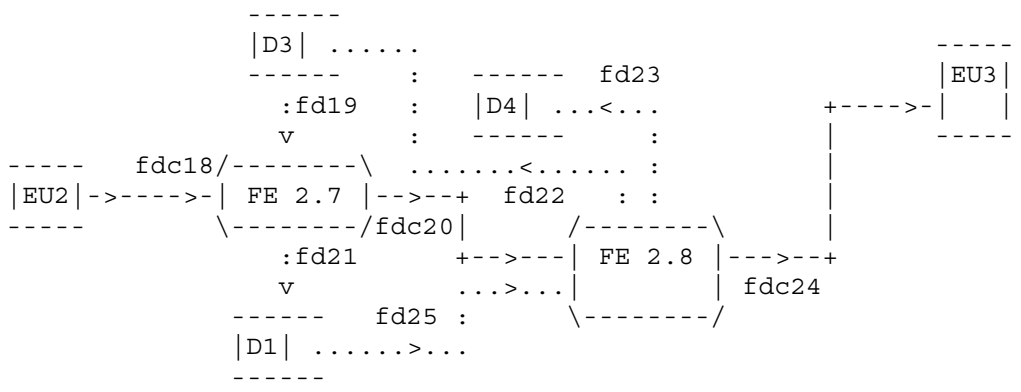
Regula 2: Multimea de stop: { EU3 }

Regula 3: Fluxurile fd20 si fd21 care ies din FE2.7 sunt intotdeauna activate impreuna => se marcheaza cu AND. Analog fluxurile fd22, fd23, fd24 care ies din FE2.8.

Regula 4: FDC vor fi fd18, fd20 si fd 24. Nici un fd de la functii spre date memorate nu e convertit in FDC, deoarece sunt in AND impreuna cu fluxuri care merg la entitati sau functii.
Deoarece ambele AND-uri contin doar un FDC si in rest FD => prin restringerea AND-urilor conform observatiei, pentru a contine doar FDC, aceste AND-uri dispar (ramane cite un singur FDC in componenta lor). Rezulta deci ca in final aceasta diagrama nu contine AND.

Regula 5: Nu este cazul pentru aceasta interpretare.

Rezultat:
Am simbolizat cu linie punctata FD si cu linie continua FDC.



Interpretarea nr. 2: Clientul (EU3) cere raportul de lucru. Pentru aceasta el activeaza functia FE 2.8 care, pentru a putea tipari raportul, are nevoie de datele pe care i le furnizeaza functia FE 2.7. Deci FE 2.8 va activa FE 2.7 care livreaza datele (materiale consumate si detalii lucrare - manopera, etc) dupa care tipareste raportul de lucru.

Regula 1: Multimea de start: { EU3 }

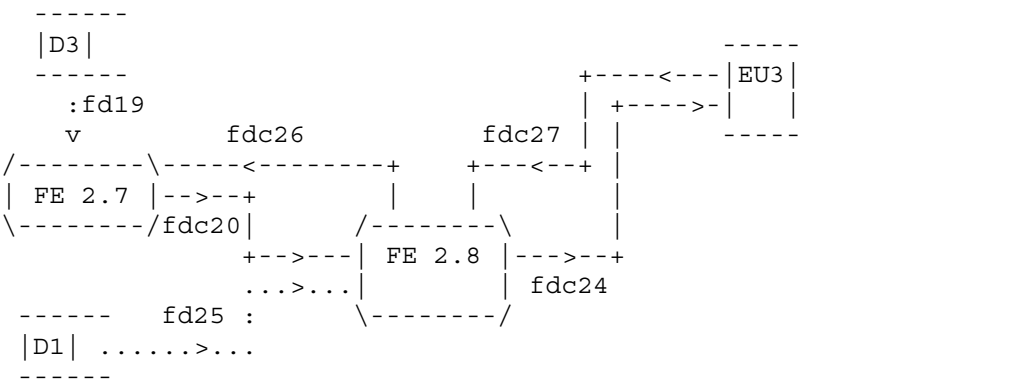
Regula 2: Multimea de stop: { EU3 }

Regula 3: In acest caz FE2.7 produce doar fd20 iar FE2.8 doar fd24. Celelalte fluxuri care ies din ele nu sunt necesare interpretarii. Rezulta ca regula 3 nu are obiect de aplicare in acest caz.

Regula 4: FDC vor fi doar fd20 si fd24.

Regula 5: Se introduce un FDC de la EU3 la FE2.8 ("Activeaza FE2.8-actualizari si rapoarte") si un FDC de la FE2.8 fa FE2.7 ("Activeaza FE2.7 - Acceptare consumuri") deoarece fluxul controlului este in aceasta interpretare invers celui de date, dupa cum se vede si din enuntul interpretarii.

Rezultat:
Am simbolizat cu linie punctata FD si cu linie continua FDC. Cele doua FDC adaugate cf. regulii 5 au fost notate fdc26 si fdc27.



Deoarece interpretarea a 3-a nu a fost considerata relevanta, ea nu se ia in consideratie mai departe.

4.5.3.2. Exemplu de aplicare a regulilor 6 - 9

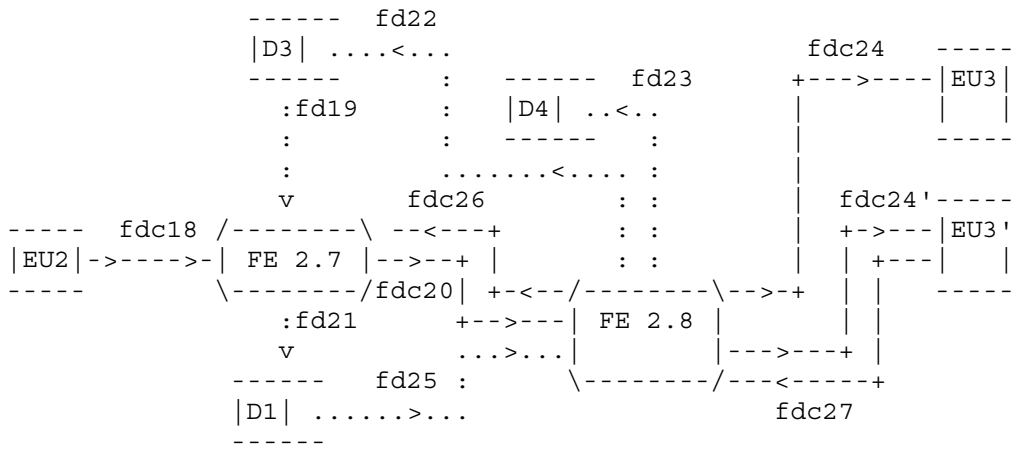
Regula 6. EU3 din interpretarea 2 este in multimea de stop si are un FDC care iese. Rezulta ca vom crea o copie a sa care are nici un FDC care iese, dar are aceleasi fluxuri care intra ca si originalul (unul singur, fdc24).

Regula 7. Vom fuziona diagramele. Diagrama obtinuta contine cite o copie din fiecare flux, functie si date memorate. Fluxurile care sunt FDC intr-una din interpretari vor fi trecute ca FDC, restul raman FD. Diagrama mai contine o copie din EU2 (din interpretarea 1) si trei copii ale EU3 (una din interpretarea 1 si doua din interpretarea 3 + regula 6).

Regula 8. Nu este cazul (Nu avem nici un OR)

Regula 9. Avem doua copii ale lui EU3 care au aceleasi fluxuri care ies (in speta nici unul), una din interpretarea 1 si cealalta creata prin regula 6. Ele vor fuziona si hotarim ca va pastra numele initial (EU3). Cealalta copie a lui EU3 (avand un flux care iese) o vom redenumi EU3'.

Rezultatul obtinut din aplicarea globala a regulilor 1-9 este deci:



Observam ca:
Multimea de start = { EU2, EU3' }
Multimea de stop = { EU3 }

5. Proiectarea tranzactiilor AF.

In cele ce urmeaza, *tranzactiile AF* vor fi numite pe scurt *tranzactii.*
Acest capitol cuprinde descrierea procesului de proiectare a unei tranzactii. In subcapitolul 5.1. sunt definite auomatele finite (AF) care vor fi folosite in continuare ca primitive in implementarea tranzactiilor.
Deoarece AF nu pot modela executia concurenta (sunt secventiale), in cap.4 s-a introdus restrictia ca o tranzactie nu poate fi declansata concurent de catre doua entitati externe. Totusi, existenta functiilor logice AND intre FDC in descrierea unei tranzactii duce la o concurenta conceptuala in interiorul sau.

Din aceasta cauza, prezentarea proiectarii unei tranzactii se va face in doi pasi:

- se prezinta la inceput rezolvarea problemei pentru tranzactii fara AND, numite si tranzactii secventiale, pentru care se arata cum se defineste un AF asociat (cap.5.2)

- se trateaza apoi cazul tranzactiilor care contin AND. Acestea se vor inlocui cu o multime de subtranzactii secventiale, interconectate prin "functii administrative". Pentru fiecare subtranzactie se va asocia apoi un AF. Tratarea acestui caz se face in 5.2-5.4.

5.1. Automate finite (AF)

AF folosite in acesta prezentare sunt cele clasice (se mai numesc si masini secventiale), avand actiuni (operatii) asociate tranzitiilor dintr-o stare in alta. Vom folosi automate deterministe, avand o stare initiala si o multime de stari finale. Operatiile asociate cu tranzitiile fac parte din definitia masinii si sunt initiate de tranzitie si nu de procese externe. Un AF nu este comun mai multor procese, fiecare corespunzand unei etape din executia tranzactiei (in cazul in care tranzactia se traduce prin mai multe AF).
Singura deosebire fata de modelul clasic este ca o parte a intrarilor sunt generate de masina (deci de operatiile asociate tranzitiilor).

Definitia formala a unui AF este urmatoarea (semnul de apartenenta a unui element la o multime s-a notat cu "<-", iar incluziunea de multimi cu "<" sau "<=")

Un **automat finit** este un sistem (S, E, A, T, s0, e0, F) unde:

1. S este multimea starilor, nevida si finita.
2. E este multimea intrarilor, finita (intrarea vida, notata ν, apartine lui E).
3. A este multimea actiunilor, finita. O actiune $a \in A$:
   - preia intrarea curenta si date din afara AF
   - le proceseaza
   - produce date catre exteriorul AF si intrarea urmatoare a masinii. Este posibil ca intrarea urmatoare sa fie ν (intrarea vida).
4. T este functia de tranzitie,
   $T : S \times E \rightarrow S \times A$
   Daca T(s, e) = (s', a) atunci fiind in starea s si citind intrarea e, automatul trece in starea s' si executa actiunea a. Daca $e = \nu$, masina se opreste.
5. $s0 \in S$ este starea initiala
6. $e0 \in E$ este intrarea initiala
7. F este multimea de stari finale, $F \subseteq S$.

Se spune ca un AF ***accepta*** secventa de intrare e0, e1, ..., en daca si numai daca:

- n este finit
- AF aflat initial in starea s0 si primind intrarea e0, face o succesiune de tranzitii (care genereaza intrarile e1, ..., en) si, in momentul citirii intrarii en, se opreste intr-o stare $s \in F$.

5.2. Implementarea tranzactiilor secventiale

O tranzactie secventiala este o tranzactie AF in care FDC nu sunt conectate prin vreun AND. AF asociat cu astfel de tranzactii se defineste in felul urmator (in cele ce urmeaza am folosit litere mari pentru obiecte, stari individuale, etc, in cazul cand ele erau indexate generic - de exemplu starea Si in loc de si - pentru a evita confuzia la citire):

1. Multimea de stari S:

a. Pentru fiecare obiect Oi care nu apartine multimii de start se asociaza o stare Si.
b. Starea s0 (initiala) se asociaza tuturor elementelor din multimea de start.

2. Multimea de intrari, E:

a. Pentru fiecare flux de date si control, FDCi, se asociaza o intrare Ei.
b. Intrarea e0 se asociaza lui λ (multimea pseudo-FDC descrisa la 4.4.)

3. Multimea de actiuni A:
a. Executia fiecarui obiect Oi (definita la 4.4.) are asociata o actiune Ai
b. Se adauga la aceasta o actiune a0: "determina care entitate din multimea de start este activata si executa actiunea asociata acesteia" (pentru cazul in care multimea de start contine mai multe elemente).

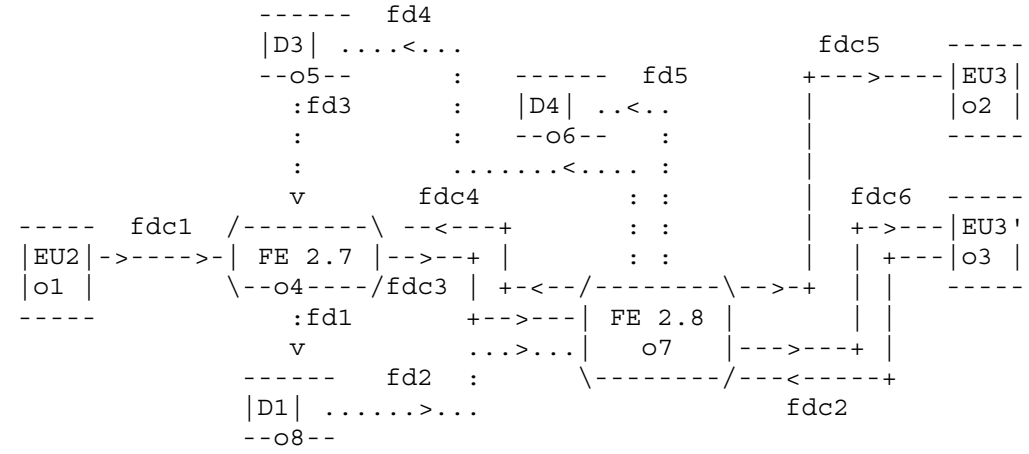
4. Functia de tranzitie, T:
a. Daca exista un FDC de la Oi la Oj, atunci
T(Si, Ei) = (Sj, Aj) unde
- Si este starea asociata lui Oi
- Sj este starea asociata lui Oj
- Ei este intrarea asociata FDC care leaga Oi de Oj
- Aj este executia lui Oj
b. T(s0, e0) = (s0, a0), cu a0 ca mai sus.

5. Starea initiala este s0, definita la 1.

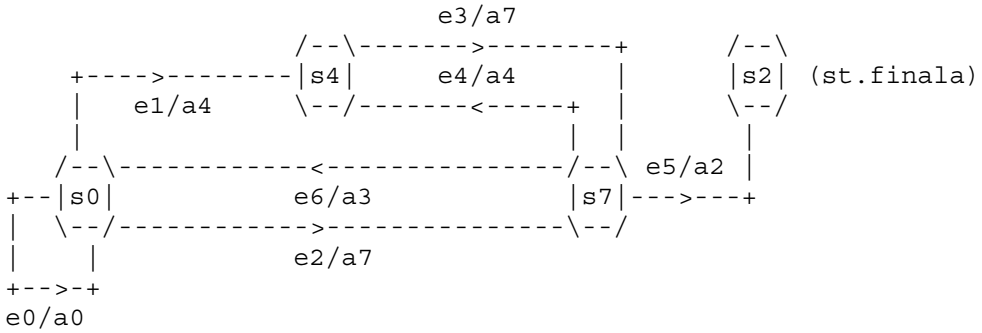
6. Intrarea initiala este e0, definita la 2.

7. Multimea de stari finale, F, contine starile asociate obiectelor din multimea de stop a tranzactiei.

Exemplu: Pentru tranzactia AF generata la 4.5.3., am facut o renumerotare a fluxurilor astfel: FDC si FD au primit numere distincte (incepand de la 1) iar obiectele (functii, entitati si date memorate) de asemenea. Ea arata deci in felul urmator:



Pentru aceasta tranzactie AF, diagrama automatului finit asociat (in care am notat starile cu noduri si tranzitiile cu arce etichetate cu intrarea si actiunea asociata ei) este urmatoarea:



Unde:

- s0 = starea asociata lui o1 si o3
- e0 = asociat lui λ (eveniment extern, preudo FDC)
- a0 = determina care dintre { o1, o3 } este seclansatorul si executa actiunea asociata acestuia.
- e1/a4 = corespunzator lui fdc1, intre o1 si o4; se executa a4, actiunea asociata lui o4 (functia elementara FE2.7).
- Analog: e2/a7, e3/a7, e4/a4, e5/a2, e6/a3 corespund FDC cu numerele 2, 3, 4, 5 si 6, executandu-se actiunile: a4 (fct.2.7), a7 (fct.2.8), a2 (stop) si a3 (entitatea EU3' - emitere semnal vid).

Observatie: din diagrama asociata s-au eliminat starile imposibil de atins (in cazul nostru cele corespunzatoare datelor memorate nelegate prin FDC de restul diagramei).

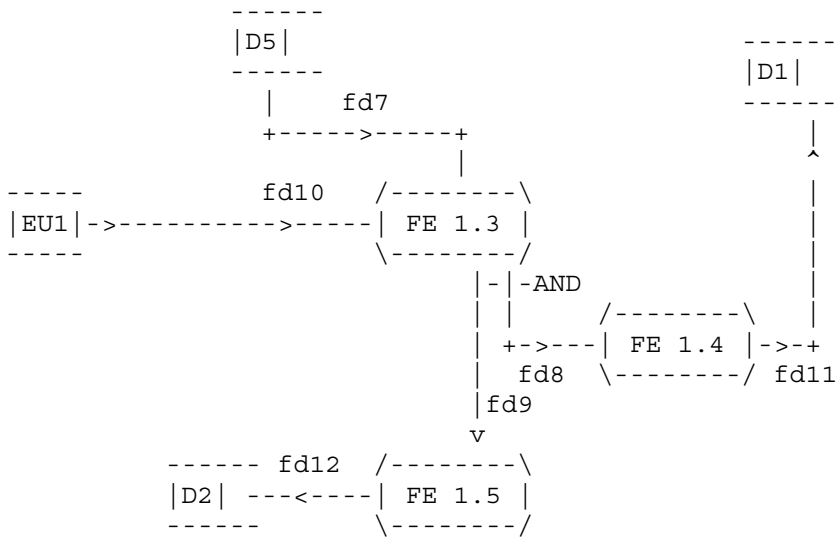
5.3. Implementarea tranzactiilor continand AND

In acest caz, pasii care trebuie urmati sunt urmatorii:

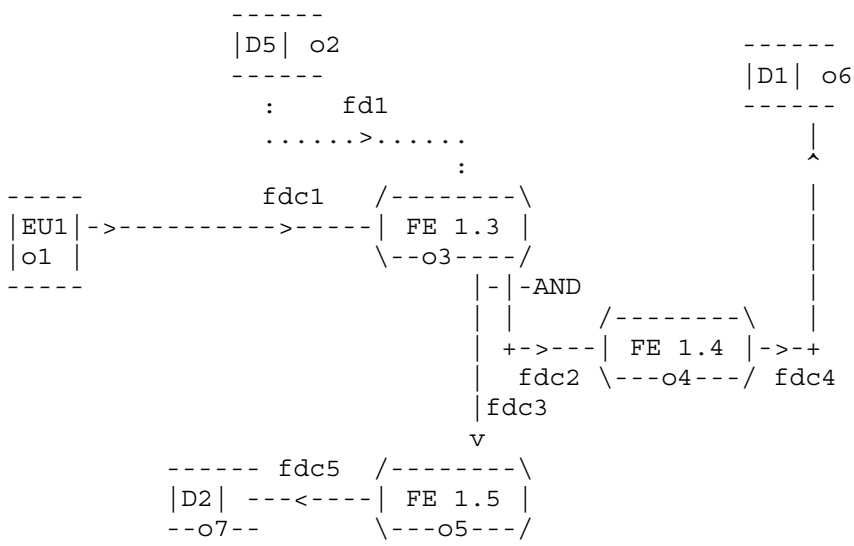
1. Definirea grafului asociat tranzactiei
2. Descompunerea acestuia intr-o multime de subgrafuri de tranzactii secventiale
3. Asocierea de tranzactii secventiale fiecarui subgraf
4. Asocierea unui AF cu fiecare tranzactie secventiala astfel rezultata.

Observatie: Exemplul anterior nu ne poate servi ca exemplificare pentru acest caz. De aceea vom folosi in continuare o alta tranzactie ADISSA, luata din DFD-1, si anume cea continand functiile elementare 1.3, 1.4 si 1.5. O prezentam in continuare.

Tranzactie ADISSA:



Tranzactie AF rezultata (fluxuri de date si de control renumerotate!):



Unde:

```
Entitati:                  Fluxuri de date (in diagrama ADISSA!)
EU1: Furnizori               fd7: Ordine de cumparare
                             fd8: Materiale cumparate
Date memorate:               fd9: Facturi si receptii
D1: Materiale in stoc        fd10: Facturi si receptii
D5: Ordine de cumparare      fd11: Materiale cumparate
D6: Cereri de lucrari        fd12: Facturi
      Functii:
      FE1.3: Inregistrare factura primita (de la furnizor)
      FE1.4: Actualizare stoc
      FE1.5: Actualizare cont furnizor
```

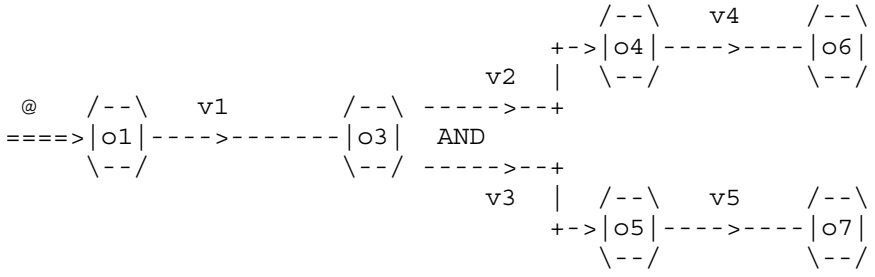
5.3.1. Definirea grafului tranzactiei

Se va defini intii ***graful de baza*** al tranzactiei, notat Gu. Acesta este un graf orientat avand ca noduri obiectele tranzactiei si ca arce FDC ale acesteia.
Din graful de baza se poate obtine un alt graf prin urmatorul ***algoritm de marcare***:

1. Se specifica o multime de puncte de intrare (= O multime de noduri din Gu. In functie de diverse specificari se pot obtine cu acest algoritm diverse grafuri)
2. Se viziteaza graful, pornind din punctele de intrare.
3. Se marcheaza toate nodurile in care se ajunge.
4. Se marcheaza toate arcele care formeaza cai de la punctele de intrare la nodurile marcate.

Graful asociat tranzactiei este format din nodurile si arcele marcate in cazul aplicarii algoritmului se mai sus avand ca puncte de intrare cele corespunzatoare lui λ (deci nodurile in care intra cel putin un pseudo-FDC). Graful rezultat se noteaza cu G0.

Exemplu: Pentru tranzactia de mai sus obtinem (am etichetat nodurile cu obiectele corespunzatoare si arcele au acelasi index cu FDC corespondent. Cu linie dubla am marcat punctele de intrare):



Observatie: arcele v2 si v3 sunt asociate prin AND.

5.3.2. Descompunerea in subgrafuri de tranzactii secventiale

Algoritmul este urmatorul:

Pasul 1. Pentru fiecare Vi din G0 care este asociat unui FDC implicat intr-un AND se creeaza un subgraf Gi setand punctul de intrare la Vi si aplicand algoritmul din paragraful anterior. Se obtine o multime de subgrafuri care il include pe G0:

{Gk} = {G0, ...}.

In cazul exemplului de mai sus, se obtine multimea {G0, G2, G3}, deoarece doar v2 si v3 sunt implicate intr-un AND.

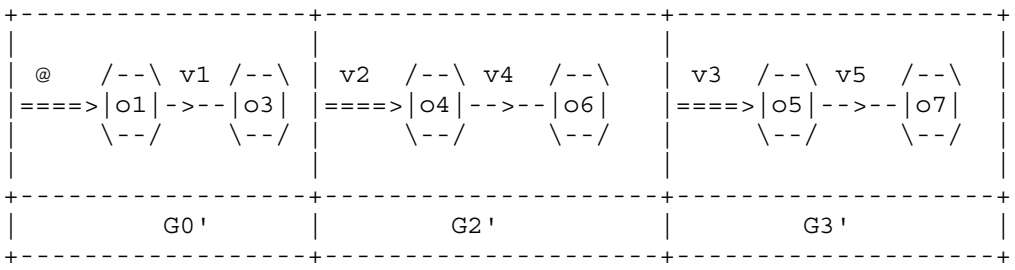
Pasul 2. Pentru fiecare subgraf Gk din aceasta multime, se calculeaza un nou graf, Gk' astfel:

a. Daca Gk nu contine arce implicate intr-un AND, Gk' = Gk.

b. Daca Gk contine arce implicate intr-un AND, acesta se elimina astfel:

2.1. Fiecare arc Vi din Gk implicat intr-un AND se elimina
2.2. Se parcurge graful rezultat pornind de la punctele de intrare ale lui Gk.
2.3. Arcele si nodurile parcurse vor fi si in Gk'.
2.4. Arcele si nodurile care nu se pot atinge in aceasta parcurgere se elimina.

Pentru exemplul de mai sus, G2' = G2, G3' = G3 iar G0' ≠ G0.
Cele trei subgrafuri sunt urmatoarele:



5.3.3. Asocierea de tranzactii secventiale subgrafurilor

In acest moment, subgrafurile obtinute nu mai contin AND si pe baza lor se pot defini tranzactii secventiale. Pentru ca acestea sa respecte cerintele sintactice ale tranzactiilor AF si in acelasi timp sa implementeze impreuna tranzactia AF din care provin (deci si AND-urile eliminate), sunt introduse urmatoarele "obiecte administrative":

a. Entitate de intrare: Cand punctul de intrare al subgrafului nu este @, se introduce o astfel de entitate care face transmiterea fluxului de control de la tranzactia apenanta la cea apelata.
b. Functie de transmisie: Transmite mai departe fluxul primit. Ea se foloseste in general pentru a evita conectarea directa a doua entitati.
c. Functie de apelare (calling function): Executa practic un AND, astfel:
- Activeaza tranzactiile implicate in acel AND (cele obtinute prin eliminarea AND-ului si setarea punctelor de intrare la arcele implicate)
- Asteapta terminarea tuturor tranzactiilor activate si trimite in acel moment un semnal entitatii de iesire corespunzatoare

d. Entitate de iesire: Specifica sfirsitul tuturor tranzactiilor corespunzatoare unui AND.

Vom considera, asemeni functiilor elementare din tranzactia ADISSA, ca functiile administrative au o implementare exacta.

Algoritm de definire tranzactii: Obiectele administrative vor deveni parte a tranzactiilor secventiale. Definirea tranzactiei Tk asociata lui Gk' se face astfel:

1. Fiecare nod este inlocuit cu obiectul original din tranzactia initiala (cel din care a provenit).

2. Fiecare arc este inlocuit cu FDC din care a provenit (din tranzactia initiala).

3. Daca Gk' ≠ G0', vor fi introduse urmatoarele elemente:
 a. O entitate de intrare
 b. O functie de transmisie
 c. Un FDC de la entitatea de intrare la functia de transmisie. Eticheta lui va fi aceeasi cu a FDC care a devenit anterior punct de intrare.
 d. Un FDC de la functia de transmisie la nodul destinatie al FDC devenit punct de intrare. Eticheta acestuia va fi aceeasi ca la c.

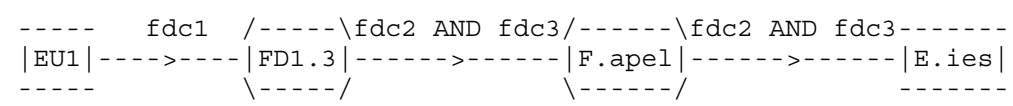
4. Daca un nod al Gk' corespunde unui obiect al tranzactiei initiale cu FDC care ies conectate printr-un AND, atunci se introduc urmatoarele elemente:
 a. O Functie de apelare
 b. O entitate de iesire
 c. Un FDC de la obiectul din care plecau AND-urile in tranzactia initiala la functia de apelare. Eticheta sa este combinarea prin AND a etichetelor tuturor FDC implicate in acel AND.
 d. Un FDC de la functia de apelare la entitatea de iesire, cu aceeasi eticheta ca la c.

Multimea de start a tranzactiei Tk obtinuta din Gk' este egala cu:

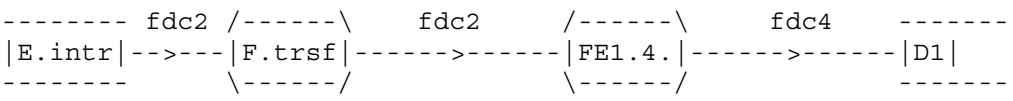
 a. multimea de start a lui T (tranzactia initiala) daca k = 0 (deci pentru T0 obtinut din G0')
 b. entitatea de intrare introdusa de algoritm, daca k ≠ 0.

Multimea de stop a tranzactiei Tk este determinata conform definitiei clasice: multimea obiectelor (entitati si date memorate) care nu au FDC care ies (deci inclusiv entitatile de iesire introduse de algoritm daca este cazul).

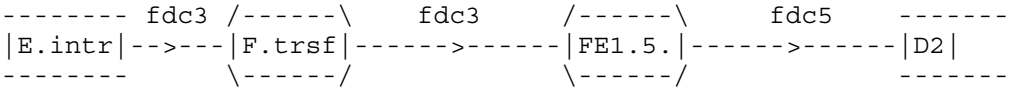
Pentru exemplul de mai sus, obtinem urmatoarele trei tranzactii AF secventiale care ilocuiesc tranzactia AF originala (continand un AND:



T0 obtinuta din G0'



T2 obtinuta din G2'



T3 obtinuta din G3'

5.3.4. Implementarea tranzactiilor prin AF

Tranzactia AF originala a fost deci inlocuita prin procedeul descris mai sus prin mai multe tranzactii AF secventiale, continand obiecte administrative. Fiecare dintre acestea va fi implementata printr-un AF cu ajutorul metodei de la tranzactii secventiale (cap. 5.2).

6. Consistenta intre specificatie si proiect

Demonstratia completa se gaseste in:
G. Babin, "Implantation esacte de transactions a l'aide de machines sequentielles", Master's thesis, Universite de Montreal, 1989.

Continutul respectivei lucrari este demonstratia consistentei intre comportarea AF rezultata ca urmare a aplicarii algoritmului si tranzactia AF initiala din punct de vedere al fluxului controlului.

Pr incipalele teoreme implicate in aceasta demonstratie sunt urmatoarele:

A. Tranzactii secventiale.
Teorema 1. Toate executiile unei tranzactii secventiale sunt implementate de AF corespunzator si nimic altceva.

Rezulta deci ca, daca exista o implementare exacta a AF, aceasta teorema demonstreaza ca implementarea fluxului de control al tranzactiei AF de catre AF este consistenta cu specificatia tranzactiei.

B. Tranzactii continand AND.

Teorema 2. Orice executie a unei tranzactii AF este executia multimii subtranzactiilor produse si nimic altceva.

Rezulta ca algoritmul de descompunere de la 5.3. produce un set de AF consistent cu tranzactia AF initiala.

C. Implementarea tranzactiilor AF prin AF Teorema 3. Orice executie a tranzactiei AF este executia multimii de AF produse si nimic altceva.

Rezulta ca daca exista o implementare exacta a AF, teorema demonstreaza ca nu numai proiectul (multimea de subtranzactii AF) ci si implementarea (multimea de AF) este consistenta cu specificatia tranzactiei.
Aceasta ultima teorema este rezultatul direct al teoremelor 1 si 2.

7. Consideratii finale

In cele de mai sus s-au facut unele asertiuni si s-au introdus unele restrictii referitoare la doua apecte:
1. Probleme referitoare la specificare
2. Probleme referitoare la concurenta

Sa evaluam corectitudinea lor.

7.1. Probleme referitoare la specificare

7.1.1. Asertiuni

a. "Pentru fiecare functie elementara exista o implementare exacta".
In general, aceasta asertiune care sta la baza demonstratiei de consistenta, este corecta prin aceea ca functiile elementare au asociata o portiune de cod care poate fi testata separat si care se poate valida ca este exacta.

b. "Pentru fiecare functie administrativa exista o implementare exacta".
Din punct de vedere al functiei de transmisie, care doar transmite fluxul de intrare nemodificat, nu sunt probleme.
In ceea ce priveste functia de apelare, asertiunea poate fi considerata corecta deoarece este o functie tipica sistemelor de operare. Daca problema este rezolvabila in cazul acestora, putem afirma ca ea este rezolvabila si in cazul nostru.

c. "Exista o implementare exacta a AF"
Aceasta asertiune sta la baza concluziei de la teorema 3. Cum regulile care definesc constructia AF pe baza tranzactiei AF secventiale sunt clar implementabile, si aceasta asertiune este corecta.

7.1.2. Specificatii incorecte

Algoritmul descris da doar garantia consistentei intre specificatii si proiect, dar nu valideaza in vreun fel specificatia.

In caz de specificare gresita, pot sa apara de exemplu cicluri infinite, din diverse cauze:

- specificarea incorecta a fluxului de control
- specificarea incorecta a fluxurilor elementare

7.2. Probleme referitoare la concurenta

In cele prezentate s-au introdus anumite restrictii, mai ales in ceea ce priveste concurenta. Ele nu sunt intotdeauna conforme cu cazurile aparute in realitate.

Concurenta poate avea loc la trei nivele:

A. la nivel de tranzactie
La 4.4 s-a exclus posibilitatea declansarii unei tranzactii de mai mult de un declansator la un moment dat. Sunt insa in practica sisteme care cer executia concurenta a unei aceleiasi tranzactii (ex: rezervare de locuri; se poate executa tranzactia de rezervare de la mai multe ghisee simultan).
Solutia la nivel de implementare poate fi: copii multiple ale tranzactiei, cod reentrant.

B. la nivel de subtranzactie
Ce se intimpla daca mai multe subtranzactii conectate printr-un AND cer acces la aceleasi date (deci probleme de tip LOCK/UNLOCK si evitare deadlock). In acest caz:
- fie cele doua tranzactii nu ar fi trebuit conectate prin AND
- fie se poate gasi o executie serializabila a acestora (in sensul dat in bazele de date).
Problema se poate rezolva implementand functia de apelare in asa fel incit ordinea in care aceasta apeleasa subtranzactiile sa fie favorabila evitarii deadlock-ului sau executiei eronate.

C. la nivel de obiect al tranzactiei.

In cazul in care un obiect apartine la doua subtranzactii, s-ar putrea crede ca este vorba de o executie concurenta a unui obiect.
Supozitia este insa eronata, deoarece atunci cand un obiect apartine la doua subtranzactii, este de fapt, la nivel de implementare, de doua copii diferite ale acestuia.

8. Concluzii

Toata metodologia descrisa se poate implementa sub forma unui sistem de proiectare asistata a sistemelor informatice (un generator de sisteme informatice interactive).
Din punct de vedere al performantelor sistemelor informatice care ar rezulta ca urmare a aplicarii acestor metode, ele nu au nici un motiv sa fie mai lente decit cele realizate pe alte cai.
Codul unui automat finit este extrem de simplu, el neputand fi motiv de intirziere. Singura problema o pun functiile elementare (cele care acceseaza date memorate mai ales), dar acestea implementeaza functiuni ale sistemului cerut de utilizator, deci vor fi prezente in orice implementare a sistemului informatic respectiv.

Avantajul prezentat de aceasta abordare este flexibilitatea deosebita a metodei, permitand dezvoltarea si intretinerea sistemelor informatice cu costuri mult reduse.