CAPITOLUL 4
METODOLOGII, TEHNOLOGII, LIMBAJ

4.1. CORBA - O ABORDARE INDUSTRIALA A SISTEMELOR DESCHISE DISTRIBUITE

Cadrul general de evolutie a CORBA (Common Object Request Broker Architecture) îl constituie OMG (Object Management Group), care este un consortiu care opereaza pe principiul de non profit în USA. Obiectivele OMG sunt de crea un standard pentru interoperabilitatea dintre aplicatiile dezvoltate independent, peste o retea. Membrii organizatiei includ majoritatea companiilor furnizoare de TI si multe companii end-user. Activitatea sa se realizeaza prin adoptarea de interfete si specificatii de protocoale într-un context adoptat de OMA (Object Management Architecture). Aceste specificatii sunt elaborate de un numar de membrii a caror activitate este colaborativa. OMG îsi orienteaza activitatea pe obiecte distribuite ca vehicul pentru integrarea sistemelor. Beneficiul major la construirea de sisteme distribuite cu obiecte este încapsularea: datele si starile sunt accesibile numai prin intermediul invocarii unui set de operatii definite si, în general, nu prin acces direct. Aceasta face mai simpla cooperarea într-un mediu eterogen (diferite implementari ale aceluiasi serviciu) deoarece diferentele legate de reprezentarea datelor sunt mascate. Obiectele, ca entitati definite, sunt simplificatoare ale evolutiei sistemului: noile servicii si implementari pot fi astfel introduse deoarece sustin aceleasi operatii ca si serviciile pe care le înlocuiesc. CORBA este componenta centrala a lui OMA si deja exista mai multe implementari comerciale care sunt accesibile.

OMA (Object Management Architecture) Arhitectura gestiunii obiectului.
Principalele componente ale OMA sunt prezentate în figura de mai jos

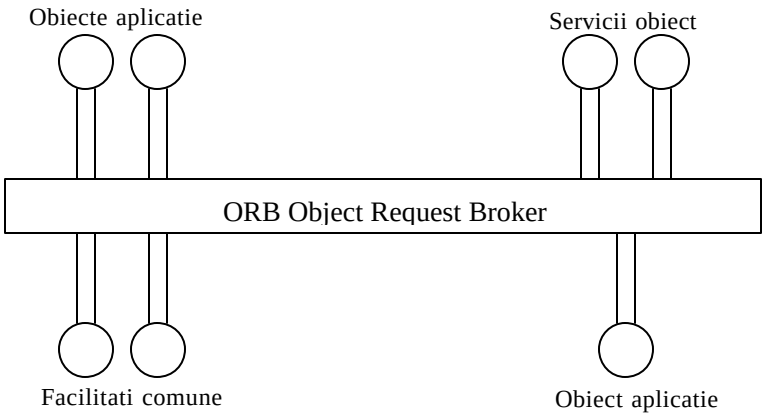


Figura 4.1. Principalele componente OMA

ORB este componenta centrala. Aceasta asigura abilitarea comunicatiei pentru toate celelalte componente. ORB încapsuleaza platforma de baza (i.e. sistemul de operare si reteaua) si asigura transparenta. La modul ideal, ORB trebuie sa asigure urmatoarele aspecte ale transparentei distribuirii unui obiect care comunica cu un altul, potential, la distanta:

- Locatia: verificarea situatiei în care celalalt obiect se afla pe aceiasi masina;
- Calea de acces: ruta pe care o ia mesajul schimbat cu un alt obiect;
- Reprezentarea: formatul datelor asociat cu celalalt obiect;
- Mecanismul de comunicare: care dintre mecanismele de comunicare sau protocoale este utilizat;
- Mecanismul de invocare: modul în care este executata metoda celuilalt obiect (detalii de procesare, startari, biblioteci legate dinamic etc.)
- Mecanismele de stocare: detalii relative la medii de stocare pe care celalalt obiect le poate sau nu le poate utiliza;
- Tipul masinii;
- Limbajul de programare: care este limbajul de programare care implementeaza celalalt obiect;

- Mecanismele de securitate: mecanisme specifice utilizate pentru controlul accesului la alte obiecte[32].

Orice modificare care survine relativ la aspectele de mai sus nu necesita recompilarea unui obiect particular. Aceasta permite ca modificarile sa fie realizate dinamic pe implementarea unui obiect, fara sa afecteze alte obiecte, nici chiar serverele sau clientii sai. În consecinta, apare simplicitatea introducerii unei implementari noi si înlocuire vechilor servicii.

Exista mai multe interfete posibile care pot fi date de ORB si care satisfac, evident, cerintele mai sus enuntate. Interfata standard a componentelor ORB ale lui OMA este denumita CORBA. Cea mai importanta caracteristica a lui CORBA este IDL (Limbajul de definire al interfetei). Acest limbaj este utilizat de alte componente ale lui OMA pentru a specifica serviciile pe care acestea le ofera celorlalti utilizatori de ORB.

Serviciile obiect sunt servicii de baza care sunt uzuale:

- Serviciile de numire care, în cazul în care este dat numele serviciului da o referinta acestuia care este la rândul ei utilizata pentru invocarea respectivului serviciu;
- Servicii ale ciclului de viata care pot fi utilizate pentru a controla obiecte, incluzând crearea, stergerea, mutarea sau modificarea acestora;
- Servicii persistente sau de stocare care pot fi utilizate pentru stocarea starii.

Serviciile obiect sunt specificate utilizând IDL-ul lui CORBA.

Facilitatile comune dau de asemenea servicii care apar uzuale pentru celelalte servicii. Facilitatile comune pot fi privite ca un "toolkit de aplicatie", în care serviciile comune pot fi privite ca "un toolkit de infrastructura" serviciile obiect vor fi disponibile pentru întreg ORB. În opozitie facilitatile comune sun optionale, dar daca exista acestea trebuie sa fie conforme cu specificatiile OMG. Exemple de facilitati comune: facilitati de interfata utilizator, facilitati de compunere a unui document, servicii specializate pentru un domeniu de aplicatii particular.

Facilitatile comune sunt specificate în IDL.

Serviciile furnizate de obiectele aplicate sunt specificate în IDL; aceste nu sunt neaparat standardizate de OMG. Astfel de exemple includ e-mail, tabloare, instrumente CASE, instrumente de cereri de date, instrumente CAD. O aplicatie OMA va consta dintr-o colectie de obiecte care interactioneaza. În mod obisnuit o astfel de colectie va consta din una sau mai multe obiecte aplicatie si un numar de servicii obiecte si facilitati comune. De altfel serviciile obiect si facilitatile comune pot da functionalitatea ceruta de un obiect, nefiind obligatoriu însasi pentru respectivul obiect sa utilizeze acea functionalitate. Exista posibilitatea implementarii functionalitatii însasi sau sa se utilizeze un serviciu nestandard furniza de obiect aplicatie.

CORBA Modelul obiect

Un obiect este o entitate care încapsuleaza stari si furnizeaza una sau mai multe operatii care actioneaza asupra starii respective. Aceste operatii pot fi cerute de catre clienti. O operatie este definita printr-o semnatura redactata în IDL, care include:

- o specificatie a parametrilor ceruti
- o specificatie a rezultatelor
- o specificatie a exceptiilor care pot sa apara la invocarea operatiei
- o specificatie a executiei semanticilor operatiei

Specificatia executiei va specifica una dintre cele doua semantici de executie diferite: 'at-most-once' sau 'best effort'. 'at-most-once' cere ca operatia sa fie realizata numai odata, în cazul în care aceasta returneaza succes; operatia este realizata cel putin odata daca returneaza exceptie. Operatiile de tipul 'best-effort' sunt de tipul numai cerere: acestea nu vor returna nici un rezultat, astfel încât clientul receptor nu va trebui sa raspunda si sa se sincronizeze pentru a încheia operatia.

Pentru a invoca o operatie, clientul trebuie sa identifice obiectul receptor. Pentru aceasta sunt utilizate referire obiectelor. O referire a unui obiect identifica totdeauna acelasi obiect. De multe ori acelasi obiect poate fi identificat prin mai multe referiri. La pornirea unei invocari de catre un obiect, acesta poate sa invoce la rândul sau mai multe operatii înainte de a analiza returnarea rezultatelor. Clientul nu are acces la invocarea directa a altor obiecte. La invocarea unei operatii pe un obiect dat, obiectul poate sa nu fie imediat accesibil pe ORB de catre obiectul receptor. O astfel de punere în

[32] Exista aplicatii, de exemplu comertul electronic în care securitatea este gestionata explicit de catre aplicatie. De cele mai multe ori se poate aplica un mecanism simplu de partitionare a grupurilor care sunt sigure si care nu sunt sigure.

asteptare se va numi dezactivarea obiectului, dar înainte de putea f executat obiectul trebuie activat. Aceasta poate solicita de la ORB o instantiere a procesului care contine starile obiectului si metodele acestuia.

O interfata este o multime de operatii posibile pe care le poate cere un client unui obiect. Interfetele sunt specificate în IDL. Asa cum este definit în OMG un obiect poate sustine mai multe interfete. Fiind data o interfata a unui obiect, un client poate invoca orice operatie din interiorul oricarei interfete sustinute de obiect. Obiectele pot fi create sau distruse va rezultat al unor operatii; mecanismul pentru aceasta este transparent pentru client. Rezultatul este perceput de catre client ca o referinta de obiect care identifica astfel noul obiect, sau ca distrugere de referinta.

Limbajul de definire al interfetei IDL

IDL este utilizat pentru a specifica componentele ORB ca servicii definite (pe nivelul aplicatie) a caror obiecte sunt disponibile clientului. IDL utilizeaza o sintaxa similara cu C++ pentru a defini operatiile dintr-o interfata. Se vor specifica complet, parametrii, rezultatele si exceptiile pentru fiecare operatie, dar nu se vor specifica semanticile pentru operatii. (acestea vor fi specificate ca 'best-effort' sau ca 'at-more-once'). Aceasta conduce la concluzia ca vor fi doua implementari care vor satisface aceiasi specificatie IDL, dar care au comportamente complet diferite. Exemplul de mai jos:

```
typedef unsigned long AccountNumber;
typedef unsigned long PersonalIdentificationNumber;

  exception NoSuchAccount {};
  exception InvalidPin {};
  exception InsufficientFunds {};

  interface Account
  {
  struct AccountRecord {
     string owner;
  float balance;
  string lastaccess;
  };
  void Credit (in float Amount);
  void Debit (in float Amount) raises (InsufficientFunds);
  void List (out AccountRecod List_R1);
};

Interface Sbank
{
Account Access (in AccountNumber acct,
  in PersonalIdentificationNumber pin)
  raises (NoSuchAccount, InvalidPin);
};
```

Figura 4.2. IDL pentru Sbank si Account

Aceasta prezinta doua interfete: una se refera la contul bancar, iar cea de a doua este o instanta a responsabilului bancar pentru accesul unui client dat la conturile sale prin furnizarea catre acesta a unei interfete la un obiect 'cont'. Specificatia starteaza cu o cupla de definitii de tip si ca urmare cu definirea a trei exceptii utilizator. Exemplul este simplu de urmarit. Este interesant ca pe interfata Sbank, exista doua posibilitati de a implementa operatia Account care returneaza o referinta la un obiect Account. Una dintre acestea presupune existenta obiectelor Account, astfel încât implementarea va stoca referirile si le va furniza la iesire ca un raspuns la o operatie Account. Cealalta alternativa este de a stoca starea asociata unu cont, de a se crea 'on the fly' un astfel de obiect la invocarea operatiei Account. De altfel, aceasta interfata ilustreaza doua aspecte importante ale CORBA. Primul se refera la faptul ca referintele obiectelor pot fi startate ca entitati din clasa întâi: sunt trecute ca parametrii de operati si sunt returnate ca rezultat. Cel de al doilea aspect se refera la faptul ca obiectele CORBA nu sunt privite neaparat ca fiind entitati mari, în termeni de stocare si de resurse de procesare solicitate. Exista un limbaj de mapare care defineste modul în care tipurile IDL CORBA sunt mapare pe tipul sistem al limbajului tinta. Suplimentar maparea specifica

modul în care anumite interfete ORB trebuie sa apara la programatorii care utilizeaza limbajul. Sunt definite astfel de mapari cel putin pentru: C, C++, Smalltalk, Ada.

ORB

Se vor utiliza termenii de 'client' si 'obiect' pentru a distinge între entitatile care primesc o invocare pentru a executa o operatie (obiectul) si entitatea care trimite cererea (clientul) Aceste doua aspecte sunt privite doar ca niste roluri pe care le joaca la un moment dat un obiect. Un client poate, el însusi, sa sustina un numar de operatii, iar când acestea sunt invocate, clientul va juca rol de obiect. Cea mai importanta functie a ORB este de a valida un client pentru invocarea unei operatii pe un obiect potential, la distanta. Pentru o astfel de realizare ORB va trebui sa mascheze protocoalele de baza si activitatea de retea utilizata pentru a trimite invocarea si a primi rezultatele. Clientul va identifica obiectul tinta prin intermediul unei referinte a acestuia. ORB este responsabil pentru localizarea obiectului, pregatirea datelor necesare pentru respectiva cerere (este posibila necesitatea activarii acestuia) si trecerea datele necesare de la cerere la obiect. Odata ce obiectul a executat operatia identificata prin cerere, daca este nevoie de o reluare, ORB este responsabil de comunicarea acestei necesitati, înapoi la client. ORB consta din mai multe componente logice distincte. Orice (incluzând canalele si structurile IDL), în afara clientului si obiectului sunt considerate ca fiind parte din ORB:

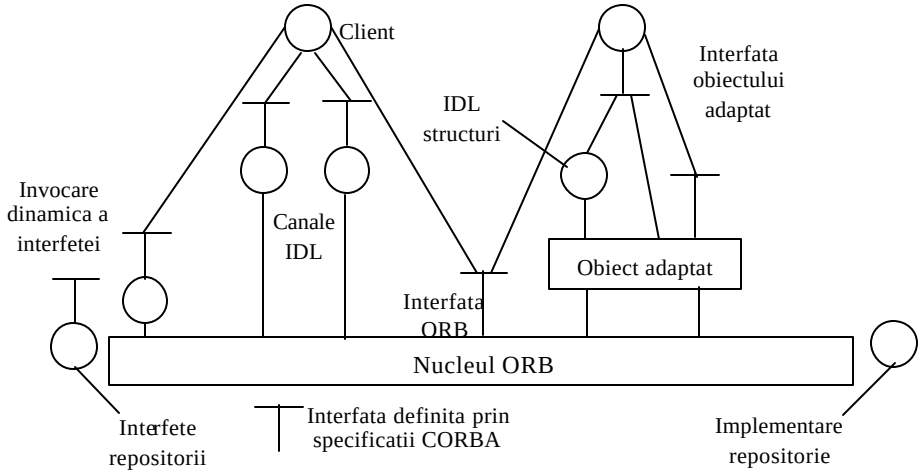


Figura 4.3. Structura de baza a unui ORB

Specificatia CORBA defineste explicit interfetele etichetate în diagrama. Exista câteva interfete care nu sunt explicit definite de specificatie: interfata dintre nucleul ORB si canalele IDL, interfata dintre adaptorul, obiectului si structura IDL, interfata dintre adaptorul obiectului si nucleul ORB. Prin nedefinirea acestor interfete specificatia CORBA permite mai multe implementari diferite. De exemplu, o implementare îsi propune sa furnizeze mai multa functionalitate în canalele IDL si adaptorul obiectului si mai putina pe nucleul ORB. Ceea ce este foarte important legat de acesta este ca atât clientul cât si obiectul nu sesizeaza diferente, deoarece CORBA specificat interfetele prin care acestia vor interactiona cu ORB. Astfel CORBA specifica regulile prin care obiectele interactioneaza unele cu altele, prin interfetele definite în IDL. Deci, implementarea obiectelor port dintr-o implementare CORBA la o alta trebuie sa fie triviale. Exista anumite lucruri care pot face ca implementarea obiectelor port sa nu fie triviala.

- CORBA nu defineste întreaga interfata utilizata de adaptorul obiectului pentru a activa obiecte (a da o definitie completa este dificil deoarece chiar ceea ce constituie activarea poate varia între sisteme de operare);
- CORBA permite existenta mai multor alternative pentru adaptorul obiectului, dat defineste interfata furnizata numai pentru una;
- Interfetele ORB sunt specificate în IDL cu o descriere în limba engleza a semanticilor fiecarei operatii definite pe interfata. Implementari diferite pot interpreta diferit specificatia redactata în engleza.

Interfata ORB furnizeaza un numar de operatii care pot fi aplicate oricarui obiect. Operatiile furnizate de interfata ORB includ:

- operatii care acopera referintele obiectelor la siruri si invers;

- operatii de release() si duplicate() pentru gestiunea memoriei utilizate de obiecte si de referintele obiect;
- operatia get_interface() necesara pentru interfata de stocare;
- operatia de create_request() utilizata în conjunctie cu interfata de invocare dinamica.

Un anumit mediu poate avea mai mult decât un ORB. Astfel, OMG include facilitati care permit obiectelor sa selecteze un ORB la initializarea acestora.

Atât canalele, cât si structurile IDL sunt generate de un program numit *'stub compiler'* din definitiile IDL ale interfetei. Fiind definite de catre OMG maparea DL pe un numar mare de limbaje, pentru un limbaj dat si o definitie data a interfetei, interfetele canal-client si structura-obiect sunt fixate.

Rolul pe care îl au canalul si structura IDL este de a masca detaliile fundamentelor ORB relativ la aplicatie, astfel încât invocarea la distanta apare ca si invocarea locala. Canalul transmite nucleului de baza ORB bufferul, împreuna cu referinta obiectului. Rutinele invocate de canalul client în nucleul ORB sunt cel mai probabil în acelasi proces ca si canalul IDL si clientul (probabil linkate într-o biblioteca a rularii). Sarcina nucleului ORB este de a localiza obiectul la distanta si de a-l pregati pentru a fi capabil de a primi cererea si în final de a o transmite, astfel încât acesta sa poata executa operatiile adecvate. Sunt de luat în considerare doua cazuri:

- cazul în care obiectul la distanta este deja activ
- cazul în care obiectul la distanta este pasiv.

În primul caz sarcina nucleului ORB din procesul client este de a localiza adresa IP a masinii pe care se afla obiectul la distanta si numarul portului pe care este vizibil obiectul la distanta. Dupa finalizarea acestui punct, partea client a nucleului ORB va transmite nucleului ORB la distanta bufferul, cu fragmentarea acestuia daca bufferul este prea mare. Nucleele ORB a partii client si a obiectului la distanta trebuie sa poata trata toate defectele care pot sa apara, cum ar fi pierdere sau duplicare de pachete, dând astfel comunicatiei semnificatia de fiabila. Sarcina de a localiza nucleul ORB al obiectelor la distanta poate deveni chiar complicata, daca obiectul la distanta este mobil, pentru aceasta fiind necesara o infrastructura foarte complicata. Aranjamentul bibliotecilor si al proceselor la obiectul la distanta va fi similara cu cea a clientului. În mod tipic obiectul la distanta va fi continut într-un proces linkat cu structura ISDL, adaptorul de baza al obiectului si anumite rutine date cu nucleul ORB. Un proces poate contine mai multe obiecte si structuri IDL (în terminologia ODP 'capsule'). Odata ce nucleul ORB al obiectului la distanta a primit bufferul si l-a reasamblat corect (daca acesta a fost fragmentat), referinta obiectului decodeaza numele operatiilor si a fiecarui parametru al acesteia, pentru a determina obiectul care trebuie sa recepteze bufferul. Daca obiectul nu este rezident în 'capsula' se va returna o eroare la nucleul ORB al partii clientului, iar restul de buffer este trecut la structura IDL pentru obiect.

Eventual nucleul ORB al clientului va un buffer de reluare. Acesta va verifica faptul ca, într-adevar, clientul este prezent în capsula si va plasa bufferul de reluare la canalul IDL. Canalul IDL va decodifica numele operatiei si al fiecarui parametru al acesteia si va transmite controlul clientului.

Invocarea unui obiect la distanta care este pasiv este similara invocarii deja prezentate, dar include un numar de pasi suplimentari în care obiectul la distanta este activat, dupa care lucrurile se deruleaza ca mai sus. Activarea unui obiect în CORBA este responsabilitatea adaptorului obiectului. În practica, adaptorul obiectului poate fi implementat ca o componenta a bibliotecii (linkata cu obiectul si canalul IDL) si o a doua parte este componenta a unui proces 'daemon' ORB care ruleaza pe fiecare host. Partea de adaptor obiect care ruleaza în 'daemon' ORB va fi partajata de toate obiectele care ruleaza pe host-ul respectiv. Pentru a activa un obiect 'daemon' va trebui sa instantieze un proces, sa ruleze o implementare a obiectului pentru orice stare persistenta a acestuia preluata din stocarea persistenta.

Odata ce nucleul ORB al clientului a determinat adresa IP si portul UDP (User Datagram Protocol) al obiectului la distanta, va contacta daemon ORB pe hostul la distanta care este vizibil pe portul UDP deja cunoscut. Daemon ORB trebuie sa transmita la ORB client daca obiectul este activ si vizibil pe vechiul port UDP. Alternativ acesta va returna noul port UDP daca obiectul este pasiv si transmite reactivarea acestuia (vechiul port poate fi deja indisponibil). Dupa ce nucleul ORB al clientului s-a asigurat de portul UDP pe care poate contacta obiectul la distanta, va transmite cererea dupa mecanismul anterior prezentat. Nu este firesc ca un client sa contacteze daemon ORB pentru obiectul la distanta pentru fiecare invocare. Mai degraba acesta va contacta servicuil ca în primul pas de utilizare a referintei obiectului, daca a aparut o eroare sau daca nu a utilizat referinta obiectului mult timp. Aceasta înseamna ca daca un client face cereri frecvente unui obiect daemon ORB va fi

implicat rar. În concluzie, canalul si structura IDL marcheaza multe dintre detaliile neplacute ale programarii distribuite pentru programator si satisface multe dintre cerintele de transparenta luate în considerare. Pentru programatorii client sintaxa invocarii obiectului la distanta este foarte apropiata de cea a invocarii locale a obiectului. Vor trebui doar sa amplifice codul scris cu exceptii suplimentare (specifice). Programatorii la distanta trebuie doar sa implementeze o operatie pentru fiecare dintre cele definite în IDL pentru interfata. De asemenea, programatorii trebuie sa scrie cod pentru tratarea activarilor si a dezactivarii, adica obiectul necesita stocare persistenta astfel încât dezactivare si activare sa se produca fara pierdere de stare. Modul în care se va produce acest cod este dependent de implementarea ORB si de specificitatea adaptorului obiectului, precum si de serviciul de obiect - persistent.

Interfata de invocare dinamica (DII)

DII furnizeaza clienti cu alternativa de a utiliza canale IDL la invocarea unui obiect. DII este uzuala pentru anumite aplicatii specializate cum ar fi bridge-urile si browser-ele. S-a anticipat ca multi clienti CORBA vor utiliza canale IDL. Un obiect care primeste o cerere nu poate conchide care din cele doua mecanisme vor fi utilizate de catre client. Asa cum s-a prezentat deja canalele IDL permit clientului sa utilizez aceiasi sintaxa la invocarea la distanta ca si la invocarea locala. Utilizarea DII implica clientul într-o suma de pasi suplimentari. Într-un anume sens, mecanismul DII este mai putin abstract ca si canalul IDL. Acesta mascheaza mai putine aspecte ale distributiei decât canalul IDL, solicitând o cantitate de munca mai mare pentru programatori. Cu toate acestea sunt clase de aplicatii pentru care utilizarea DII este mai eficienta. Una dintre aceste clase de aplicatii sunt browser-ele. Într-o abordare cu utilizarea canalului IDL pentru invocare browser-ul trebuie reconstruit pentru a încorpora un canal IDL pentru fiecare noua clasa care este adaugata bibliotecii. Prin utilizarea DII se poate baleia orice clasa fara a fi necesara încorporarea de canale IDLL atragerea facilitatilor comune apare necesitate utilizarii DII pentru a construi bridge-uri si gate-uri între diferite ORB. În schimb, noile versiuni al CORBA au inclus facilitatile corespunzatoare pentru servere, interfete structura dinamica, specifica astfel încât bridge-urile pot fi sustinute. DII sunt suport pentru invocari asincrone si invocari multiple asincrone.

Depozitul interfetei

Depozitul interfetei este un serviciu care da accesul la definitia interfetei. Acesta poate fi utilizat atât pentru componentele interne ORB, cât si de catre aplicatiile ORB. Cea mai buna abordare a întelegerii acestui serviciu este de a privi fiecare definitie IDL ca pe un arbore a sintaxei abstracte (AST). Exemplificarea se va face pe definitia exemplului anterior.

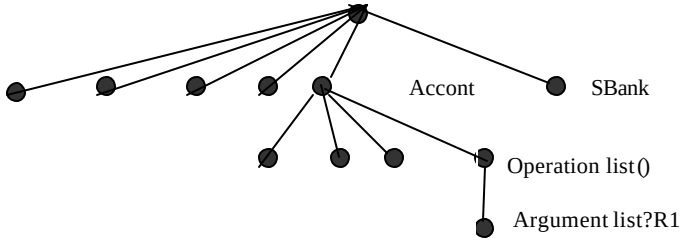


Figura 4.4. Arborele sintaxei abstracte

Nodului radacina i se poate atasa orice declarat în domeniul general. În acest caz a fost considerat ca fiind typedef, exceptiile si cele doua interfete: Account si SBank. Tot ceea ce este declarat în domeniul interfetei Account este atasat respectivului nod: definirea tipurilor pentru structura si arborele operatiilor. Atasat de nodul reprezentând o operatie se va plasa câte un nod pentru fiecare argument declarat în domeniul operatiei. Acestea sunt nivelele AST. Fiecare nod din AST va avea un numar de atribute asociate lui, iar fiecarui nod i se atribuie un atribut care este numele sau.

Depozitul interfetei trateaza fiecare nod din AST ca un obiect si da o multime de operatii pentru a descoperi relatiile pe care un obiect le are cu altele. Obiectele operatii sustin operatii care returneaza referinte la parametrii obiectelor, referinte la obiectele exceptie, precum si alte informatii relative la operatie. În revers, fiind data o referinta la o interfata obiect, pot exista operatii care pot fi invocate pentru a returna o lista a interfetelor declarate în domeniul respectivei interfete.

Definitia stocarii interfetei din OMG include un tip obiect primitiva pentru fiecare primitiva IDL, incluzând: interfata, operatia, parametrii, definirea tipurilor si exceptiile. Defineste, de

asemenea codurile tipurilor pentru toate tipurile de baza IDL, precum si modul în care sunt construite codurile tipurilor pentru tipurile complexe,cum ar fi structurile.

O stocare a interfetei poate fi responsabila cu gestiunea unui mare numar de definitii IDL redactate de diferiti programatori. Pentru aceasta gestiune este utilizata notiunea de modul.

În consecinta, fiecare stocare a interfetei apare ca si un obiect container, care furnizeaza un numar de operatii pentru a gestiona continutul acestuia. Pentru cazul în care este invocata operatia: contents() aceasta va returna lista referintelor obiect la obiectele modul pe care le contine. Daca invocarea respectivei operatii se face pe un obiect modul aceasta va returna lista referintelor la obiecte care include referintele la alte obiecte modul si la obiectele interfete pe care acestea le contin. Deci, pornind de la obiectul radacina (obiectul depozit al interfetei) este posibila navigarea într-un spatiu structurat la orice obiect interfata specific care poate furniza toate informatiile continute în specificatia IDL a unei interfete date. Operatii cum ar fi contents() regaseste informatia relativa la obiecte pe un nivel mai jos fata de obiectul curent. Fiecare obiect dintr-un depozit al interfetei are un identificator unic (în interiorul depozitului), cum ar fi un nume. Depozitul interfetei accepta operatii care vor returna o referinta la un obiect asociat cu un identificator unic, precum si operatii care vor returna lista obiectelor care au acelasi nume. Depozitul interfetei poate fi implementat ca un obiect aplicatie. Alternativ, acesta poate sa nu existe ca un obiect, caz în care ORB îsi asuma sarcina de a gestiona toata informatia relevanta si de a o face sa apara la nivelul programatorului de aplicatie ca unul sau mai multe depozite ale interfetei care deja exista. De altfel, pot fi scrise destul de multe aplicatii fara a fi necesara utilizarea depozitului interfetei. Acesta apare ca natural si necesar pentru mai multe clase de aplicatii cum ar fi: instrumentele CASE, clasa browser-elor, clasa gateway-urilor si a aplicatiilor care determina ec hivalenta tipurilor. Depozitul interfetei poate fi de asemenea utilizat prin invocarea dinamica pentru verificarea tipurilor Canalele IDL pot avea informatie de verificare a tipului codata în interiorul lor de catre 'stub compiler' deoarece acestea sunt specifice unei interfete particulare. În contrast cu aceasta DII nu ar sti ce interfata este necesar sa invoce, astfel ca acesta are nevoie de acces la specificarea interfetei (care este obtinuta în depozitul interfetei), pentru cazul în care este necesara orice verificare de tip.

Adaptorul obiect de baza

OMG specifica detalii ale unui singur adaptor obiect numit adaptorul obiect de baza (BOA). Un ORB poate furniza adaptoare obiect suplimentare. Principalele functii ale BOA sunt:

- generarea si distrugerea de obiecte
- activarea si dezactivarea de obiecte
- invocarea de obiecte prin intermediul IDL.

Suplimentar, BOA furnizeaza un suport minimal pentru securitate. Un obiect poate invoca operatia get_principal(), care va returna o referire la un serviciu de autentificare care este responsabil pentru invocarea curenta. Autentificarea în sine si detaliile control al accesului nu sunt specificate în OMG. Asa cum definitiile interfetei sunt stocate în depozitul interfetei, implementarile pentru BOA se asteapta a fi stocate într-un depozit de implementari. Un depozit de implementari tipic poate stoca programe compilate, scripte partajate pentru a starta si a initializa obiecte. Una dintre sarcinile mari ale BOA este aceea de a activa un obiect. Activarea unui obiect are loc în doua faze. În prima faza, BOA activeaza implementarea, ceea ce, în mod uzual înseamna startarea programului care contine obiectul. Un program poate contine mai multe obiecte, ceea ce conduce la situatia în care în faza a doua BOA sa activeze un obiect particular care este solicitat. Politica de activare a BOA descrie modul în care un obiect poate fi mapat pe un proces. OMG pretinde ca BOA sa accepte patru astfel de politici:

- activarea numai a unui singur obiect pe proces
- activarea mai multor obiecte pe proces
- rularea unui nou proces pentru a executa fiecare operatie
- permisivitatea ca o entitate din afara BOA sa activeze obiectul

Cu toate ca CORBA specifica detaliile numai pentru BOA , anumite medii pot contine mai mult adaptoare obiect. În consecinta, OMG include detalii asupra modului în care un obiect îsi poate selecta propriul adaptor obiect pe parcursul initializarii sale.

Interoperabilitate

Unul dintre scopurile principale ale CORBA este de a asigura interoperabilitatea dintre diferite ORB-uri, astfel ca un client dintr-un ORB poate invoca obiecte din diferite ORB-uri. Aceasta permite programatorilor de a construi aplicatii are utilizeaza obiecte care ruleaza pe diferite ORB-uri,

produse de mai multi furnizori. Abordarea din CORBA 2.0 este bazata pe un protocol de comunicatie comun care poate fi utilizat pentru comunicare cu obiecte care lucreaza pe diferite ORB-uri. Acest protocol este numit GIOP (General Inter-ORB Protocol), iar maparea acestuia pe TCP-ul Internet este IIOP (Internet Inter-ORB Protocol). GIOP specifica o reprezentare comuna a datelor, precum si mesajele si formatele acestor mesaje care pot fi transmise între client si server (obiect receptor al cererii) ORB. GIOP specifica, de asemenea, un numar de cerinte pentru protocolul de transport; acestea includ: livrarea fiabila 'at-most-once- a datelor, nereordonarea datelor si suportul pentru fragmentare. Anumite ORB-uri vor fi capabile sa sustina IIOP ca un protocol nativ, fie ca este singurul protocol pe care acestea îl utilizeaza pentru comunicare, fie ca poate sustine stive de protocoale multiple simultan. Cea de a doua abordare necesita ca ORB sa cunoasca pentru un mesaj dat ce protocol utilizeaza, realizând astfel comunicarea cu obiectele dintr-un alt OB prin utilizarea IIOP. Din pacate nu toate ORB-urile sunt capabile sa accepte IIOP ca protocol nativ. Pentru astfel de ORB-uri, OMG a specificat o alta abordare a interoperabilitatii – bridge-urile.

Un bridge este o aplicatie care permite obiectelor care utilizeaza un ORB de a comunica cu obiecte care utilizeaza alt ORB, chiar daca cele doua ORB-uri nu utilizeaza acelasi protocol. Daca unul dinte ORB-uri nu accepta IIOP ca protocol nativ interoperabilitatea se va realiza prin furnizarea asa numitului 'half-bridge' ce se manifesta între protocolul sau intern si IIOP. Figura de mai jos exemplifica interoperarea a doua ORB care utilizeaza 'half-bridge' pentru IIOP (cele doua 'half-bridge' lucreaza împreuna pentru a forma un 'bridge' complet. Cele doua obiecte care utilizeaza ORB-uri diferite sunt considerate ca fiind în domenii diferite. Trebuie remarcat faptul ca 'bridge'-urile din exemplu utilizeaza DII (interfata invocarii dinamice) si acesteia îi corespunde pe partea de server DSI (interfata structurii dinamice). Aceasta din cauza ca 'bridge'–ul este generic; poate fi utilizat pentru a invoca orice obiect indiferent ce interfata are acel obiect. 'Bridge'–ul poate fi construit si utilizând canalele si structurile IDL, dar în acest caz acesta nu poate fi utilizat pentru orice obiect, exceptie facând acele canale care sunt încorporate în 'bridge'. În consecinta, adaugarea de obiecte noi în cele doua domenii din exemplu, va fi necesara adaugarea a unei perechi noi de 'bridge' pentru a le face accesibile în alte domenii. Utilizarea DII si DIS evita aceasta complicatie.

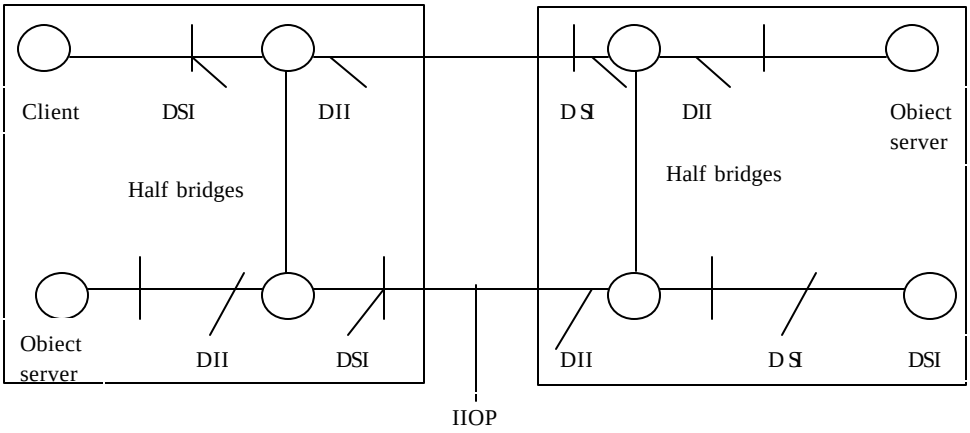


Figura 4.5. Utilizarea DII si DIS

Problemele se complica, aparând dificultati noi, pentru necesitatea maparii diferitelor reprezentari ale referintelor obiectelor care sunt trecute ca parametru sau returnate ca rezultat. De aceea, se impune ca toate ORB-urile conforme cu CORBA sa accepte IIOP fie ca protocol nativ, fie prin furnizarea unui 'half-bridge' la IIOP. ORB-urile pot astfel interopera cu alte platforme obiect, putând fi astfel utilizate ca vehicul al integrarii sistemelor.

Specificarea serviciilor obiect comune

Exista mai multe tipuri de servicii care sunt general utilizate, indiferent de domeniul aplicatiei; exemplele vor include stocarea, serviciile de stocare persistenta si de numire. Pentru a dezvolta aceasta OMG specifica interfetele standard pentru COS (Common Object Services). Sunt date câteva dinte serviciile care sunt cuprinse în COS.

Serviciul de numire

În serviciul de numire numele sunt tratate ca pseudo-obiecte. Aceasta înseamna ca limbajul de constructie permite programatorilor de a manevra numele ca obiecte CORBA ordinare. Cu toate acestea nu este o cerinta expresa ca acestea sa fie implementate ca si obiecte CORBA;

implementarile se vor realiza tinând cont de criterii de eficienta. Aceasta are avantajul de a masca reprezentarea interna a numelor. Contextele sunt obiecte ordinare CORBA si pot fi limitate la numele dintr-un context, exact ca si alte obiecte. Serviciul de numire defineste interfata contextului care contin operatiile:

- legarea unui obiect cu un nume în context
- stergerea unei legaturi dintre un obiect si un nume din context
- legarea unui nou context cu un nume, în contextul curent
- rezolvarea unui nume la un obiect dat (referinta)

Se propune standardizarea unei interfete de comert ca parte a COS. O astfel de interfata va permite clientilor sa specifice restrictii relative la valorile unor proprietati arbitrare pentru un serviciu la care se doreste legarea. Acesta va returna clientului unul sau mai multe referinte ale obiectelor care sunt asociate acestuia si care satisfac restrictiile proprietatilor enuntate. Acest serviciu apare cert ca si un serviciu de brokeraj.

Serviciul eveniment

Modelul de interactiune standard CORBA este unul în care clientul transmite o cerere la un obiect si asteapta un raspuns. Pot fi utilizate tratari care sa permita continuarea altor activitati la client, înainte de a primi raspunsul asteptat. De multe ori este de preferat un model de interactiune asincrona. Pentru a sustine acest tip de interactiune OMG defineste un serviciu eveniment. Conceptual un astfel de serviciu furnizeaza un canal eveniment între unul sau mai multi consumatori de eveniment si unul sau mai multi producatori de eveniment

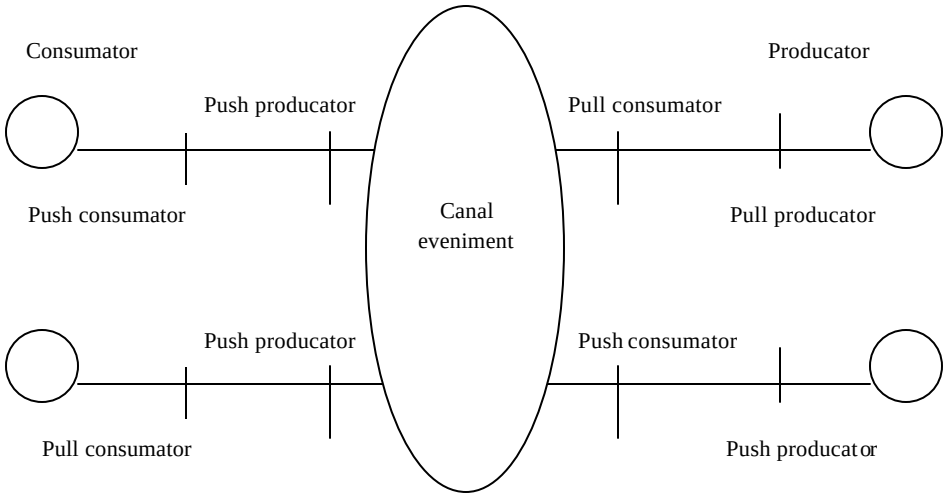


Figura 4.6. Definirea canalului eveniment

În exemplificarea din figura de mai sus la utilizarea unui canal eveniment cu o interfata call-back. Canalul eveniment este prezentat ca un singur obiect asa cum acesta pare la programatorul de obiecte client sau furnizor. În practica, implementarea se va produce, de regula, prin mai multe obiecte care colaboreaza. Atât pentru consumator cât si pentru furnizor sunt doua moduri de baza de interactiune: 'push' si 'pill'. În modul de interactiune push consumator, un canal eveniment va plasa evenimentul la consumator prin apelarea unei operatii de pe interfata sa call-back, cu informarea clientului relativ la faptul ca evenimentul s-a produs. În modul pull un consumator trebuie sa invoce o operatie pe canalul eveniment solicitându-l (în cazul în care nu s-a aparut înca un eveniment raspunsul poate fi imediat sau se poate produce blocarea pâna la aparitia unuia). În modul push furnizor furnizorul invoca operatia push() la canalul eveniment pentru a-l informa despre producerea evenimentului. Aceasta operatie are un singur parametru si nu se returneaza nimic. Invers, în modul pull canalul eveniment invoca o operatie la interfata call-back pentru a verifica aparitia unui eveniment sau blocarea pâna la aparitia unuia. Este acceptat ca diferite canale eveniment sa aiba semantici diferite si sa sustina calitatea serviciilor diferita.

Servicii ale ciclului de viata

Serviciile ciclului de viata definite de OMG sunt proiectate pentru a sustine crearea, mutarea, copierea si stergerea de obiecte. Sunt astfel definite trei interfete, dupa cum urmeaza:

- obiecte ale ciclului de viata care sustin operatii de copiere, mutare, eliminare

- interfata de exploatare utilizata pentru crearea obiectelor
- interfata de exploatare a detectorului utilizata pentru a detecta exploatarile

Un obiect care accepta interfata ciclului de viata trebuie sa furnizeze operatii pentru mutarea, copierea si eliminarea lui însusi. Pentru a sustine primele doua cerinte este necesara întelegerea conceptului de locatie. Semanticile tuturor celor trei operatii vor fi specifice aplicatiei.

Este definita o interfata de exploatare generica pentru a crea obiecte. O exploatare generica, uzual, va invoca cod specific aplicatiei pentru a crea un obiect de un tip particular. O exploatare data reprezinta o implementare pe o locatie particulara si este responsabila de crearea de obiecte pe locatia respectiva. Exploatarile nu sunt obiecte speciale: interfata lor este definita în IDL, iar accesul unui client la o exploatare necesita un obiect la care se face referire. La crearea unui obiect de catre o exploatare, aceasta este responsabila cu alocarea de resurse pentru obiectul respectiv, incluzând stocarea persistenta a acestuia. Protocolul dintre exploatare si obiectul pe care îl creeaza este specific aplicatiei. Pentru a regasi o exploatare este utilizata cea de a treia interfata. Aceasta permite unui client de a regasi o exploatare pe o locatie particulara.

Serviciul de relatii

COS defineste un serviciu de relatii pentru gestiunea relatiilor dintre obiecte. Relatiile pot fi 'many-to-many', aceasta necesitate privind din punctele de vedere practice care se manifesta la nivelul aplicatiilor. Un alt concept important care este definit prin specificarea relatiilor se refera la 'rol'. Acelasi obiect, privit din puncte de vedere diferite poate juca mai multe roluri. Utilizarea acestuia într-o relatie cu alte obiecte este esential dependenta de rolul pe care, în particularitatea relatiei respective, îl joaca obiectul. Atât relatiile cât si rolurile sunt obiecte de clasa întâi. OMG defineste interfetele:

- RelationshipFactory
- Relationship
- RoleFactoy
- Role

Se va limita tratarea la interfata Role care , de altfel este si cea mai complexa fiind definita prin specificarea relatiei. Aceasta contine mai multe operatii pentru gestiunea rolurilor:

- poate enumera relatiile în care acesta participa, prin returnarea unei referinte la fiecare dintre obiectele relatiei corespunzatoare;
- furnizeaza o operatie pentru a returna alt obiect rol, daca este data o interfata la un obiect relatie;
- da o operatie pentru distrugerea obiectului rol;
- da o operatie pentru legarea la un alt rol într-o relatie nou creata.

Relatiile si rolurile nu necesita stocarea vreunei stari din partea obiectelor carora le sunt asociate. Astfel, nu este necesara nici o schimbare a starilor pentru un obiect la modificarea diferitelor roluri si relatii în care acesta este implicat. Aceasta permite obiectelor imuabile sa fie relationate, iar acele relatii sa fie manevrate, fara activarea obiectelor în sine.

Serviciul de tranzactii

Tranzactia furnizeaza garantii care sunt uzuale pentru constructia aplicatiilor distribuite dependente. COS defineste un serviciu de tranzactii ca fiind compatibil cu standardul DTP al X/Open. Aceasta este, de altfel, si intentia declarata, ca aplicatiile care utilizeaza serviciul de tranzactii trebuie sa fie capabile sa interopereze cu aplicatii care sunt conforme cu DTP al X/Open.

Pentru a se crea o tranzactie se invoca o operatie create a unei 'transaction factory'. Aceasta va returna o referinta la un obiect terminator si la un obiect coordonator, obiecte care sunt utilizate pentru a gestiona tranzactia.

Obiectul terminator este utilizat pentru a savârsi sau aborta tranzactia. Obiectul coordonator gestioneaza starile tranzactiei, ceea ce include: înregistrarea obiectelor care sunt participante la tranzactie si a tranzactiilor imbricate care sunt create în domeniul dat al tranzactiei curente.

Sunt definite si alte doua interfete importante:

- interfata Resource – contine operatiile pe care le va invoca sistemul de gestiune al tranzactiilor asupra unei resurse pe perioada tranzactiei.
- interfata TransactionalObject – aceasta interfata nu contine nici o operatie. Este utilizata pentru a declara ORB-ului de baza ca obiectul este unul tranzactional. Un obiect tranzactional este responsabil pentru implementarea tuturor operatiilor obiectelor. La invocarea unui astfel de obiect, acesta este responsabil cu înregistrarea unui obiect resursa cu coordonatorul tranzactiei.

Pot fi invocate mai multe obiecte în interiorul unei singure tranzactii. Aceasta semnifica faptul ca se impune propagare contextului tranzactiei la toate aceste obiecte. Aceasta poate fi realizata atât implicit si explicit. În cazul în care propagarea este explicita, se va include o referinta la obiectul coordonator ca unul dintre parametrii operatiei. În acest caz argumentul trebuie sa apara în definitia IDL a operatiei. Alternativa: propagarea implicita determina ORB sa recunoasca faptul ca respectiva operatie este pe o TransactionalObject si include, în mod automat, contextul tranzactiei, ca parametru.

Serviciul tranzactiilor defineste interfata dintre ORB si managerul tranzactiei. Managerul tranzactiei este utilizat de catre acest serviciu pentru a implementa obiectele care gestioneaza tranzactie cum ar fi terminator si coordonator. Este esential faptul ca managerul tranzactiei se auto înregistreaza cu un ORB si în consecinta ORB-ul îl utilizeaza pentru a asigura propagarea contextului tranzactiei si a faptului ca tranzactia este bine gestionata.

Serviciul de tranzactii este un serviciu obiect uzual în care specificare necesita participarea ORB-ului de baza si specifica interfetele pe care ORB trebuie sa le sustina. În particular, ORB-ul este responsabil cu propagarea contextului tranzactiei (pentru propagarea implicita) si astfel cu informarea permanenta a managerului tranzactiei, printr-o cerere de raspuns care este emisa sau receptionata în interiorul tranzactiei. Mesajul este evaluat de managerul tranzactiei care va examina contextul acesteia si va decide asupra 'comiterii' tranzactiei sau a abortarii acesteia. Un ORB tranzactional poate fi construit prin utilizarea de adaptoare de obiect diferite, prin modificarea canalelor IDL si a structurilor IDL si prin reutilizarea nucleului ORB existent. Multe din componentele ORB nu vor necesita modificari. Obiectele care nu sunt tranzactionale vor utiliza adaptoarele de obiect si canalele si structurile IDL initiale.

Facilitatile comune

Facilitatile comune vor da nivelul aplicatie sau facilitati utilizator care sunt interconectate cu ORB si COS. Astfel de servicii vor avea interfete standard definite în IDL. Sunt definite urmatoarele categorii de facilitati comune:

- interfata utilizator – aceasta include sustinerea pentru afisarea obiectelor, a mecanismelor de stocare si prezentare a informatiilor de help a aplicatiilor, precum si facilitati de utilizare;
- managementul informatiei – aceasta include sustinerea stocarii si regasirii informatiei (SQL), precum si mecanisme care valideaza interoperarea obiectelor prin schimbul de date (EDI, ASN.1 SDL etc.);
- managementul sistemului – serviciile din aceasta clasa trata cu managementul si controlul retelelor si al obiectelor (incluzând entitati fizice cum ar fi nodurile, ruterele, precum si entitati logice ca utilizatori si aplicatii). Astfel de servicii vor trebui sa includa tipul de functionalitati date curent de SNMP si CMIP;
- managementul sarcinilor – serviciile din aceasta categorie vor trebui sa sustina concepte cum ar fi 'workflow'. Astfel de facilitati trebuie sa poata furniza suportul pentru migrarea obiectelor prin sistem de la un utilizator la altul ca o unitate de activitate. Alte servicii din aceasta categorie vor furniza suport pentru agenti – programe care migreaza în retea actionând diferite obiecte. Caracteristic pentru acestea este Java;
- Piata – serviciile din aceasta categorie vor sustine activitati de tipul contabilitate, fabricarea integrata, simularea distribuita.

Este momentul sa se faca o prezentare diferentiata asupra interfetelor si obiectelor. Probabil cea mai importanta diferenta între CORBA si ODP este modul în care ODP trateaza interfetele. Atât în CORBA cât si în ODP un obiect poate avea mai multe interfete de tipuri diferite. ODP trateaza interfetele ca si entitati de clasa întâi: clientii au referinte la interfetele care le valideaza dreptul de invocare pe respectiva interfata. Un client necesita o referinta separata pentru fiecare interfata a obiectului pentru a fi capabil de a face invocarea unei operatii pe acea interfata. În ODP nu exista notiunea de referinta a obiectului. În contrast, CORBA nu are notiunea de referinta a interfetei. Daca un client are o referinta la un obiect acesta poate invoca orice operatie pe orice interfata a obiectului. Facând referintele interfetelor entitati din clasa întâi se vor obtine mai multe avantaje:

- interfete diferite pot sustine diferite functionalitati;
- interfete diferite pot avea controale de acces diferite.

Chiar daca la prima vedere apare ca o introducere de complexitate suplimentara, beneficiile se vor regasi în micsorarea costurilor. CORBA este convenabila pentru o mare varietate de domenii de aplicatii si în particular pentru construirea de sisteme integrate si de aplicatii de management.

Obiective

Procesul HOORA îsi propune sa furnizeze suportul pentru analiza cerintelor orientate obiect, ierarhice si sa dea sustinerea particulara pentru sisteme în timp real (fara a fi exclusiv orientata spre aceasta). Este necesar un suport adecvat pentru ierarhie, ca mecanism principal pentru tratarea complexitatii în sistemele mari. Din nefericire ierarhiile sunt, de obicei neglijate, în abordarea orientata obiect. HOORA îsi va propune, sa prezinte un set de un ghid clar de integrare a ierarhiilor cu orientarea obiect. Versiunile anterioare HOORA [Galle95], [Galle96a] ataca, diferentele semnificative, dintre conceptele orientarii obiect, pe de o parte si conceptele functionale sau orientate pe date pe de alta parte. Aceasta apare ca nucleul succesului metodelor orientate obiect: integrarea aspectelor (asa zise) total diferite într-un cadru coerent. Se subliniaza ca tendintele actuale ale orientarii obiect sunt legate de supralicitarea aspectelor structurale în detrimentul aspectelor dinamice (si a fortiori) a aspectelor de timp real, aruncându-le catre proiectarea de detaliu si respectiv catre implementarea sistemului. Aceasta apare ca o separare neta dintre lumea OO si lumea RT, ceea ce este prezentat în figura de mai jos [Buhr, Casselamn 96].

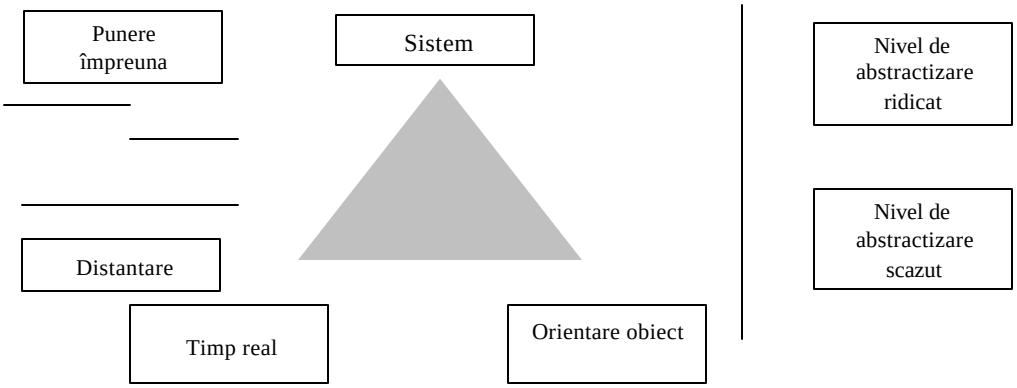


Figura 4.7. Relatia dintre OO si RT

Unul dintre scopurile principale este de a lega anumite aspecte dinamice si de timp real fazei de cerinte. Este necesar, în acest sens un cadru conceptual pentru a valida aspectele astfel introduse fara prea multe detalii specifice de proiectare. Cadrul trebuie sa permita realizarea proiectarii aspectelor de timp real si dinamice la acest nivel înalt. Standardul UML a definit un set consistent de notatii, dar acest punct de vedere este înca putin integrat. Nu apare în scopul UML sa defineasca un set integrat de puncte de vedere, asa cum aceasta vrea a fi echivalenta cu definitia unui proces UML. Procesul HOORA îsi propune descrierea unei abordari integrate, care combina orientarea obiect cu conceptele de timp real sau dinamice.

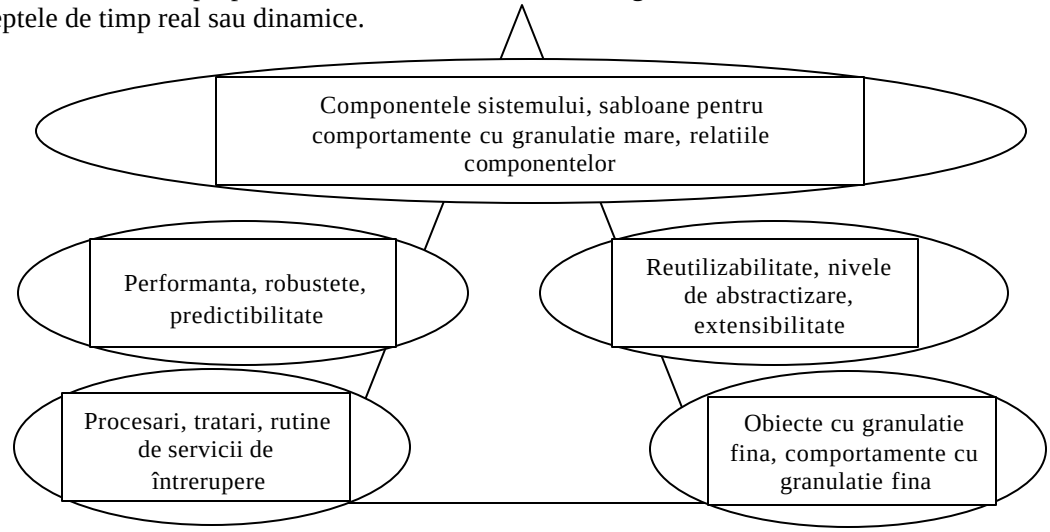


Figura 4.8. Abordarea integrata OO, RT

Procesul HOORA este conform total cu UML 1.3. UML este utilizat pentru toate aspectele notationale ale procesului HOORA. Chiar termenul de proces este utilizat în sensul dat de UML. Un proces defineste ceea ce trebuie realizat, când trebuie realizat si modul în care se produce realizarea. În analiza cerintelor software, scopul declarat este de a construi un produs de analiza consistent si neambiguu. Este necesar ca un astfel de proces sa serveasca drept ghid pentru toti participantii: clienti, utilizatori, producatori si manageri executivi. HOORA îsi propune sa furnizeze o ghidare efectiva prin descrierea secventei de pasi care trebuie urmati pentru a putea aplica analizele orientate obiect. Aceasta este direct legata de ghidari si diagrame cu semantici si reguli de integrare superiore. Sunt utilizate diagramele UML dar semanticile acestora sunt rafinate în contextul pasilor procesului HOORA. Hoora este, în mod particular convenabila pentru medii spatiale.

Metode de analiza orientate obiect în spatiu.

Metoda de proiectare ca a fost recomandata în contextul ESA si care este utilizata în mod curent pentru fundamentarea software este HOOD[HOOD95]. Aceasta ofera o metoda si o notatie pentru a proiecta obiecte software si cooperarea acestora. HOOD acopera faza de proiectare arhitecturala, faza de proiectare detaliata, pâna la ODS (Object Description Skeleton) si chiar pâna la generarea codului în Ada. Pentru a acoperi si faza de specificare metoda HOORA este prelungita cu E2S, metoda care este bazata în principal pe OMT si care a fost completata si cu aspecte ale timpului real în programul Realism.

Noua versiune HOORA, descrie în acest document, combina beneficiile comerciale ale UML cu Know-how-ul existent, prin exprimarea metodei HOORA în UML, în consecinta complementându-l pe acesta cu o metoda.

UML este caracterizat de suma de avantaje, în principal pentru ca este foarte utilizat în piata. O suma de informatii, educatie, carti si instrumente sunt deja disponibile. UML ofera mai mult decât OMT si acopera faza de cerinte, fiind flexibil si relativ simplu de croit. Diferitele puncte de vedere pe care le ofera UML pentru sisteme pot fi utilizate pentru definirea seriilor de test acoperind astfel sistemul din aceleasi puncte de vedere. Aceasta abordarea permite o testare, pe tot parcursul ciclului de viata fiind mai completa decât traditionalele teste functionale. Pentru proiectare, însa UML este putin convenabil fiind completat de HRT-HOOD, de care este nevoie în continuare, fiind de altfel propusa si o mapare dintre UML si HOOD. Deoarece UML este doar o notatie sper necesara existenta unei “ceva în completare” pentru a face aceasta notatie mai solida. Acest nedefinit înca “ceva” este apelat succesiv un proces, un set comun de stereotipuri, o semantica comuna etc. (care poate fi verificata cu un instrument de verificare personalizat) sau o metoda (posibil ierarhizata) UML, însa, din pacate, nu ia în considerare timpul real ca o aprofundare a întelegerii din spatiul software. Se pune, deseori întrebarea daca orientarea completa obiect este convenabila pentru acest tip de software. Exista legaturi între metoda de proiectare si limbajul de implementare, anumite caracteristici ale proiectului fiind obligatoriu de sustinut de limbaj. Sunt astfel evidentiate perechi de tehnici care se sustin reciproc: HOOD3/Ada83, HOOD4/Adfa95, UML/C++. Cu toate acestea UML poate fi utilizat bine în combinatie cu Ada95, dar va trebui sa fie restrictionat la utilizarea lui în combinatie cu Ada83. Pe parcursul ciclului de viata a software-ului, UML este utilizat pentru cerintele clientului, împreuna cu specificatiile textuale. Este utilizat mai mult ca furnizor de cerinte.

Suportul specificatiilor UML este centrat cu precadere pe studiul de caz. Modelul datelor poate fi exprimat prin diagrame clasice, permitând reutilizarea acestuia la nivelul proiectarii. Task-urile pot fi mapate pe clase asociate cu diagramele de stare pentru a putea exprima sincronizari si comportamente.

Scopul modelarii

Pentru a valida întelegerea sistemului se vor construi modele. Modelele astfel construite sunt conforme numai relativ la elementele sistemului luate în considerare si care trebuie validate. Din toate celelalte puncte de vedere acestea sunt incomplete. Tocmai aceasta incompletitudine este benefica, deoarece nu impune constructia întregului sistem pentru a considera o parte din acesta. Un model va consta, deci, dintr-o multime de perspective diferite care investigheaza aspecte diferite, relativ independente ale sistemului. Cu privire la sistemele software exista mai multe lucruri care trebuie testate fara sa determine cresterea costurilor constructiei întregului sistem.

- analistul va trebui sa testeze propria întelegere a problemei pe care sistemul se presupune ca o va rezolva;
- analistul si proiectantul trebuie sa testeze daca solutia propusa rezolva problema pentru care se naste sistemul;

- proiectantul trebuie sa testeze daca structura software propusa va functiona asa cum a fost planificat, în termeni de functii si performanta;
- proiectantul trebuie sa testeze daca structura software propusa poate fi modificabila, mantenabila si reutilizabila.

Aceste patru domenii ale continutului pretind patru tipuri diferite de modele. În HOORA se face limitarea la primele doua tipuri de modele. Aceste doua tipuri de modele difera numai prin nivelul lor de detaliere. HOORA este exclusiv directionat pe obiectivele analistului. Se definesc scopurile modelelor de analiza ale HOORA [Davis93]:

- (validarea) întelegerii sistemului
- definirea restrictiilor
- rafinarea restrictiilor
- rezolvarea conflictelor dintre restrictii
- expandarea informatiei
- verificarea de consistenta

Procesul care va fi descris va starta cu faza de cerinte textual. Textul nu va fi niciodata perfect, dar este important sa se porneasca cu o descriere în limbajul utilizatorului. Obiectivul modelarii va fi cheia pentru structurarea aceste descrieri si identificarea ambiguitatilor si a domeniului problemei. La finalizarea tuturor modelarilor, se va completa întelegerea printr-o multime de cerinte textuale, care identifica si solutioneaza diferite ambiguitati.

Descrierea HOORA

Caracteristica majora a proiectului HOORA este ca acesta necesita definirea unei metode clare. Aceasta implica mai mult decât notatia UML. UML standard descrie posibilitatea de a amplifica notatia UML cu semantici (specifice procesului) suplimentare. Unul din obiectivele propuse pentru acest document este de a defini o astfel de metoda.

Metoda HOORA este orientata pe cerinte – proiectele software mari necesita o faza de identificare clara a cerintelor. Captarea cerintelor sau analiza nts a cerintelor nu se confunda cu modelarea sau diagramarea. Faza de cerinte îsi propune producerea cerintelor care sunt elemente textuale tipice care explica caracteristicile cerintelor unui sistem software. Procesul cerintelor nu trebuie limitat prin utilizarea unei notatii sau a unui instrument specific. Trebuie însa asigurata trasabilitatea pentru ca utilizatorul sa poata relationa elementelor modelului aceste cerinte. Suplimentar, trebuie furnizata si sustinerea inginerie unui astfel de proces: reorganizarea, lincarea, setarea de atribute.

Metoda HOORA este orientata pe studiu de caz – Metoda permite startarea unui proiect prin studii de caz si documentarea studiului de caz, în principal prin abordari gen template. Studiile de caz structureaza cerintele functionale. Un studiu de caz este pisa a functionalitatii sistemului care va da utilizatorului un rezultat sau o valoare. Un studiu de caz necesita încarcarea mai multor câmpuri, cum ar fi: scopul, preconditiile, postconditiile, declansarile, actorul primar, actorii secundari, spatii de baza ai scenariului, extensiile, domeniul de variatie a valorilor etc.

Simultan cu identificarea si elaborarea studiului de caz, acestea vor fi reprezentate într-o diagrama de studii de caz, reprezentând actori, studii de caz, asociatii între studiile de caz si asociatiile dintre studiile de caz si actori. Identificare si elaborarea studiilor de caz este mai importanta, evident, decât diagrama rezultata. Se vor crea legaturi între studiile de caz si cerintele utilizatorului si se vor crea secvente de diagrame.

Metoda HOORA este iterativa si incrementala – dezvoltarea unui produs software este o sarcina ampla care ia un timp mare. Utilizând oricare abordare structurata a ciclului de viata aceasta va fi divizata în sub-proiecte mici. Fiecare dintre aceste sub-proiecte este o iteratie care va completa mai târziu rezultatele anterioare. Metoda permite identificarea, planificarea si vizualizarea unor astfel de incremente. Proiectele sunt divizate în iteratii. Iteratiile produc iesiri vizibile, cu semnificatie pentru utilizatorul final. Aceasta abordare este legata de cresterea monitorizarii proiectului si redirectionarea capabilitatilor. Iteratiile sunt legate cu studiile de caz si/sau scenarii specifice (a se interpreta ca fiind diagrame de secvente) legate de studiile de caz. Studiile de caz sunt specificate, proiectate, dezvoltate si acestea vor constitui sursa de la care se vor construi cazurile de test.

Metoda HOORA este centrata pe arhitectura - Arhitectii distribuie sistemul într-o forma. Aceasta forma necesita proiectarea de o maniera la care sistemul devine posibil a fi modificat. Aceasta implica faptul ca arhitectura trebuie sa identifice multimea corecta a abstractizarilor, denumite subsisteme sau clase, carora le sunt alocate responsabilitatile cerute de sistem. Scopul alocarii responsabilitatilor de sistem este de a avea stabilitate, chiar într-o lume marcata de schimbari.

Constructia acestei forme este de multe ori confundata cu crearea întregii multimi de diagrame UML: diagramele de clase, diagramele de obiect, diagramele de pachete, diagramele colaborative, diagramele de secvente, diagramele de stare, diagramele de activitate,diagramele componentelor, diagramele de desfasurare. În metoda HOORA, se decide restrângerea numarului diagramelor la definirea pasilor procesului la aplicarea unei submultimi de diagrame, ceea ce suplimenteaza structurarea acestui aspect.

Metoda HOORA încurajeaza descompunerea ierarhica – Pachetul UML (care la rândul sau poate fi descompus în pachete UML) sustine ierarhii. Ierarhiile vor reprezenta nivele de abstractizare. Fiecare nivel de abstractizare poate fi parte a unui proces similar. Orientarea obiect este, de obicei, privita ca fiind incompatibila cu ierarhiile. Ierarhiile nu sunt neaparat legate de structuri imbricate, în sensul limbajelor de programare. Cu toate acestea ierarhiile sunt necesare pentru a organiza si prezenta procesul de reflectie. Ierarhiile sunt, de asemenea unitati ale configurarii managementului. Parti ale ierarhiei pot fi verificate separat.

Metoda HOORA suporta un context ierarhic - Fiecare sistem poate fi considerat ca o multime de clase: aceste clase sunt static relationate (acestea se cunosc reciproc) si colaboreaza cu fiecare dintre celelalte (dialogheaza unele cu altele). HOORA defineste o cale pentru a utiliza ierarhia, înca strict conforma cu regulile UML (slabe în acest domeniu). Într-o metoda ierarhica, este de preferat sa fie interzisa vizibilitatea între clase prea distante, ceea ce implica existenta unor reguli considerate pentru nivelele ierarhice. Structurile orientate obiect sunt de multe ori incompatibile cu conceptele ierarhice. Este curent acceptata ideea ca toate clasele apartin unei structuri plate (omogene), care nu este restrictionata de nici o regula ierarhica. Într-adevar multe dintre sistemele orientate obiect sunt nestructurate. Conceptele cerute pentru tratarea subiectelor complexe (i.e. identificare claselor care lucreaza împreuna cu o multime bine definita de responsabilitati) nu aplicabila numai la nivelele de detaliu, dar si la cele înalte (în care o arhitectura optimala este mult mai importanta). Se va pretinde si nivelelor înalte de abstractizare sa fie constituite (limitate la un numar rezonabil) de multimi de clase care lucreaza împreuna. Descrierile nivelelor înalte nu este numai o simplificare a unei realitati complexe.

Metoda HOORA enunta dinamicile – Procesele UML dezvolta aspectele dinamice – Procesele UML, de cele mai multe ori dezvolta relatii statice. Metoda HOORA tinde sa prezinte dinamicile cu un model dinamic. Scopul este nu de a gasi toti partenerii posibili (obiecte sunt clase) de care ar putea fi nevoie si carora li se pot aloca responsabilitati ci este acela de a obtine multimea minima a claselor care, lucrând împreuna, satisfac cerintele unui nivel ierarhic particular. Acest model dinamic este, de altfel, consistent cu diagrama de secvente prezentata la interactiunea dintre clase.

Metoda HOORA este bazata pe stare – Starile sunt piese importante ale informatiei care vor fi utilizate pentru explicarea claselor, studiilor de caz, colaborarilor si metodelor. În cazul claselor, tranzitia starilor va corespunde evenimentelor sau operatiilor definite pentru clasa în sine. Dintre alte caracteristici importante ale metodei se disting:

- Sustine dezvoltarea în timp real – Cerintele de timp real sunt la rândul lor tot cerinte software. Utilizarea tehnicilor este legata de o cerinta de timp real particulara. HOORA permite identificarea si împartirea pe categorii a unor astfel de cerinte în timp real. HRT-HOOD acopera tehnicile pentru analiza si validarea acestor cerinte. HOORA, în principal permite colectarea informatiilor de timp real si adnotarea fisierelor HOOD cu astfel de informatii.
- Integreaza complet analiza si proiectarea – analiza si proiectarea sunt activitati distincte care necesita competente distincte; cu toate acestea ambele pot utiliza tehnici de modelare similare. Metoda HOORA descrie o tranzitie catre formatul de proiectare HOOD pentru a capta informatia de analiza într-un proiect initial, cu care, de altfel se porneste (în loc de a se demara cu un proiect pagina alba). Suplimentar se întrevede integrarea cu SDL.
- Este complet integrata cu o abordare a testarii – la fiecare nivel de abstractizare si asociat elementelor relevante ale modelului, se vor defini specificatiile de test. Cazurile de test sunt legate cu studiile de caz si/sau diagramele de secventa.
- Este cooperativa – un proiect consta dintr-un set de sarcini si responsabilitati. Din punctul de vedere al metodei HOORA, acesta consta din necesitatea actualizarii diferitelor nivele ierarhice (acestea fiind de asemenea unitatile configurarii managementului). O abordare colaborativa necesita posibilitatile schimburilor de unitati, prin retea, cu pastrarea trasei, verificarea pe intrare si pe iesire, la care se mai adauga rearanjarea nivelelor ierarhiei, sustinuta explicit.
- Sustine trasabilitatea si permite generarea de documente.

Descrierea succinta a procesului HOORA se prezinta în continuare:

1. Identificarea cerintelor textuale
   - identificarea cerintelor textuale în limbajul utilizatorului
2. Analiza studiilor de caz
   - identificarea studiilor de caz, utilizarea diagramelor studiilor de caz si text
3. Elaborarea scenariilor
   3.1. identificarea scenariilor relationate cu studiile de caz , editarea rutelor de baza a actiunilor reprezentând însiruirea principala si rutele alternative pentru mai putin traversatele cai si pentru conditiile de eroare.
   3.2. Elaborarea scenariului în diagrame de secvente, cu identificarea componentelor cerute (clasele domeniului) pe pachete si servicii pe care trebuie sa le ofere.
   3.3. Alocarea cerintelor elementelor modelului.
4. Modelul dinamic
   4.1. Elaborarea modelului dinamic pentru fiecare dintre clasele identificate, cu prezentarea claselor care vor lucra împreuna cu alte clase
   4.2. Revizuirea modelului pentru asigurarea faptului ca rolurile sunt clare si neambigue pentru toate clasele si ca modelul este sigur pentru modificari ulterioare
   4.3. Identificarea de noi clase de abstractizare / verificarea cu modelul static si cu modelul pachetului / refactorizarea
5. Modelul static
   5.1. Elaborarea modelului static pentru fiecare clasa identificata, cu prezentarea statica a relatiilor dintre clase
   5.2. Revizuirea modelului, realizarea analizei de robustete
   5.3. Verificarea cu modelul dinamic si modelul pachetului/refactorizare
6. Modelul pachetului
   6.1. Elaborarea dependentelor dintre pachete, rezumarea relatiilor statice si dinamice
   6.2. Revizuirea nivelelor de abstractizare/refactorizarea
7. Modelul starii
   7.1. Este utilizat sub forma diagramei de stare pentru a vizualiza comportamentul (dependent de timp) aferent unei clase
8. Trasabilitatea
   8.1. Alocarea cerintelor textuale unei clase, operatii si relatii.

Conceptele si entitatile HOORA

Clasa abstracta

O clasa care nu poate fi instantiata. Instantele unei clase date nu exista. Instantele unei subclase ale unei clase date pot exista.

Antonim: clasa concreta

Clasele abstracte sunt o modalitate de a factoriza comportamente si caracteristici comune. Actualmente, utilizarea reala a claselor abstracte nu este uzual vizibila pe diagrama de care apartine ci numai în parti ale modelului care nu necesita proprietati specifice ale subclaselor. Aceste parti depind exclusiv de clasele abstracte. Prin aceasta modalitate gradul de dependenta dintre elemente este redus si se accentueaza modificabilitatea. Exemplul de mai jos este elocvent pentru utilizarea acestui concept:

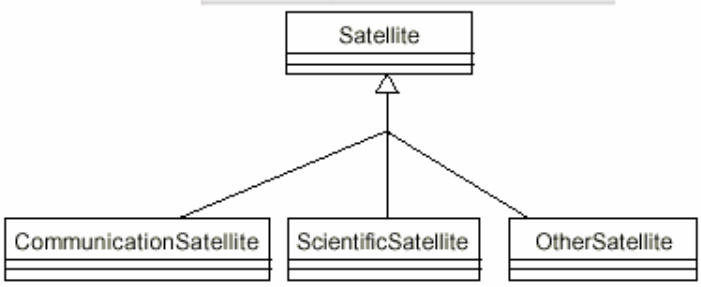


Figura 4.9 Clasa abstracta

Se pune problema unei clase Satelit cu subclasele Communication Satelite ScientificSatelite si OtherSatelite. Este evident ca fiecare instanta a clasei Satelite este de asemenea o instanta a uneia dintre cele trei subclase. O parte dintre utilizatori nu vor avea nevoie explicita de a specifica

diferitele subclase de sateliti si în consecinta vor depinde numai de Satelite. Lipsa subclasei OtherSatelite transforma clasa Satelite dintr-o clasa abstracta ceea ce conduce la un model mai greu de înteles si mai greu de întretinut.

Operatia abstracta

Operatia abstracta este operatia care este închisa implementarii. O implementare trebuie furnizata de o subclasa concreta. Numai clasele abstracte pot avea operatii abstracte. Daca exista cel putin o operatie abstracta într-o clasa, devine fara sens existenta unor instantieri ale acelei clase si respectiva clasa trebuie sa fie abstracta.

Antonim: operatie concreta

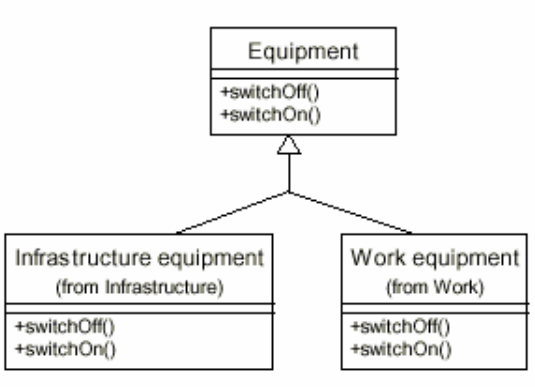


Figura 4.10 Operatia abstracta

SwitchOn si SwitchOff pot fi operatii care sunt specificate numai la nivelul Equipment. Acestea pot da comportamentul detaliat numai într-o subclasa concreta.

Use case abstract

Use case abstract nu este un use case pur în sensul actori primari care au un scop.

Antonim: use case concret

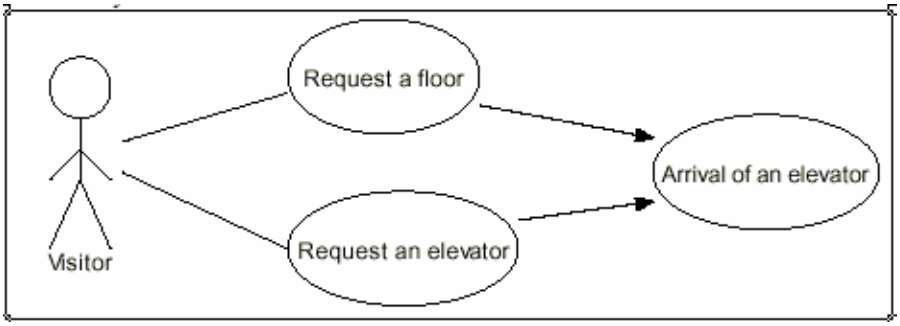


Figura 4.11 Use case abstract

Diagrama de activitate

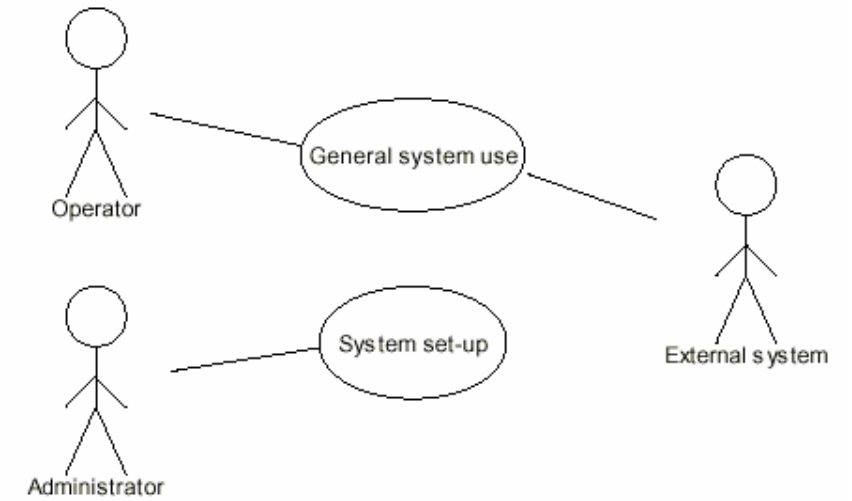
Diagrama de activitate este un element de prezentare reprezentând o multime de activitati conectate împreuna. Este definita în UML standard. Nu este parte a procesului HOORA.

Actor

Este un element al modelului reprezentând o entitate care este în afara limitelor sistemului. Actorii sunt similari rolurilor. Aceste roluri pot fi realizate de o persoana fizica, sau de un sistem extern.

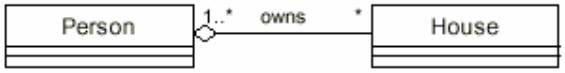
Identificarea actorilor este o decizie importanta, deoarece aceasta determina frontierele sistemului Actorii sunt parte a modelului use case. Actorii apar, de asemenea, si în modelul dinamic atunci când se decide aspra claselor care pot comunica între ele. Analiza robustetii starteaza cu actorii care limiteaza astfel tipurile de clase cu care actorii sunt abilitati sa comunice. Actorii trebuie totdeauna sa interactioneze cu asa numitele clase de frontiera. Exemplificarea acestui concept este data în figura de mai jos.

Figura 4.12. Actorul



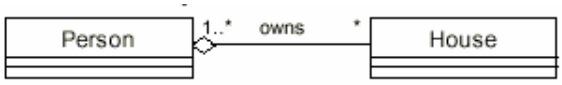
Agregatul
O clasa care reprezinta întregul într-o relatie de agregare

Figura 4.13. Agregatul



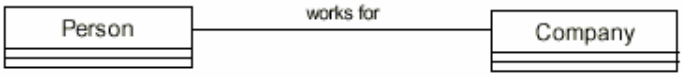
În exemplu Person este agregatul.
Agregarea
O forma de asociere reprezentând partea dec cuprindere a relatiei (are o) care este mai slaba decât o compozitie. La disparitia întregului, partea nu dispare. Partea poate exista si fara întreg. Întregul poate exista si fara parti. Partile pot sa se mute de la un întreg la altul.

Figura 4.14. Agregarea



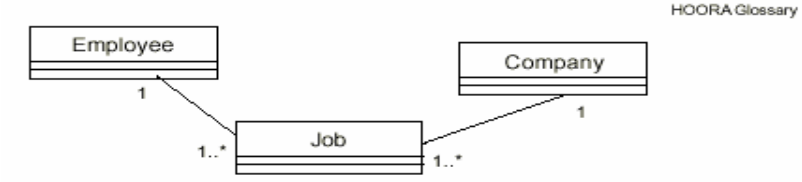
Modelul analiza
Este modelul produs pe parcursul fazei de analiza, care accentueaza asupra ceea ce sistemul face/trebuie sa faca si nu asupra modului în care aceasta este solutionata. Un model analiza trebuie sa nu contina nici un element specific solutiei. Numai cerintele actuale vor conduce modelul.
Antonim: modelul proiectului
Este uzuala crearea unui model analiza separat deoarece acest model scoate în evidenta numai necesitatile functionale de baza. Modelul proiectare va scoate în evidenta baza evolutiilor tehnologice.
Asociatia
Este un element al modelului reprezentând relatia dintre doua sau mai multe clase. Spre deosebire de agregare cele doua parti sunt considerate pe un nivel egal. Clasa A nu este subordonata clasei B sau invers. Clasa A nu dispare la disparitia clasei B sau invers. Agregarea si compozitia sunt forme speciale ale asocierii. Toate asocierile realizeaza relatii statice 'cunoscute' între obiecte. Aceste relatii statice sunt cerute pentru a valida interschimbul (dinamic) al mesajelor între obiecte. Cele mai multe asociatii sunt binare, i.e. între doua clase. Asociatiile n-are sunt exceptii.

Figura 4.15. Asociatia



Clasa asociatiei
O asociere este de asemenea o clasa. Fiind o asociere, aceasta reprezinta o relatie semantica între clase. Fiind o clasa, aceasta poate avea operatii, atribute si relatii statice (asociere, agregare, compozitie, mostenire). Un exemplu în acest sens este dat în figura de mai jos:

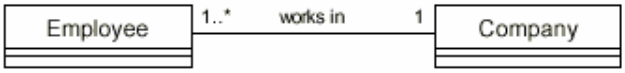
Figura 4.16. Clasa asociatiei



Multiplicitatea asocierii

Multiplicitatea informatiei legata de o asociere prezinta modul în care mai multe instante ale unei clase A pot fi legate cu instantele clasei B. Multiplicitatea informatiei poate fi legata atât pentru finele asocierii, cât si cu relatiile de agregare si compozitie.

Figura 4.17. Multiplicitatea asociatiei



Rolul asocierii

Este un element de model care descrie contextual relatiile, în contextul unei colaborari specifice. Rolurile asocierii sunt reprezentate într-o diagrama de colaborare. Rolul unei asocieri poate fi referit cu o asociere de baza. Rolul asocierii leaga rolurile sortatorului, precum si sortatorii.

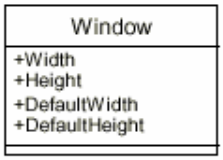
Eveniment de apel asincron

Este un eveniment de apel, legat cu un mesaj sau tranzitie de stare , reprezentând o cerere de la un obiect care o transmite pentru a se realiza o actiune. Obiectul care transmite nu asteapta rezultatul sau sincronizarea

Atribut

Este un element al modelului care descrie un slot dintr-o clasa: atributele au un tip. Acest tip poate fi o clasa; poate fi de asemenea un tip de baza. Atributele vor fi utilizate numai pentru valorile fara identitate i.e. tipuri de date orientat pe valori. Toate tipurile de date de baza sunt orientate pe valori. Clasele definite de specificator pot fi, la rândul lor orientate pe valori. Pentru obiecte cu identitate, se va utiliza o asociere (uzual o agregare sau o compozitie). În acest caz, poate fi important sa se considere relatia în ambele sensuri. Pentru tipurile de date orientate pe valori aceasta nu este relevanta. Se va face distinctia dintre instanti si atributele clasei. Instantele atributelor se aplica instantelor. Atributele clasei se aplica clasei.

Figura 4.18. Atributul



Clase de frontiera

Este o clasa utilizata la modelarea comunicatiei între sisteme aflate în afara mediului (actori) si activitatea interna acestora.

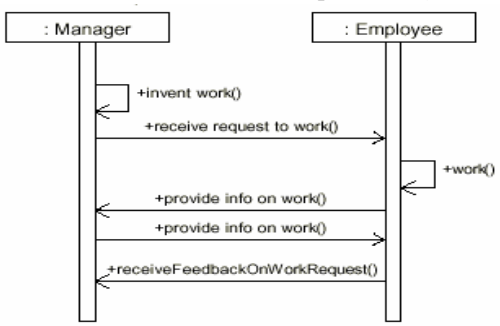
Obiectele din afara comunica cu obiectele din interior. Clasele de frontiera sunt introduse pentru a lasa libera comunicarea oricarui obiect cu un altul. Actorii pot doar sa între în dialog (discutie) cu clasele de frontiera, iar la rândul lor clasele de frontiera pot doar dialoga cu actorii, cu clasele de frontiera si cu alte clase de frontiera. Aceasta tehnica este parte a analizei de robustete si asigura o alocare consistenta a responsabilitatilor claselor.

Clasele de frontiera încapsuleaza cunostinte despre actorul cu care comunica si despre protocolul de pe interfata. Aceste clase pot fi aplicate în contextul interfetei utilizator, al interfetei sistem, sau a interfetei echipamentelor.

Evenimentul apel

Evenimentul apel este un element al modelului, legat de un mesaj sau de o tranzitie de stare, care capteaza evenimentul receptionarii unui apel. În HOORA, este asigurat ca toate evenimentele dintr-o diagrama de secvente si dintr-o diagrama de stari, devin asociate cu o operatie definita pentru receptia unei clase. Chiar pentru conditii care trebuie verificate permanent, pentru a raspunde la semnalele externe, sau pentru a astepta o cuanta de timp, este asigurata o operatie clara care se declanseaza (de o alta clasa, un actor extern, sau chiar de clasa însasi via un apel intern).

Figura 4.19. Evenimentul apel



Relativ la exemplul dat trebuie notat faptul ca sagetile reprezinta controlul fluxului si nu fluxul de date. Cele doua sageti 'provide info on work' necesita informatii de retur, dar aceasta nu este prezentat în diagrama de secvente. Sagetile pornesc de la obiectele declansatoare (apelante) catre obiectele declansate (apelate). Numele operatiei asociate evenimentului trebuie sa aiba sens pentru serviciul clasei receptoare. Împreuna cu mesajul, se va indica si tipul mesajului de control al fluxului: sincron sau asincron.

Clasa

Clasa este un element al modelului care reprezinta o multime de obiecte care partajeaza aceleasi atribute, operatii, relatii, comportamente. O clasa reprezinta un concept care trebuie modelat. În cadrul modelului de analiza, o clasa reprezinta domeniul conceptului care trebuie modelat. Clasele vor reprezenta entitatile lumii reale. Lumea reala nu înseamna neaparat lume tangibila. O regula, anumite combinatii de cunostinte pot fi exemple de astfel de clase. O clasa poate fi orice, care atunci când este identificata duce la cresterea calitatii modelului. Un astfel de model descrie un domeniu ca o multime de obiecte care sunt relationate unele cu altele (modelul static) si care interactioneaza unele cu altele (modelul dinamic). O clasa este o multime de atribute, operatii, relatii si comportament, a caror combinatii au sens în domeniul modelului.

Diagrama clasei

Este un element de prezentare care este o reprezentare a perspectivei statice a unei parti a modelului si care prezinta combinatia de clase, asocieri, agregari, compozitii si relatii de mostenire. În sensul UML, o diagrama a clasei poate contine si module. În sensul HOORA diagrama de pachet si diagrama de clasa sunt separate, astfel încât diagramele de clasa HOORA nu pot contine module, iar diagramele de module HOORA pot contine numai module.

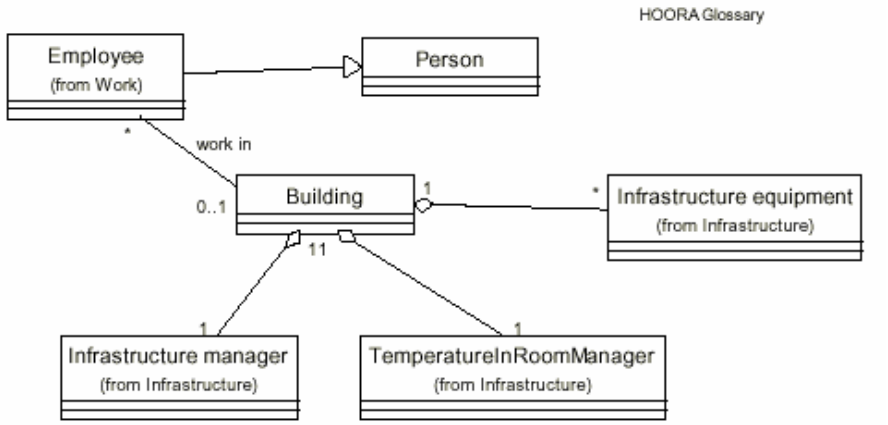


Figura 4.20
Diagrama clasei

Diagramele claselor individuale sunt create pentru a scoate în evidenta clasele specifice si relatiile statice ale acestora. Este decizia modelorului de a reprezenta mai multe concepte pe o diagrama de clasa, sau de a crea doua sau mai multe astfel de diagrame. În mod normal, fiecare diagrama de clasa reprezinta o submultime a domeniului la un nivel particular de abstractizare.

Sortator

Sortatorul reprezinta un nume general de clasa, actor, tip de date, use case, sau alte concepte UML care nu sunt în mod specific sustinute de HOORA (componente, interfete, noduri, semnale, subsisteme).

Rolul sortatorului

Este un element care reprezinta o parte a colaborarii care descrie rolul jucat de catre un participant la respectiva colaborare. Un rol al sortatorului apare în scenariul reprezentat de diagrama de secvente si/sau diagramele de colaborare. Un rol al sortatorului este similar cu o instanta, dar nu este egal cu o instanta individuala, acesta reprezentând o multime de instante care pot fi parte a colaborarii. Un astfel de rol ofera un sortator, care se va numi, pentru respectivul rol sortator de baza. Sortatorul are multiplicitate, iar rolul sortatorului poate fi anonim sau denumit.

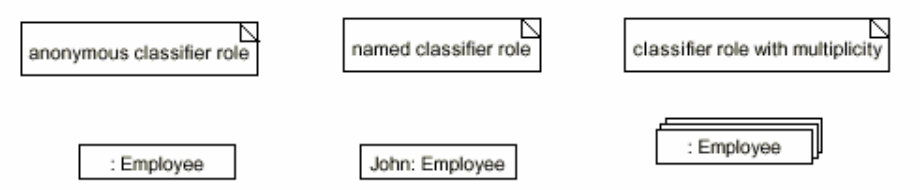


Diagrama de colaborare
Este un element de reprezentare a unui scenariu ca o multime de mesaje între rolurile sortatorului legate de rolurile asocierilor.

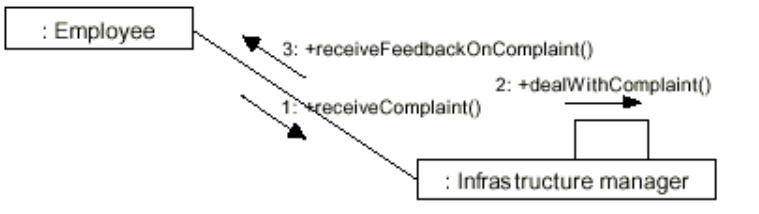


Figura 4.21. Rolul sortatorului

Diagramele de secventa si diagramele de colaborare reprezinta informatii similare. Dimensiunea temporala este explicit prezentata în diagramele de secventa. Într-o diagrama de colaborare, informatia este disponibila prin mesajele numerotate. Rolurile asocierilor sunt explicit enuntate într-o diagrama de colaborare. Aceste informatii nu sunt disponibile în diagramele de secventa. O diagrama de colaborare este legata cu sortatorul al carui scenariu se elaboreaza, i.e. cu clasa sau cu user case.

Comentariu
Este un element al modelului care reprezinta o adnotare care poate fi atasata de la zero la mai multe elemente ale modelului. Un comentariu poate fi adnotat pentru orice tip de element al modelului. Comentariile sunt utilizate pentru a clarifica o multime de elemente ale modelului pe diagrame. Comentariile se mai utilizeaza, de asemenea, pentru scopuri mai specifice, adica pentru a reprezenta pasi în scenariu (partea dreapta a textului unei diagrame de secventa) sau pentru a reprezenta cerinte.

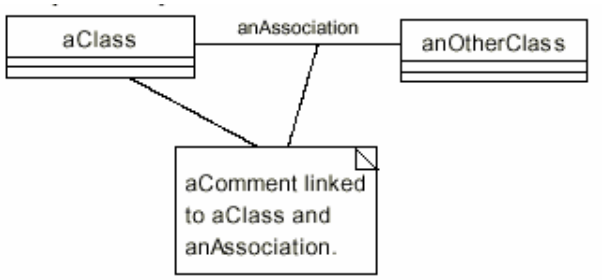


Figura 4.22. Comentariul

Diagrama de componenta
O diagrama de componenta este un element de prezentare a unei multimi de activitati conectate împreuna. Acest tip de diagrama este definit în UML standard si nu face parte din procesul HOORA

Asocierea de comunicatii
Este un element al modelului care reprezinta conexiune dintre un actor si un use case din use case-ul modelului. În UML, o asociere de comunicatii este de asemenea posibila si are loc între noduri (elemente care nu sunt acceptate în HORA). Informatia asociatiei de comunicatii poate fi derivata din informatia evenimentului extern.

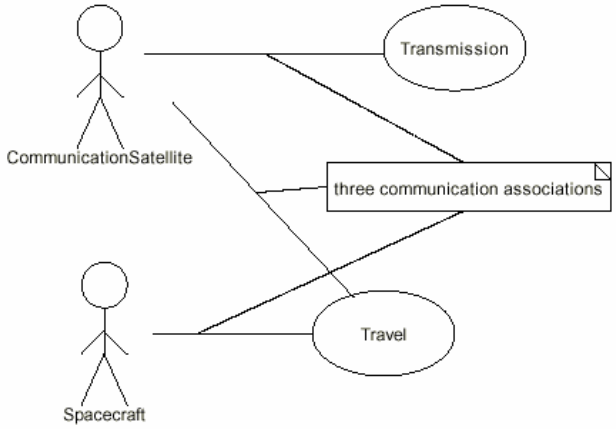
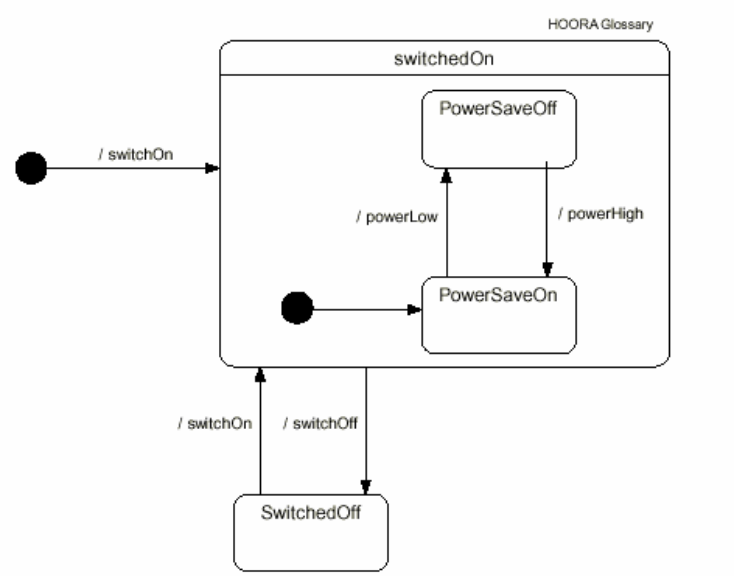


Figura 4.23. Asocierea de comunicatii

Compozita
Este o clasa care reprezinta întregul unei compunerii de relatii.
Stare compozita
Este o stare care consta din mai multe alte stari. O stare compozita este rafinata fie în substari secventiale fie în substari concurente.

Figura 4.24. Starea compozita

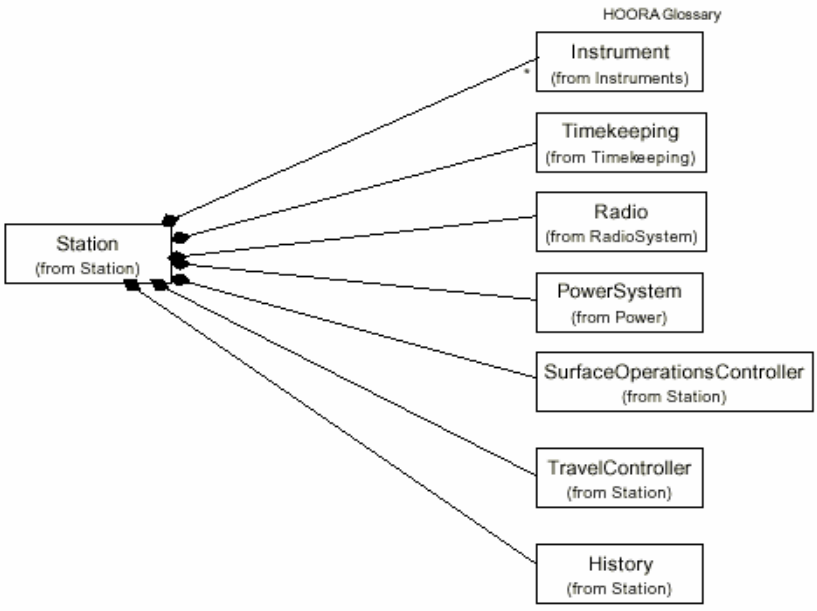


Starile pot fi organizate de o maniera ierarhica si suporta abstractizarea. Fiecare stare compozita poate avea stari de start si de final separate.
Compunerea
O forma a unei asocieri reprezentând o relatie de tipul are un care este mai tare decât o agregare caracterizata printr-o puternica relatie de apartenenta si coincidenta a timpului de viata a partii fata de întreg. Pentru relatiile de compunere se aplica urmatoarele reguli:

- O parte nu poate exista fara întreg. O parte poate fi numai apartinatoare unui întreg.
- La stergere unui întreg, aceasta trebuie sa se propage pâna la stergerea tuturor partilor sale
- Este permis fie ca partile sa fie sterse dintr-un întreg fie sa nu fie (si adaugate la un alt întreg), în orice moment înainte de stergerea întregului. Întregul nu mai are raspunderea timpului de viata a fiecarei parti în parte, oricare dintre acestea putând fi stearsa pe aceasta cale.

Daca una dintre aceste reguli nu se aplica, atunci trebuie aleasa una dintre relatiile de agregare sau de asociere.

Figura 4.25. Compunerea



Aceasta este o relatie mai tare decât agregarea. Partea nu are semnificatie daca nu este conectata la întregul de care apartine.

Concret

Este un element de model care poate fi instantiat: o clasa, un use case, o operatie.

Antonim: abstract

Clasa control

Clasele de control formeaza liantul dintre clasele de frontiera si clasele de entitati. Clasele de control sunt clase care încapsuleaza comportamentul specific solicitat, asa cum este acesta captat într-unul sau mai multe use case. Clasele de control coordoneaza activitatile. Clase de control sunt introduse pentru a nu lasa clasele de frontiera sa interactioneze direct cu clasele de entitati. Actorii nu au dialog cu clasele de control, precum nici clasele de control nu au dialog cu actorii. Clasele de control dialogheaza cu alte clase de control, cu clasele de entitati si cu clasele de frontiera. Aceasta tehnica este parte a analizei de robustete si asigura alocarea consistenta a responsabilitatilor la clase.

Delegarea

Abilitatea unui obiect de a propaga o cerere unui alt obiect (care se va numi delegat) delegarea trebuie sa fie privita ca o alternativa (cu mult mai putin structurata, dar mai puternica) la mostenirea din extensiile limbajelor OO. Actualmente este o tehnica de a reduce dependenta puternica implicata uzual de catre mostenire.

Delegarea este o tehnica de a crea clase orientate ca o responsabilitate stricta care pot, deci, fi asociate (prin asocieri, agregari sau compuneri) mai simplu si mai flexibil cu alte clase. Mostenirea este mai automata (operatiile nu trebuie sa fie propagate; propagarea decurge automat). Mostenirea este mai putin flexibila (clasele sunt determinate la momentul cererii; ulterior modificarile nu mai sunt posibile; o instanta nu poate schimba clasa pe care se bazeaza).

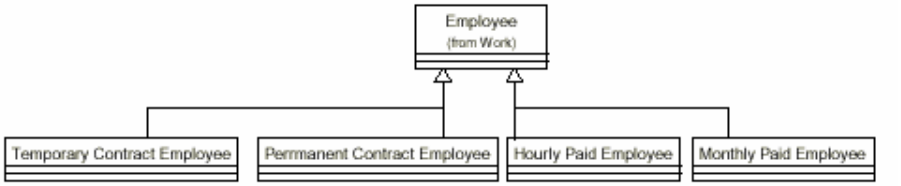
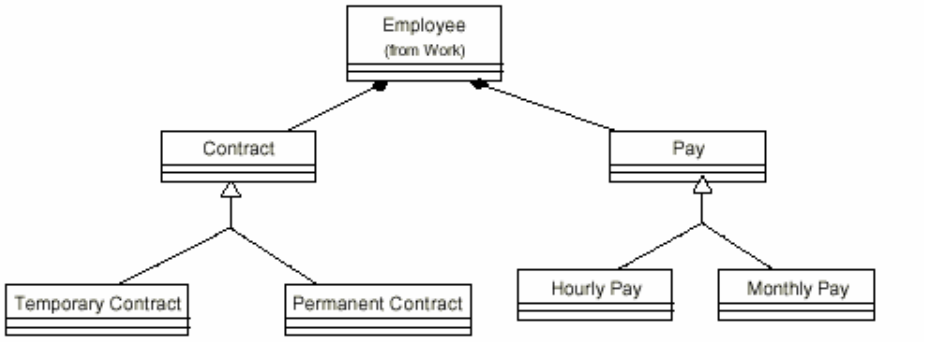


Figura 4.26.
Delegarea



Dependenta

În UML, este un element al modelului care reprezinta un tip generic de relatie care poate fi creat între oricare doua elemente ale modelului. În HOORA, este utilizata pentru a crea module de dependente între modulele din diagramele de module si dependente ale use case între use case din diagramele use case.

Diagrame de desfasurare

O diagrama de desfasurare este un element de prezentare care reprezinta procesarea la rulare a nodurilor si a instantelor componentelor acestora. Este definita în UML standard si nu este parte a procesului HOORA.

Modelul proiect

Modelul produs pe perioada fazei de proiectare , care accentueaza modul în care un sistem trebuie sa-si împlineasca cerintele. Un model proiect contine elemente specifice solutiei. Un model proiect adauga aceste elemente la modelul de analiza original.

Antonim: model de analiza
Este uzuala crearea unui model de analiza separat, deoarece acest model se va dezvolta numai bazat pe necesitatile functionale. Modelul proiect se va dezvolta bazat pe evolutiile tehnologice. HOOR furnizeaza recomandari specifice pentru tranzitia de la modelul de analiza la cel proiect si utilizeaza notatii specifice HOOD si SDL:

Discriminator
Este un element al modelului care este o cale pentru a distinge între mai multe subclase într-o relatie de mostenire.

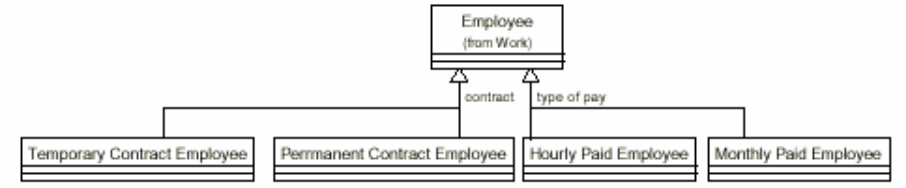


Figura 4.27 Discriminatorul

De cele mai mult ori discriminatorii pot fi reformulati utilizând tehnicile de delegare. Un avantaj suplimentar este acela ca modelul bazat pe delegare poate fi mult mai simplu translatat într-un proiect.

Restrictii de durata
Un tip de cerinte de timp real care indica ca durata unui proces corespunzatoare executiei unei functii (sau a unei secvente de functii) trebuie realizata într-un timp limitat. Limbajele vor trebui astfel sa dispuna de mecanisme de TIMEOUT.

Dependenta dinamica
Dependente dinamice sunt dependente între clase. Acestea sunt derivate din mesaje. Aceste mesaje sunt evidentiate în diagramele de colaborare si în diagramele de secvente. Un mesaj de la un rol al sortatorului al clasei A catre rolul sortatorului clasei B va cauza o dependenta dinamica, care arata ca clasa A depinde dinamic de clasa B.

Modelul dinamic
Multimea tuturor elementelor modelului care capteaza diferite interactiuni între clasele din model, i.e. care reprezinta dependentele dinamice. Diagramele modelului dinamic vizualizeaza aceste informatii ca pe o multime consistenta de diagrame. Un model dinamic are semnificatia de a creste întelegerea contextului dinamic al fiecarei clase. Un model dinamic trebuie, de asemenea, sa fie comparat cu modelul static pentru a identifica dependentele dinamice altele decât cele date de legaturile corespunzatoare din modelul static. Relatiile (structurale) statice din modelul static valideaza dinamicile sistemului. În afara acestora, instantele A si B nu se 'cunosc' (i.e. au o relatia statica cu) între ele si, în consecinta, nu vor fi capabile sa comunice (i.e. sa aiba o dependenta dinamica). Modelul dinamic consta din clase si mesajele care se transmit între acestea.

Diagrama modelului dinamic
Diagrama modelului dinamic pentru o clasa este un element de prezentare care vizualizeaza: o clasa, toate mesajele trimise de aceasta clasa altor clase si toate mesajele trimise de alte clase catre clasa respectiva. O diagrama a modelului dinamic, pentru o clasa data vizualizeaza, astfel o clasa si toate dependentele sale dinamice. Diagrama modelului dinamic vizualizeaza modul dinamic si permite compararea cu modelul static.

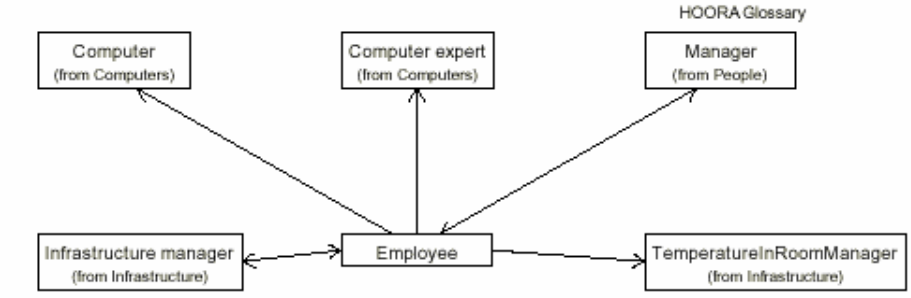


Figura 4.28. Diagrama modelului dinamic

Diagrama modelului dinamic are formatul diagramei de clase. Conexiunile din diagrama sunt asocieri stereotipate si acestea reprezinta dependentele dinamice. Mesajele responsabile cu dependentele dinamice sunt legate de aceste asocieri.

Clasa entitate
O clasa utilizata pentru captarea evenimentelor din lumea reala si în continuare utilizata pentru a modela informatia si comportamentul asociat. Clasele entitate sunt uzual persistente. Clasele entitate trebuie sa nu comunice direct cu obiectele externe (actori sau clase de frontiera). În schimb, acestea trebuie sa comunice numai clasele de control si alte clase entitate.
Actiunea de intrare
Este un element de model care identifica actiunea realizata la intrarea într-o stare. O astfel de actiune de intrare este optionala pentru fiecare stare.
Eveniment
Este un element al modelului care este specificarea unei ocurente în timp si spatiu legata de un mesaj sau de o tranzitie de stare. Evenimentele cauzeaza tranzitiile de stare. Evenimentele sunt transmise între obiecte, asa cum se prezinta în diagramele de colaborare si în diagramele de secventa.

În concordanta cu standardele UML exista patru tipuri de evenimente:

- Evenimente apel (declansate de receptarea unei cereri de invocare a unei operatii)
- Evenimente de modificare (declansat de satisfacerea unei conditii booleene)
- Eveniment semnal (declansat de cercetarea unui semnal extern)
- Eveniment timp (declansat de praguri temporale)

Pentru a nu se crea confuzii si pentru a asigura ca scenariile produse vor produce rezultate uzuale, HOORA se limiteaza la a avea numai evenimente de apel.
Actiunea exit
Este un element de model care identifica actiunea realizata la iesirea dintr-o stare. O astfel de actiune exit este optionala pentru fiecare stare.
Eveniment extern
Este un eveniment care este o specificare a unei ocurente din afara sistemului la un moment dat de timp, identificabil, într-un punct determinat si careia sistemul trebuie sa-i raspunda. Un eveniment extern i se poate atribui unui actor.
Stare finala
Este o stare speciala dintr-o stare compozita care indica faptul ca executia acestei stari compozite s-a încheiat. O stare finala pe un nivel înalt se poate referi la distrugerea obiectului. Este posibil, de asemenea de a avea mai multe stari finale într-o stare compozita.
Generalizarea
Este un element de model care reprezinta relatia dintre un element de model mai general si un element de model mai specific. În HOORA, clasele si use case pot avea relatii de generalizare.
Antonim: Specializarea
Generalizarea semnifica faptul ca toate caracteristicile superclaselor sunt mostenite de subclasa. Pentru o clasa, aceasta include operatii, asocieri, agregari, compuneri, atribute. Mai important, acesta include, de asemenea si pre- sau postconditii definite pentru operatiile clasei. Regula este ca aceasta trebuie sa fie disponibila sa aibe o instanta actuala a subclasei în care se presupune existenta unei instante a subclasei. La momentul la care se apeleaza o operatie de catre superclasa, este asigurat ca preconditiile sunt satisfacute si se poate conta ca post conditiile sunt satisfacute dupa apelul operatiei. Astfel, subclasa trebuie sa aiba aceleasi preconditii sau mai putine. Subclasa nu trebuie sa impuna preconditii suplimentare. Similar, postconditiile subclasei trebuie sa fie cel putin de nivelul celor specificate în superclasa; pot fi mai mult. Aceasta este de fapt semnificatia unei relatii de tipul is-a. Poate avea loc o mostenire numai daca is-a este satisfacuta. Regula is-a este denumita si regula de substitutie a lui Liskov. În procesul generalizarii se pot produce clase ca subclase ale altora. Se poate introduce o clasa abstracta ca superclasa a amândurora. Mostenirile multiple implica faptul ca o subclasa mosteneste din mai multe superclase.

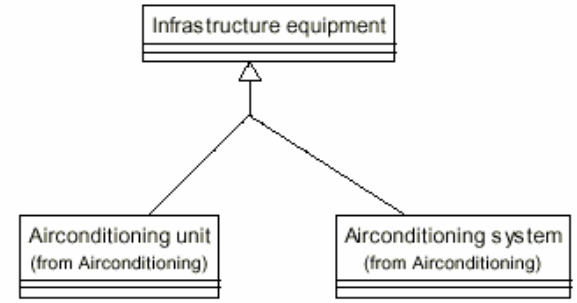


Figura 4.29. Generalizarea

Generalizarea use case implica ca un use case mosteneste toate descrierile comportamentale ale use case parinte, incluzând comunicatia asociata actorilor.

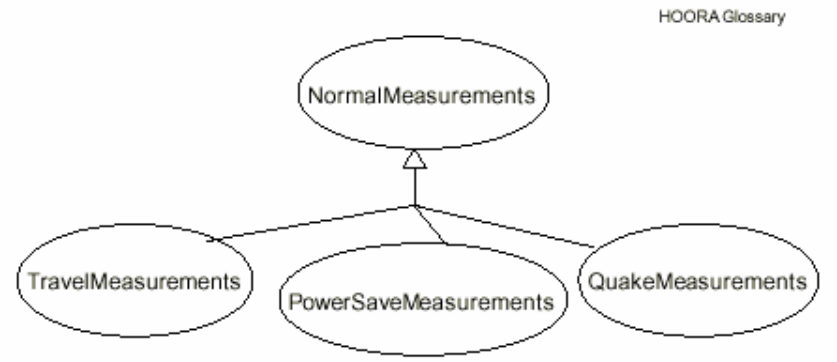


Figura 4.31. Masuratori

Conditia de garda

Este un element al modelului reprezentând o conditie care trebuie satisfacuta pentru a valida declansarea unei tranzitii de stare.

Ierarhie

Este o strategie necesara pentru mascarea detaliilor si gestiunea complexitatii. Se realizeaza, prin partitionare si organizarea în nivele de abstractizare. Ierarhia HOORA consta dintr-o multime de module imbricate. Modulele reprezinta nivelele de abstractizare. Pe fiecare nivel, sunt definite clase, precum si relatiile statice si aspecte dinamice. Pentru fiecare nivel sunt esentiale cerintele de tarsabilitate. Actorii si use case sunt identificati numai la nivelul de vârf. Pentru a restrânge accesul dintr-un nivel ierarhic la oricare altul sunt folosite metrici ierarhice.

Metrici ierarhice

Metricile ierarhice sunt mijloace de a furniza informatii cantitative care încurajeaza ierarhiile revizuite în care elementele modelului si elementele modelului care le acceseaza sunt strâns legate între ele. Pentru toate dependentele dintre elementele modelului A si B (i.e. asocieri, agregari, compozitii, mosteniri, mesaje) se va calcula:

- numarul nivelelor inferioare
- distanta totala (numarul de nivele superioare si numarul de nivele inferioare) pentru a trece de la A la B.

Asocierile, agregarile si compozitiile sunt bidirectionale si sunt legate de dependentele dinte A si B precum si de dependentele dintre B si A. Pentru mosteniri se va decide daca A mosteneste pe B, caz în care se va crea o dependenta a lui A catre B dar nu si invers. Pentru un mesaj se considera ca A apeleaza B, ceea ce duce la crearea unei dependente A a lui A catre B dar nu si invers.

În fiecare modul si submodul din interiorul unui proiect, trebuie calculate urmatoarele metrici:

- distanta maxima
- numarul maxim de nivele superioare
- numarul maxim de nivele inferioare
- media distantei
- media numarului de nivele superioare
- media numarului de nivele inferioare

Aceste metrici pot fi utilizate pentru a localiza punctele în care nivelele ierarhice sunt tare distorsionate, astfel încât se vor putea lua decizii de refactorizare (mutarea unor clase în alte module; crearea/stergerea de module; crearea de clase suplimentare pentru a servi ca intermediari între module care sunt prea distante). HOORA defineste aceste metrici. Regula normala este ca numarul maxim de nivele inferioare este 1. Numarul maxim de nivele superioare este nedefinit. Aceasta implica ca un obiect are vizibilitatea fratilor sai (un nivel superior peste propriul modul apoi un nivel inferior catre frate), catre toti stramosii sai (numar infinit de module superioare si un nivel inferior pentru a vedea continutul unui modul dat), catre propria filiatie (un nivel inferior), dar nu si catre nivele inferioare. Metricile ierarhice sunt importante pentru a mentine o structura sanatoasa în modelul HOORA. Metricile ierarhice pot de asemenea fi utilizate pentru a asigura ca o constructie a unui model proiect initial din modelul de analiza nu este prea dificil.

HOOD
HOOD este metoda standard ESA (European Space Agency) pentru proiectul arhitectura si proiectul de detaliu. HOOD semnifica Hierarchical Object-Oriented Design. HOORA descrie o tranzitie de la analiza la proiect, în care proiectul face apel la notatiile HOOD. Regulile ierarhiei HOOD sunt mult mai stricte ca cele ale HOORA. Generarea unui proiect HOOD sintactic corect necesita propagarea ascendenta si descendenta a interfetelor pentru a permite ca doua clase din pachete diferite sa se poata apela una pe cealalta. O cale de a limita aceasta este de a seta valorile metricilor ierarhiei HOORA pentru a fi în acord cu regulile, HOOD.

HOORA
Este un proces care poate fi aplicat pe perioada fazei de analiza a unui proiect software. Procesul consta din pasi care fac uz de notatia UML.

Identitate
Un obiect este inerent clar daca acesta este facut distinctibil de alte obiecte. Doua obiecte diferite pot avea aceleasi valori ale atributelor, chiar aceleasi valori ale asocierilor. Daca sunt doua obiecte atunci aceasta distinctie trebuie pastrata. Aceasta este posibil deoarece obiectele au o identitate.

Starea initiala
Este o stare speciala în interiorul unei stari compozite care indica faptul ca executia acestei stari compozite va fi declansata în aceasta stare. Starea initiala pe nivelul de vârf se refera la punctul la care obiectul îsi starteaza ciclul de viata. Nu este posibil ca una si aceiasi stare compozita sa aibe doua stari initiale.

Instanta
Este un element al modelului care reprezinta entitatea individuala: un obiect, care este rezultatul instantierii unei clase.

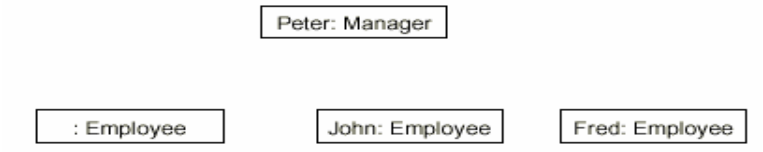


Figura 4.32. Instanta

Legatura
Este un element al modelului care reprezinta o instanta a unei asocieri: conexiunea între doua instante reprezentând asocierea dinte aceste doua instante

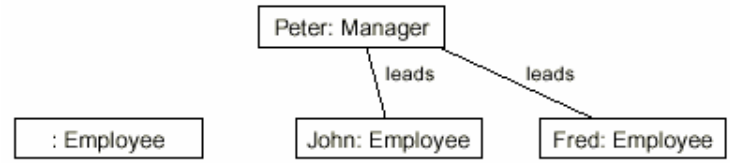


Figura 4.33. Legatura

Mesajul
Este un element al modelului care reprezinta trimiterea unui semnal de la un obiect (emitatorul) la un alt obiect (receptorul). În HOORA un mesaj este totdeauna legat cu un apel al unei operatii definite pentru o clasa (i.e. eveniment apel). Mesajele pot fi vizualizate în diagramele de secvente, diagramele de colaborare si diagramele modelului dinamic. Odata cu mesajul se indica si tipul mesajului de control al fluxului: sincron sau asincron.

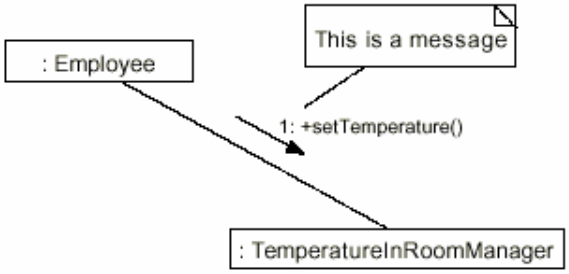


Figura 4.34 Mesajul

Model
Este o abstractizare completa a unui sistem

Element al modelului
Este un element care abstractizeaza informatia sistemului care trebuie modelat. Un element al modelului este în proprietate unui pachet. Un element al modelului nu trebuie sa fie confundat cu un

element de prezentare al carui unic scop este de a prezenta informatia modelului de o maniera inteligibila.

Asocierea n-ara

Asocierea n-ara este o asociere între trei sau mai multe clase. În general este recomandat sa se testeze acoperirea unei asocieri n-are într-o clasa suplimentara si sa se seteze o asociere binara.

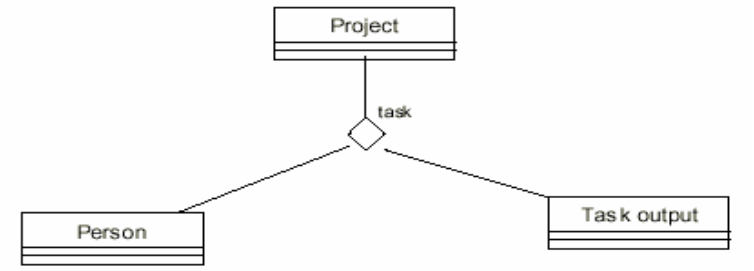
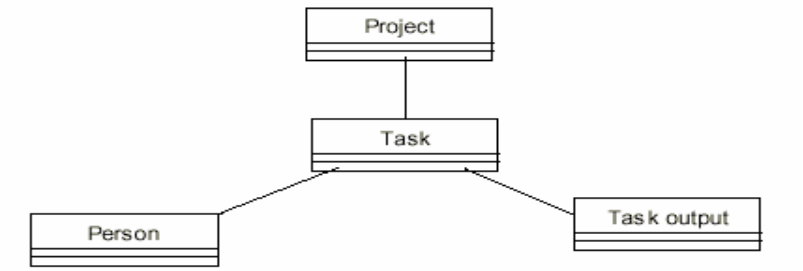
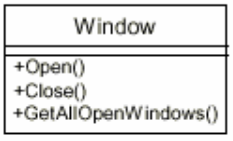


Figura 4.35 Asocierea n-ara



Cerintele nefunctionale

Acestea sunt cerintele care nu specifica un comportament functional. În mod uzual aceste cerinte sunt legate de procesul utilizat pentru rularea proiectului (proiectul si restrictiile implementarii, cerintele de securitate, cerintele de portabilitate, configurarile software, cerintele, de calitate, cerintele de întretinere a software, cerintele de personal etc.). În mod uzual modelarea nu clarifica inteligibilitatea cerintelor nefunctionale aplicate tuturor obiectelor. Operatia este un element de model care reprezinta specificare unei transformari sau cereri pentru care obiectul poate fi apelat pentru a o executa. O operatie are un nume si o lista de parametrii formali. O operatie a unei clase date C poate redefini o operatie a unei superclase al lui C. Operatia din superclasa poate fi sau poate sa nu fie abstracta. Se face distinctia dintre instantele si clasele operatiei. Instantele operatiilor se aplica instantelor, pe când clasele operatiei se aplica numai claselor.



Operatiile ‘Close’ si ‘Open’ se aplica fiecarei instante Windows, pe când operatia ‘GetAliOpenWindows’ se aplica clasei Windows.

Modul

Modulul este un element al modelului care ofera un mecanism general pentru gruparea altor elemente ale modelului. Modulele suporta conceptul de ierarhie si reprezinta nivele de abstractizare. Modulele pot contine module imbricate. Toate elementele modelului si toate elementele de prezentare vor rezida într-un modul sau într-un sortator (use case sau clasa).

Cel mai înalt nivel al modulului este apelat si ca model. Regulile ierarhiei HOORA sunt bazate pe conceptul de modul. Un element al modelului, care este în proprietate modulului P, poate accesa un alt element al modelului, care este proprietate modulului Q, dând metricile ierarhiei (care trebuie sa fie în limite date) aceste metrici sunt calculate de la modulul P la modulul Q. Probleme de ierarhie pot fi solutionate prin reorganizarea structurii modulelor.

Functiunea modulului

Functiunea modulului este un element al modelului care reprezinta o relatie de dependenta generica. Modulele au relatii de dependenta cu alte module. În cele mai multe cazuri, aceasta este utilizare functionala, care scoate în evidenta ca elementul de model apartine modulului Q. Exista, în acest

sens, doua tipuri de relatii de dependenta ale modulului: functiuni ale modulului specificate de catre utilizator si functiunile rezumate ale modulului.

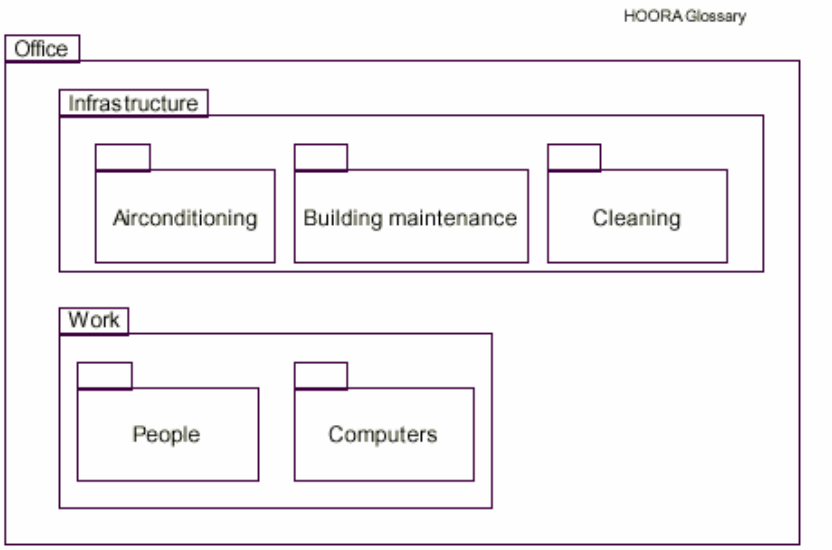


Figura 4.36. Modulul

Functiunile modulului specificate de catre utilizator sunt elemente de model în sine, care relationeaza un modul cu un altul, fara a specifica explicit care dintre modelele informatiei static si/sau dinamic este responsabil de respectiva dependenta.

Functiunile rezumate ale modulului sunt derivate din urmatoarele modele ale informatiei:

- mesajele rolului sortator al clasei A a modulului pA pentru a clasifica rolul clasei B a modulului pB (se creeaza astfel dependenta rezumata care prezinta faptul ca modulul pA depinde de modulul pB);
- relatiile de asociere, agregare, compozitie dintre clasa A a modulului pA si clasa B a modulului pB (care creeaza doua dependente rezumate ale modulului si care prezinta faptul ca modulul pA depinde de modulul pB, iar modulul pB depinde de modulul pA);
- relatia de mostenire care prezinta clasa A a modulului pA ca mostenire a clasei B a modulului pB (se creeaza o dependenta rezumata a modulului care prezinta faptul ca modulul pA depinde de modulul pB);
- dependenta modulului specificata de utilizator prezinta modulul pA ca depinzând de modulul pB (se creeaza o dependenta rezumat a modulului care prezinta ca modulul pA depinde de modulul pB).

Diagrama de modul

Este un element de prezentare care prezinta modulele si dependentele modulului specificate de utilizator.

Modelul modulului

Multimea tuturor elementelor de model care prezinta diferite dependente între modulele din model. Aceste informatii sunt prezentate în diagrama modulului, în diagrama modelului static si în diagrama modelului dinamic. Modelul modulului consta din toate dependentele modulului, din relatiile statice si mesajele dintr-un model dat. Diagramele modelului modulului vizualizeaza aceste diagrame ca o multime de diagrame consistente.

Diagrama rezumat a modulului

Este un element de prezentare care prezinta toate dependentele rezumate ale modulului unui modul dat cu alte module si care deriva din informatia modelului static, informatia modelului dinamic si informatia modelului modulului.

Parametru

Este un element de model care specific a variabila care trebuie schimbata într-o operatie.

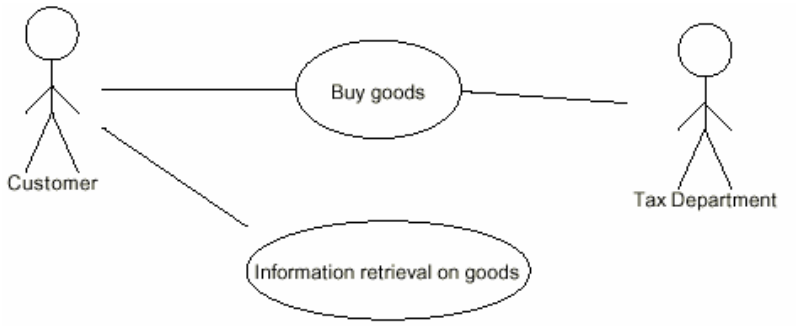
Elemente de prezentare

Elementele de prezentare prezinta informatia dintr-un model într-o maniera grafica sau textuala. Elementele de prezentare nu adauga continut modelului. Cele mai multe instrumente externe care 'citesc' un fisier model, vor ignora elementele de prezentare. Elementele de prezentare nu sunt elemente ale modelului. Acestea vor fi în proprietatea unui modul.

Actor primar

Este definit ca actorul care are scopul primar într-un use case.

Figura 4.37.
Actorul primar



Cerinte de timp real
Cerinta de timp real este un nume generic pentru diferite cerinte legate de timpul real. Se disting restrictii de durata, restrictii relative la rata utilizarii si restrictii de penetrabilitate.
Variabile de timp real
Acestea sunt variabile specificate de HOORA pentru a permite formularea cerintelor de timp real într-o maniera standardizata. Variabilele de timp real sunt asociate cu un scenariu. Variabilele de timp real sunt utilizate pentru a exprima cerintele de timp real.
Cerinta
Glosarul standard IEEE pentru terminologia ingineriei software (1997) defineste o cerinta ca:

- O conditie sau o capabilitate necesara unui utilizator pentru rezolvarea unei probleme sau pentru îndeplinirea unui obiectiv;
- O conditie sau capabilitate care trebuie îndeplinita sau posedata de catre un sistem sau o componenta a sistemului pentru a satisface un contract, un standard, o specificatie sau alte documente formale impuse;
- O reprezentare documentata a unei conditii sau capabilitati definite în punctele anterioare.

Cerintele sunt elemente de model cum ar fi clasele, use case etc. Procesul HOORA este gândit pentru a capta cerinte. Cerintele sunt structurate în ierarhii si sunt notificate într-un document al cerintelor. Acceptiunea uzuala a privirii cerintelor ca elemente ale modelului este foarte utila la crearea trasabilitatii relatiilor dintre cerintele (sursa) si elementele de model ale acestora (destinatie).
Responsabilitate
Responsabilitatea este definita ca si un contract sau o obligatie a unei clase. Orientarea obiect este deseori explicata ca fiind: 'identificarea unei multimi convenabile de clase care sa modeleze comportamentul cerintelor sistemului'. Aceasta se poate reexprima: 'identificarea responsabilitatilor cerute pentru a capta sistemul'. Acest mod de abordare statueaza explicit faptul ca operatiile nu sunt grupate arbitrar în clase. De fapt, se asigura ca operatiile sunt alocate claselor în concordanta cu regulile de consistenta, pentru a asigura faptul ca operatii care lucreaza împreuna sunt alocate aceleiasi clase. Responsabilitatile pot fi identificate explicit sau pot ramâne implicite.
Analiza de robustete
Aceasta tehnica asigura alocarea consistenta a responsabilitatilor la clase. Aceasta tehnica distinge: actori, clase de frontiera, clase de control si clase entitate.
Scenariu
Este o secventa de evenimente care ilustreaza un comportament. Un scenariu ilustreaza un comportament tipic al unei clase sau a unui use case. Se face distinctia frecventa între scenariile de baza si variante ale acestora. Un scenariu este reprezentat de diagrama de secvente sau o diagrama de colaborare.
SDL
Limbajul de specificare formala (Specification and Description Language) este un standard international elaborat de ITU. SDL sustine descrierea sistemelor de comunicatie distribuite complexe. Atât structura, cât si comportamentul pot fi descrise utilizând SDL, pe diferite nivele de abstractizare. SDL are definire semantici formale. HOORA descrie legaturile posibile între analiza HOORA si proiectarea SDL.
Actor secundar
Actorul secundar este actorul care participa într-un use case, dar nu are ca scop principal declansarea respectivului use case.

Figura 3.38 Actorul secundar

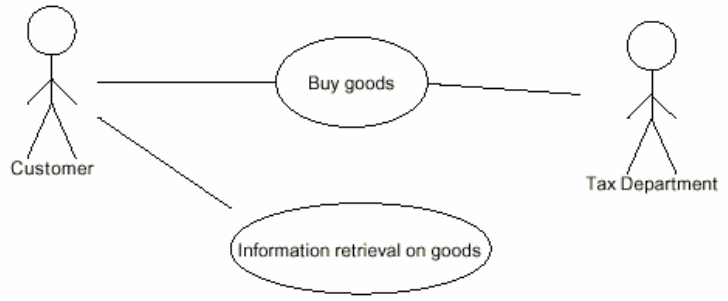
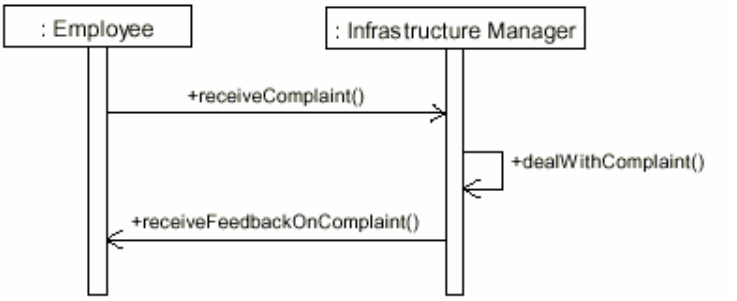


Diagrama de secventa

Este un element de prezentare care reprezinta un scenariu, i.e. rolurile sortatorului implicat si secventa de timp a mesajelor schimbate între acestea.

Figura 4.39. Diagrama de secvente



Diagramele de secventa si diagramele de colaborare reprezinta informatii similare. Dimens iunea timp este explicita pe diagramele de secventa. Într-o diagrama de colaborare, aceste informatii sunt disponibile prin mesajele numerotate. Rolurile asocierii sunt explicit prezentate pe o diagrama de colaborare, informatii care nu sunt disponibile pe o diagrama de secventa. O diagrama de secventa este legata de sortatorul al carui scenariu este considerat, i.e. cu o clasa sau cu un use case.

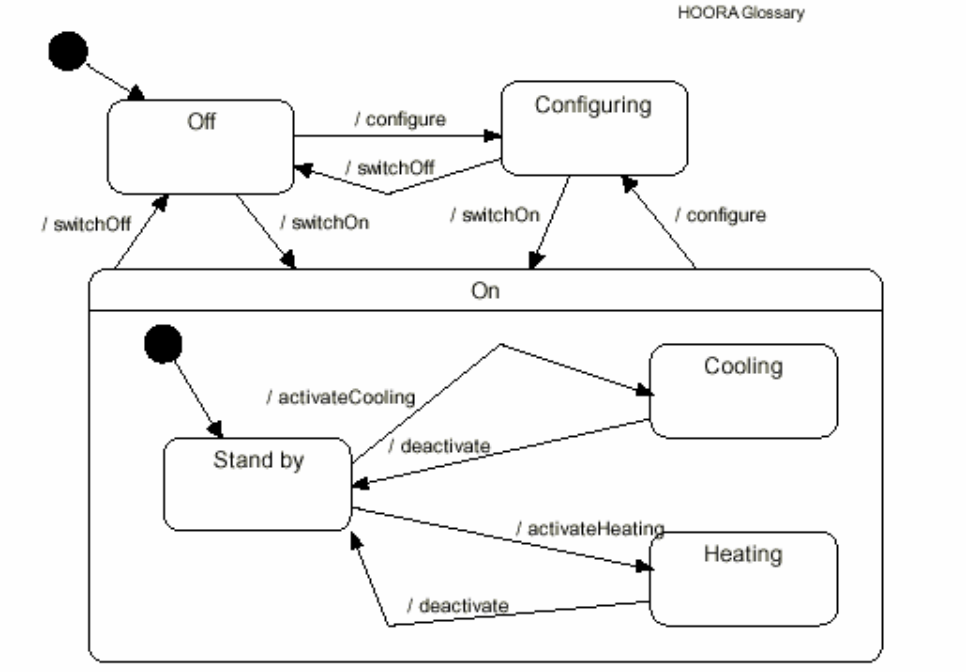
Starea

Starea este un element al modelului care reprezinta o faza din ciclul de viata al unui obiect, pe parcursul careia obiectul satisface o anumita cerinta, realizeaza anumite activitati, sau asteapta un eveniment. Actiunile de intrare si actiunile de iesire pot fi optional legate de o stare. O stare este proprie unei clase, ale carei instante le descrie. Starile sunt legate între ele prin tranzitiile de stare.

Diagrama starilor

Aceasta diagrama este un element de prezentare, care reprezinta un model al starilor, legate de o clasa.

Figura 4.40.
Diagrama starilor



Tranzitia starii
Este tranzitia de la o stare la alta datorate unui eveniment apelant. În HOORA, tranzitia starilor este totdeauna legata cu operatiile clasei legate de modelul de stare la care apartine respectiva tranzitie de stare.
Modelul static
Este multimea tuturor elementelor care capteaza diferitele relatii statice dintre clasele modelului. Aceste informatii sunt prezentate pe diagramele claselor si pe diagramele modelului static. Modelul static este gândit pentru a creste gradul de întelegere (perceptibilitate) a contextului structural al fiecarei clase. Modelul static prezinta care dintre celelalte clase sunt relationate cu o stare data. Un model static trebuie, de asemenea, sa fie comparat cu modelul dinamic pentru a identifica relatiile statice care nu au corespondent în dependentele dinamice. Relatiile statice (structurale) din modelul static valideaza dinamicile sistemului. Fara acestea, instantele A si B nu se vor 'cunoaste' (i.e. nu au o relatie statica) si, în consecinta, nu vor putea comunica (i.e. nu au dependente dinamice). Modelul static consta din clase si relatiile acestora (asociere, agregare, compozitie, generalizare).
Diagrama modelului static
Diagrama modelului static pentru o clasa este un element de prezentare care enunta o clasa si toate relatiile statice dintre respectiva clasa si celelalte clase. O diagrama a modelului static vizualizeaza modelul static si permite compararea acestuia cu modelul dinamic. Spre deosebire de diagrama modelului dinamic si diagrama rezumat a modulului, diagrama modelului static nu contine elemente noi de model. Conexiunile dintre clase sunt relatii statice definite în model. În diagrama modelului dinamic si în diagrama rezumat a modulului conexiunile sunt elemente noi ale modelului care sunt generate, pe baza informatiilor din model.

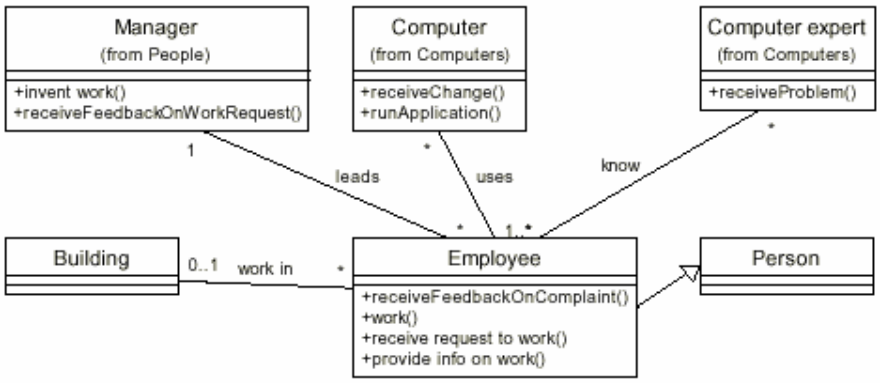


Figura 4.41
Diagrama modelului

Relatia statica
Relatia statica refera o relatie de asociere, agregare, compozitie si mostenire. Modelul static consta din toate clasele si toate relatiile lor statice. Relatiile statice valideaza dinamicile sistemului. În lipsa acestora, instantele A si B nu se vor 'cunoaste' unele pe altele si, în consecinta, acestea nu vor fii capabile sa comunice între ele. Mostenirea este un tip special de relatie dinamica între instante. Aceasta este numai o relatie a clasei.
Stereotipul
Stereotipul este un element al modelului care extinde un tip al unui element al modelului. Un stereotip introduce o extensie a unui element al modelului. (o specialClass ca o extensie a unei clase). O astfel de clasa poate avea semantici suplimentare si iconuri suplimentare. Utilizarea stereotipurilor nu trebuie exagerata, deoarece aceasta limiteaza interschimbabilitatea cu alti membrii ai comunitatii UML. Exista, de altfel, anumite stereotipuri standard definite în standardul UML. Stereotipurile se refera numai la elementele modelului si nu la elementele de prezentare. Toate extensiile HOORA, UML definite pot fi implementate prin utilizarea stereotipurilor si/sau numelor conventiilor. Numele conventiilor sunt cerute pentru prezentarea elementelor (i.e. diagrama modelului dinamic, diagrama modelului static, diagrama rezumat a modulului). Stereotipurile sunt utilizate pentru extensii specifice: dependentele dinamice (stereotipul asocierii), dependenta rezumat a modulului (stereotipul dependentei modulului) modulul utilitate (stereotipul modulului), cerinte (stereotipul comentariilor), cerintele în timp real (stereotip al comentariilor). Stereotipurile pot definite pentru o necesitate specifica proiectului; de exemplu: indicarea clasei de securitate. Semnificatia precisa a unui astfel de stereotip depaseste specificitatea proiectului.
Subclasa
O subclasa este o clasa specializata dintr-o relatie de generalizare.

Sinonime: clasa derivata, clasa filiatie
Antonim: superclasa
Superclasa
Superclasa este o clasa generalizata dintr-o relatie de generalizare.
Sinonim: clasa de baza, clasa parinte
Antonim: subclasa
Restrictia puterii de trecere
Aceasta restrictie este un tip al cerintei de timp real care indica ca puterea de trecere (care corespunde atât numarului de intrari procesate pe unitatea de timp, cât si numarului de date emis pe unitatea de timp) trebuie sa se încadreze în limite temporale date.
Trasabilitate
Trasabilitatea este tehnica de trasare a elementelor modelului catre elemente ale modelului care le cauzeaza. Trasabilitatea cerintelor permit trasarea elementelor originale ale modelului i.e. cerintele cu orice tip de elemente de model care le capteaza. Trasabilitatea cerintelor tine cont de structura ierarhica a modulului: cerintele pot fi legate de module. Elementele modelului pot numai sa fie legate de cerintele deja legate de modulele lor parinte. Elementele modelului caracteristice clasei (operatii, atribute, stari) pot fi legate numai de cerintele deja legate de clasa lor parinte. Orice cerinta poate fi legata de orice numar de elemente ale modelului. HOORA sustine relatii de trasabilitate din cerinte pentru:

- module
- clase
- relatii de asociere, agregare, compozitie, generalizare dependenta
- operatii
- stari
- atribute

UML
UML este un limbaj standard OMG pentru specificarea, vizualizarea, construirea si documentarea produselor sistemelor software ca si pentru modelarea si alte sisteme ne-software. UML reprezinta o colectie de cele mai bune practici ingineresti care si-au dovedit succesul în modelarea sistemelor mari si complexe. UML este în esenta un set de conventii pentru sintaxa si semantici care specifica elemente ale modelului si elemente de prezentare. UML este numai o notatie. Procesele pot specifica reguli suplimentare secvente de pasi, preferentiale care trebuie urmati. HOORA este un exemplu pentru aceasta.
Use case
Use case este un element al modelului care specifica o secventa de actiuni, incluzând variante si extensii, pe care sistemul le poate realiza si care determina un rezultat observabil al unei valori la un actor principal particular. Un use case descrie scenariul de baza tipic declansat de actorul primar, precum si câteva variante si extensii ale acestui scenariu. Un use case îsi propune sa fie o piesa a functionalitatii coerente. Un use case, grupeaza, de asemenea, o multime de scenarii reprezentate de diagrame de secventa sau de diagrame de colaborare. O diagrama de secventa sau o diagrama de colaborare, apartine de asemenea unei clase sau unei use case.
Diagrama use case
Diagrama use case este un element de prezentare care reprezinta actori, use case si asocieri de comunicare între use case si actori. Diagrama use case este utilizata, tipic, pentru informatia modelului use case si pentru a adauga structura unui set mare de scenarii (reprezentat de diagrame de secventa sau de colaborare).

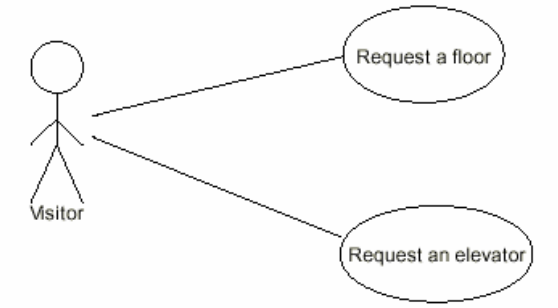


Figura 4.42
Diagrama use case

Modelul use case
Este multimea tuturor elementelor modelului care capteaza comportamentul functional al sistemului. Modelul use case consta din use case, actori, relatii de asociere a comunicatiei dintre use case si actori, scenarii reprezentate de diagrame de secventa sau de colaborare.
Restrictia ratei de utilizate
Restrictia ratei de utilizare este o cerinta de timp real care indica faptul ca rata de utilizate a resurselor sistemului trebuie sa se gaseasca în anumite limite (de obicei specificate sub forma de procente).
Modulul utilitatii
Este un modul care necesita sa fie accesibil de alte module, indiferent de locul pe care îl ocupa în ierarhie. 'Utility Package' din HOORA este o extensie a UML si poate fi considerat ca un modul stereotip.

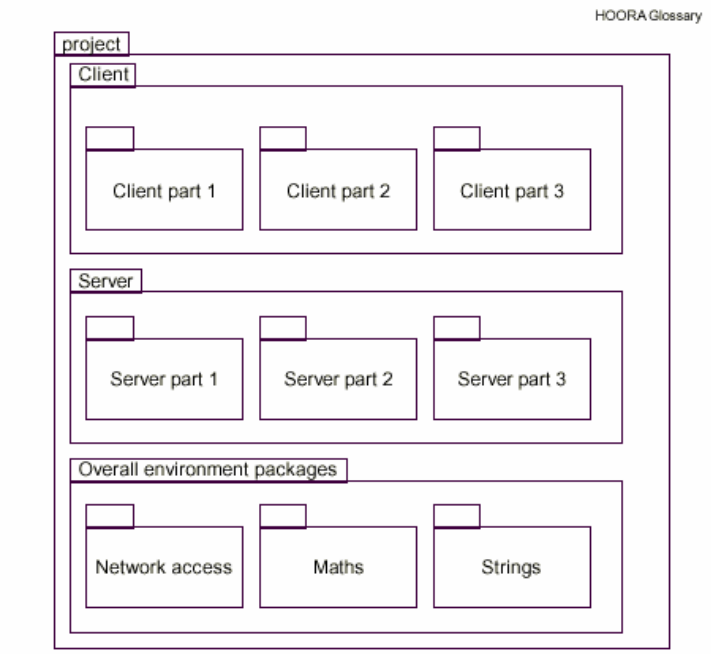
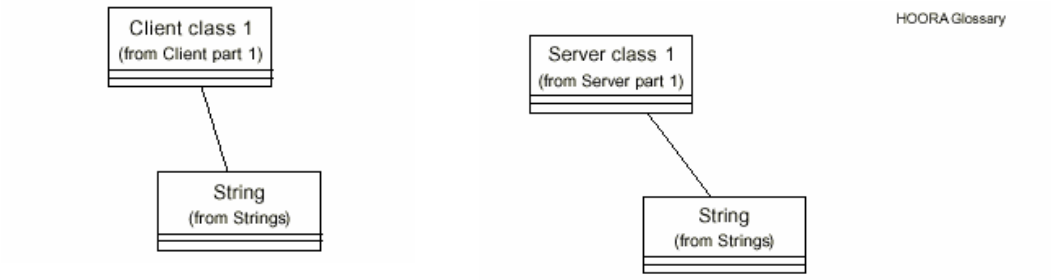


Figura 4.43. Modulul utilitatii

În exemplul prezentat mai sus a fost creat modulul 'Overall environment package'. Acest modul contine alte module. Aceste module trebuie sa fie accesibile si toate celelalte module, fiind astfel module utilitate. Ca rezultat, se poate accesa clasa String (care este proprie modulului utilitate String) de oriunde din afara modelului, independent de regulile de ierarhie si de valorile alese pentru maximum de nivele permise superior si inferior.



Vizibilitatea
Toate elementele modelului dintr-un spatiu de nume au o specificare a vizibilitatii. Aceasta denota permisiunea de a referi elemente ale altor spatii de nume. În general, numele spatiului este un modul sau o clasa. (operatiile si atributele pot fi specificate, de asemenea, pentru use case, dar aceasta se produce destul de rar). În cazul modulului, vizibilitatea va denota elementele modelului modulului (clase, use case, actori, module de nivel scazut). În cazul clasei, vizibilitatea va denota elementele modelului clasei (atribute, operatii, clase de nivel inferior) Elementele de prezentare nu au specificarea vizibilitatii.

Exista trei posibilitati pentru vizibilitate: publica, protejata, privata. Vizibilitatea publica implica faptul ca accesul este permis (regulile furnizate de ierarhie sunt înca valide). Vizibilitatea privata implica faptul ca nu este permis nici un acces din exterior. Accesul este restrictionat la spatiul de nume din interior (modul sau clasa). Vizibilitatea protejata pentru elementele modelului din interiorul unei clase implica faptul ca o subclasa poate 'vedea' atributele si operatiile superclasei (protejate). Vizibilitatea protejata pentru elementele modelului dintr-un modul este fara sens.

XMI

XML Metadata Interchange, este un standard OMG al carui scop principal este de a valida schimbul de metadate între instrumentele de modelare (este bazat pe UML al OMG) si al depozitelor de metadate (bazat pe OMG MOF) în sisteme eterogene distribuite. XMI integreaza trei standarde cheie industriale:

- XML – eXtensible Markup Language (standard W3C)
- UML – Unified Modeling Language (standard de modelare OMG)
- MOF – Meta Object Facility un standard OMG pentru metamodelare si depozitarea metadatelor

Standardul XMI consta din:

- O multime de DTD (Document Type Definition) a XML pentru producerea regulilor de transformare a metamodelelor bazate pe MOF în DTD uri XML.
- O multime de documente XML pentru producerea de reguli pentru codarea si decodarea metadatelor bazate pe MOF.

XML

Este un limbaj bazat pe Web, pentru schimbul electronic de date. XML este un nou format desemnat sa lege informatia structurata de un Web. XML este o tehnologie standard deschisa a lui W3C, grupul de standarde responsabile cu întretinerea si perfectionarea HTML si a altor standarde legate de Web. XML este o submultime a SGML care întretine aspectele arhitecturale importante ale separarii contextuale la stergerea caracteristicilor neesentiale. Formatul documentului XML încapsuleaza continutul din interiorul tintelor care exprima structura. XML furnizeaza de asemenea abilitatea de a exprima reguli pentru structura unui document. Aceste doua caracteristici permit separarea automata a datelor si a metadatelor, precum si validarea de catre instrumente generice a documentelor XML relativ la gramatica acestora. Spre deosebire de HTML, un document XML nu include prezentarea informatiei. În schimb, un document XML poate fi utilizat pentru o prezentare vizuala prin aplicarea informatiei prin tehnologii ca XSL (eXtensible Style Language). Seturile de Web si browserele cresc semnificativ în viteza prin adaugarea XML si XSL functionalitatii lor.

4.3. LIMBAJUL DE DESCRIERE SI SPECIFICARE SDL

SDL este un limbaj formal orientat obiect definit de The International Telecomunications Union - Telecomunications Standardization Sector (ITU-T) (Formal Comite Consultatif International Telegraphique et Telephonique (CCITT)) ca recomandarea Z.100. Limbajul este realizat în intentia de a specifica aplicatii complexe, interactive, în timp real, gestionate prin evenimente, care implica mai multe activitati concurente care vor comunica prin intermediul semnalelor discrete.

Cresterea pietei internationale de echipamente si aplicatii sporeste în mod consistent, prin natura eterogenitatii produselor hardware sau software, necesitatea disponibilitatii acestora de a comunica unele cu altele. În consecinta, metodele formale pot fi standardizate la nivel international. SDL este unul dintre limbajele care îsi propune specificarea de aplicatii complexe, în timp real. Puterea limbajului rezida în posibilitatile oferite în a descrie structuri, comportamente si date ale sistemului. Este acceptat, în mod curent ca elementul cheie în succesul dezvoltarii unui sistem este de a produce, în mod riguros specificatiile de sistem si proiectarea acestuia. Aceasta sarcina necesita un limbaj de specificare convenabil care sa satisfaca urmatoarele cerinte:

- Un set de concepte bine definite.
- Specificatii clare, precise, concise si neambigue.
- O baza minutioasa si exacta pentru analiza specificatiilor.
- O baza pentru determinarea consistentei specificatiilor.
- Suport procesabil pentru generarea aplicatiilor, fara a necesita faza traditionala de codificare.

SDL a fost definit pentru a raspunde acestor necesitati. Este un limbaj de specificare grafic care este în acelasi timp si formal si orientat obiect. Limbajul este capabil sa descrie structuri, comportamente si date pentru sisteme de comunicatie în timp real, distribuite având rigoarea matematica care elimina ambiguitatile si garanteaza integritatea sistemului. Dispune de o sintaxa grafica ceea ce îi confera o privire extrem de intuitiva.

Astfel de limbaje includ alte notatii de nivel înalt ca: OMT (Object Modelling Technique) / UML (Unified Modelling Language), modele obiect si studii de caz pentru centrul mobil de comutare (MSC (Message Sequence Chart)) ca si notatii abstracte (ASN.1) sau CORBA (Common Object Request Broker Arhitecture) / limbajul descrierii interfetei pentru definirea tipului de date (IDL). Mai mult, exista deja instrumente disponibile care pot genera cod executabil, spre exemplu C/C++ sau limbajul de nivel înalt ITU (CHILL), direct din proiectul SDL. Teste pot fi de asemenea generate direct din Specificatiile SDL prin formarea de serii de test în TTCN.

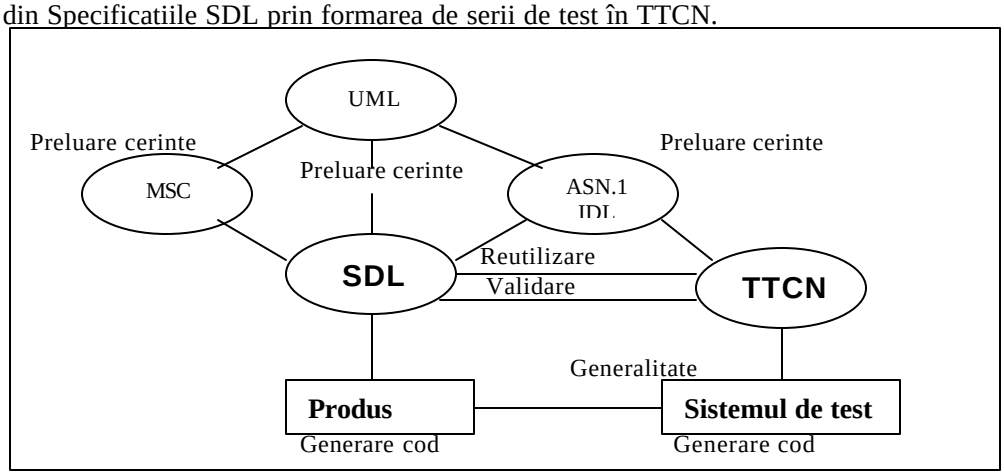


Figura 4.44. Relatia dintre SDL si alte limbaje de specificare

În mod tipic procedura pentru analiza cerintelor în vederea producerii unei implementari si a testarii acesteia va trebui sa implice urmatorii pasi:

- Colectarea cerintelor initiale într-un document text;
- Studierea rezultatelor într-un numar de modele obiect OMT/UML si studii de caz MSC cu descrierea scenariilor tipice. Clasele rezultate sunt implementate în SDL ca diagrame SDL si definitii de tip de date SDL/ASN.1/IDL.
- Completarea diagramelor SDL si a specificatiilor ASN.1 sau IDL la nivelul la care acestea pot fi simulate si li se poate verifica consistenta tinând cont de analiza cerintelor sistemului.
- Utilizarea verificarilor si validarilor pentru a determina daca proprietatile cerute sunt implementate corect si complet.
- Realizarea seriei de test TTCN. Testele pot fi generate din specificatiile SDL. În anumite cazuri aceste teste pot fi deja disponibile sau o parte bune din testele disponibile pot fi înca reutilizate.
- Generarea de cod pentru a crea o serie de test executabila care poate rula sub un sistem de test.
- Rularea testelor executabile si testarea aplicatiei într-un mediu convenit.

Modelul teoretic de baza al unui sistem SDL consta într-un set de masini finite de stare (FSM) care ruleaza în paralel. Aceste masini sunt independente între ele si comunica prin semnale discrete Un sistem SDL va avea urmatoarele componente:

- Structura - sistem, bloc, procese si proceduri ierarhizate.
- Comunicatie - semnale, cu parametrii optionali si canale (sau semnale de cale).
- Comportament - procese.
- Date - tipuri abstracte de date (ADT).
- Succesiuni - descrierea relatiilor si a specializarilor.

Divizarea sistemului în blocuri proces si procese ierarhizate se numeste partitionarea sistemului. Obiectivele partitionarii sunt:

- Încapsularea informatiilor irelevante si plasare acestora pe nivelul inferior.
- Urmarirea unor subdiviziuni natural functionale.
- Crearea de module de dimensiune controlabila.
- Crearea de corespondente cu structurile actuale hardware si software.
- Reutilizarea specificatiilor deja existente.

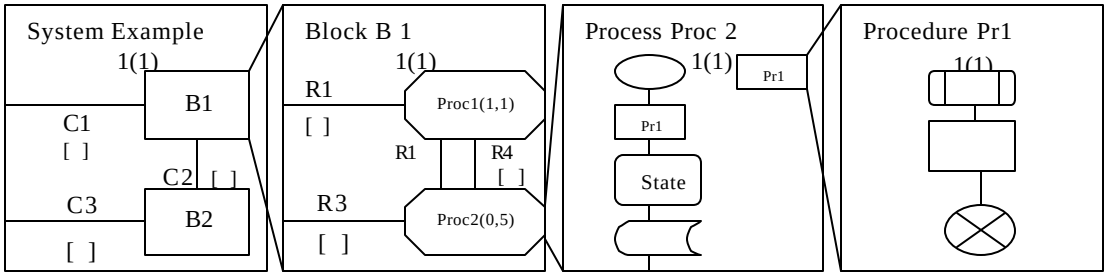


Figura 4.45 – Structura arhitecturala generala a SDL

Fiecare tip de proces SDL este definit ca un set imbricat de masini de stare ierarhizate. Fiecare masina de stare este implementata intr-o procedura. Procedurile pot fi recursive; acestea pot fi locale pentru un proces, sau pot fi globale dependente de scopul pe care îl îndeplinesc. SDL suporta de asemenea paradigma procedurilor la distanta ceea ce permite apelul unei proceduri care se executa în contextul unui alt proces.

Proceselor SDL li se aloca spatii de memorie distincte (adica: date sunt locale unui proces sau unei proceduri). Acesta este un aspect foarte important deoarece reduce substantial numarul deficientelor si creste robustetea. Un set de procese pot fi grupate logic într-un bloc (care este definit în acest sens ca subsistem). Blocurile pot imbricate unele în altele pentru a transforma, în mod recursiv sistemul într-o succesiune de subsisteme mai mici, încapsulate si controlabile. Acest mecanism de transformare este important pentru eforturile de dezvoltare pe termen lung, SDL simplificând aceasta si furnizând de asemenea interfete clare pentru subsisteme.

Structura statica a unui sistem este definita în termeni de blocuri si canale. Un bloc este perceput ca un modul cu un model black box bine cunoscut. *Structura dinamica* este definita cu ajutorul proceselor si a conceptelor de semnal de cale. Un proces este un instrument independent care reactioneaza la stimuli în termeni de semnale .

Comportamentul dinamic în sistemul SDL este descris în termeni de procese. Sistemul/ierarhia blocurilor este o descriere numai statica a structurii sistemului. Procesele în SDL pot fi create la startarea sistemului sau la momentul executiei acestuia. Pot exista mai multe instantieri ale unui proces. Fiecare instantiere este identificata în mod unic de catre identificatorul de proces (PId). Acest mecanism face posibila adresarea unui semnal la o instantiere a procesului. Implementarea conceptelor de proces si instantiere a procesului, care lucreaza autonom si concurential fac din SDL un limbaj pentru timp real în adevaratul sens al cuvântului.

SDL accepta doua moduri de descriere a datelor, tipul abstract de date (ADT (Abstract Data Type)) si ASN.1. Integrarea structurilor ASN.1 permite partajarea datelor între limbaje, ca si reutilizarea structurilor de date existente.

Conceptul ADT utilizat în SDL este convenabil pentru un limbaj de specificare. O ADT este un tip de date fara specificare de structura. În schimb ADT specifica o multime de valori, de operatii si ecuatiile pe care operatiile trebuie sa le îndeplineasca. Aceasta abordare face, deosebit de simpla, reprezentarea tipurilor de date SDL în alte limbaje de nivel înalt utilizate. Multimea tipurilor predefinite creeaza posibilitatea de a actiona, într-un mod traditional, cu datele în SDL. ADT în SDL poate fi utilizat mai mult decât reprezentarea datelor:

- Acoperirea manipularilor de date
- Acoperirea partii algoritmice a unei specificatii
- Crearea de interfete cu rutinele externe

Conceptele OO (orientate obiect) pe care le implementeaza SDL dau utilizatorului instrumente puternice pentru a realiza structurarea si reutilizarea. Conceptele sunt bazate pe declaratia de tipuri. Declaratia de tipuri se poate plasa în oriunde, chiar si în interiorul sistemului închis fata de context, sau la nivel sistem. Declaratiile de tip pot fi de asemenea plasate în pachete externe sistemului pentru a fi partajate cu alte sisteme. În SDL specializarea tipurilor poate fi împlinita în doua moduri:

- Un subtip poate adauga proprietati nedefinite în supertip.
- Un subtip poate redefini tipurile virtuale si tranzitiile virtuale definite într-un supertip.

Un sistem este modelat în SDL ca un numar de instante proces, concurente, care comunica între ele, schimbând instante semnale între ele si în mediu sistemului. Fiecare instanta proces este o masina finita de stare extinsa. Semnalele transporta identitatea instantei proces emitator si poate transporta

valori de date de la emitator la receptor. În cazul adresarii explicite a unui semnal, se va transporta de asemenea si identitatea instantei procesului receptor. Un tip proces defineste un tipar care poate fi utilizat pentru a defini multimea instantelor procesului. Fiecare multime a instantelor procesului (obiect) este definita într-un context precis al sistemului. Instantele individuale ale procesului pot fi create ca fiind membre ale multimii instantelor procesului. O definitia de tip de proces consta din:*Definitii de porti, Clauze de specializare, Definitii de parametrii formali, Definitii anexate (înglobate) (încapsulate),Corpul procesului*

Portile sunt interfetele tipului de proces. O poarta este caracterizata prin semnale permise într-o directie. Portile pot avea limitari de punct final, indicate într-o boxa plasata la finele liniilor conectate la o poarta.

Parametrii actuali pot sa fie trecuti în proces la crearea acestuia. Parametrii actuali se verifica relativ la concordanta cu parametrii variabilei formale, înainte de startarea interpretarii instantei procesului. Definitia unui proces poate sa contina definitii atasate. Acestea pot fi definitii de servicii, proceduri, semnale, timere, tipuri de date si variabile. Variabilele pot fi definite numai în cadrul procesului, serviciului sau procedurii si pot fi accesibile numai în cadrul aceleiasi instante a procesului, serviciului sau procedurii.

Comportamentul unui proces este definit prin corpul procesului. Corpul procesului consta din o tranzitie de start, un numar de stari cu declansatorul de tranzitii si tranzitii de stare asociate. Începutul corpului procesului este indicat printr-un simbol de start Într-o stare, instanta unui proces va astepta o declansare pentru a realiza o tranzitie din respectiva stare. În momentul în care instanta unui proces paraseste o stare, aceasta interpreteaza tranzitia de stare. Dupa realizarea tranzitiei, instanta procesului intra într-o noua stare, care va fi referita ca starea urmatoare.

Instantele procesului sunt identificate utilizând tipuri de date predefinite, speciale, numite *PId.* PId include o valoare speciala *Null*, care nu denota nici o instanta proces. Fiecare instanta proces este identificata printr-un *PId* unic. Sunt disponibile patru *PId* predefinite, pentru fiecare instanta proces, uzuale pentru comunicarea cu alte procese:

- *self* – desemneaza instanta proces însasi
- *sender* – desemneaza instanta proces de la care instanta proces a primit, cel mai recent, un semnal
- *parent* – denota instanta proces care a creat o instanta proces
- *offspring* – denota instanta proces cea mai recent creata de instanta proces.

Dependentele temporale pot fi modelate prin *timere*. Un timer este un ceas care poate fi setat cu o un timp de expirare. La trecerea acestui interval de timp, expirarea timerului este semnalata instantei proces ca un *input* normal. În cazul în care timerul nu mai este necesar, poate fi resetat înainte de expirare, pentru a evita expirari neasteptate.

Abordarea axiomatica În abordarea axiomatica, semanticile sunt date prin echivalenta de expresii. Echivalenta expresiilor este enuntata, printr-un numar de ecuatii fiecare continând o parte stânga a expresiei, simbolul "==" si o parte dreapta a expresiei. Ecuatiile pot fi calificate. O ecuatie calificata introduce un numar de nume cu tip, care, în ecuatie desemneaza orice valoare a tipului respectiv, ceea ce înseamna ca partea stânga si partea dreapta a expresiei sunt egale daca pentru fiecare ocurenta a numelui se substitue cu aceiasi valoare. Alegerea valorii actuale nu produce diferente daca cele doua parti sunt echivalente pentru toate valorile.

Abordarea algoritmica În abordarea algoritmica, comportamentul operatorilor este specificat într-o diagrama operator utilizând tranzitia, ca la procese. Într-o diagrama operator nu este permisa manevrarea starilor globale ca variabile de proces sau intrari. Abordarea algoritmica este mai "orientata pe implementare" decât cea axiomatica si, de multe ori apare ca mai usor de utilizat, deoarece se aseamana cu specificarea proceselor. Pe de alta parte, este mai putin expresiva decât abordarea axiomatica.

Specificarea comportamentului într-un formalism alternativ de date. Aceasta abordare este utilizata la interfatarea SDL cu alt limbaj, cum ar fi notatia de specificare a datelor ASN.1, sau la translatarea SDL într-un limbaj de programare, folosind astfel toate avantajele caracteristicilor oferite de un limbaj dat. Notatia alternativa este cuprinsa între cuvintele cheie *alternative* si *endalternative*. Din punctul de vedere al SDL, constructia este privita ca un text informal, i.e. SDL nu furnizeaza relatii formale pentru alte formalisme de date. Un exemplu de interfatare cu limbajul C este data mai jos:

```
newtype Stringutility;
operators count: Charstring, Character -> Natural
alternative C
#include "stringutil.h"
```

```
#include "stringutil.c"
endalternative;
endnewtype Stringutility;
```

Structura sistemului Interpretarea unui sistem SDL este, de fapt, interpretarea unui numar de procese interpretate în paralel. În termeni de instante si tipuri, interpretarea unui sistem SDL este deci, interpretarea unei instante sistem.

O instanta sistem este un container pentru un numar de instante bloc. O *instanta bloc* este de asemenea, un container pentru un numar de instante proces sau instante (sub)bloc.O *instanta proces* este, fie o masina de stare, fie un numar de instante serviciu alternante, interpretate secvential din interiorul unei instante proces. O **instanta serviciu** este o masina de stare.

O specificatie SDL consta dintr-un numar de diagrame care, împreuna specifica comportamentul instantei sistem. Diagrama sistem defineste instanta sistem însasi, pe când toate celelalte definesc tipuri si instante necesare pentru a defini instanta sistem.

Tipul asociat cu o instanta sistem este dat la definirea instantei. Tipurile: sistem, bloc, proces, serviciu pot fi definite prin diagrame specifice fiecaruia.

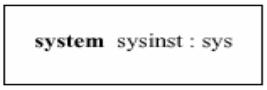


Figura 4.46. Diagrama si instanta a tipului sistem

Rutele semnal se aseamana cu canalele, fiind diferentiate prin:

? Rutele de semnal sunt utilizate pentru a conecta instantele proces si instantele serviciu, pe când canalele sunt utilizate pentru a conecta instante blocuri.

? Canalele pot implica temporizari necontrolabile la transmiterea semnalelor. Aceasta caracteristica este uzuala la specificarea sistemelor distribuite, pentru care nu se poate considera o transmisie instantanee.

? Canalele pot avea o substructura de canal atasata pentru a elabora mediul comunicational pentru sisteme distribuite.

? Rutele de semnal sunt optionale, si atunci când sunt omise se deriveaza implicit ca fiind toate caile de comunicatie posibile.

Ca o alternativa la definirea instantelor proces, un tip bloc poate fi un container pentru un numar de instante bloc interconectate si conectate pe frontiera blocului cu canale. În acest caz, instantierea tipului bloc rezulta în crearea blocurilor continute. Daca un tip bloc contine un numar mare de procese, se va proceda la gruparea conceptuala a proceselor relationate. Conceptul de poarta este utilizat de asemenea, pentru specificarea punctelor de conexiune a tipurilor proces si a tipurilor servicii. Definirea comportamentului unui proces se poate face atât printr-o masina de stare, cât si în termeni de instante de servicii. Fiecare astfel de instanta serviciu este o masina de stare separata, dar la un moment dat, în graf se executa un singur serviciu. Când, la executarea serviciului se atinge o anumita stare, serviciul este capabil sa consume semnalul urmator din portul de intrare al instantei procesului, adica instantele serviciu partajeaza portul de intrare al instantei proces prin variabile si valori ale expresiilor *self*, *parent*, *offspring* si *sender*. O instanta serviciu este capabila sa consume un semnal, daca semnalul nu este salvat într-o stare data si daca semnalul poate fi receptionat de instanta serviciu respectiva. Doua instante serviciu diferite nu pot sa receptioneze acelasi semnal. Serviciile sunt utilizate pentru cazul în care comportamentul procesului poate fi descris ca un numar de activitati independente (numai partajarea de date) cu cele doua directii ale unui protocol, sau pentru cazul în care exista activitati comune pentru instante proces ale unor tipuri proces diferite.

Constructiile pentru definirea instantelor blocului si al instantelor procesului, de fapt, definesc multimi de instante, dar multimile constau numai dintr-o instanta, în afara de cazul în care la definirea instantei a fost utilizata constructia “number of instances”. Continutul multimii de instante proces se poate modifica în timp (o instanta proces este stearsa din multime la stoparea acestuia si este inserata la crearea actiunii care mentioneaza ca multimea instantelor este interpretata). Numarul de instante specificat la definirea multimilor instantelor este, în consecinta, doar numarul initial care va fi creat la crearea instantei sistem. În cazul în care nu se pot crea confuzii se vor utiliza termenii de instante bloc si instante proces, în locul termenilor de multime a instantelor bloc, respectiv proces.

Specializarea Un tip poate fi o specializare (o extensie) a unui alt tip. Conceptual tipurile dintr-o specificatie sunt grupate într-o ierarhie de tipuri (o arborescenta) în care nodul parinte al unui tip din ierarhie desemneaza supertipul din care tipul a fost specializat. Toate nodurile parinte ale unui tip sunt supertipuri ale tipului, iar acesta este subtipul lor. De multe ori, nivelele superioare ale ierarhiei sunt introduse numai pentru scopuri de clasificare a tipurilor, adica anumite tipuri pot fi 'abstracte' implicând faptul ca nu are sens ca acestea sa fie utilizate la crearea instantelor. În general, un tip specializat poate sa adauge orice proprietate care poate fi definita pentru tip.
Virtuali De multe ori, subtipurile trebuie sa faca modificari asupra supertipurilor, pentru a adapta supertipul la necesitatile cerute. În consecinta, la specializarea unui tip, o parte dintre tipurile definite local vor trebui redefinite. Supertipul defineste care sunt tipurile pe care un subtip le poate redefini. Aceste tipuri se vor numi tipuri virtuale. Pentru fiecare tip care defineste o masina de stare (tipuri proces, servicii sau proceduri) se pot redefini de catre subtip, tranzitiile pe stari, inclusiv tranzitia de start. Partile dintr-o masina de stare care pot fi redefinite de catre subtip (ca permisivitate data de supertip) sunt denumite: tranzitii virtuale. Pentru a pastra proprietatile unui tip care contine tipuri virtuale, apar limitari relativ la modul în care tipul virtual poate fi redefinit. Implicitul si limitarile minime constituie redefinirea a unui tip virtual, care redefinire trebuie sa fie o specializare a tipului virtual însasi.

O implementare semnificativa este realizata de Cinderella, ale carei furnituri acopera o gama larga din activitatile de evaluare a suportului de specificatii SDL.
Analizorul Cinderella al SDL Analizorul sintactic si static semantic este unul incremental si poate fi setat pentru a realiza analizele de baza la specificarea unui sistem SDL. Analizorul incremental analizeaza numai partile care s-au modificat sau care sunt afectate de modificare. Astfel, analiza sistemului este mult mai scurta ca durata de timp decât o reanaliza completa.
Simulatorul Simulatorul Cinderella este complet integrat în mediul de dezvoltare. Împreuna cu analizorul incremental, valideaza simularea specificatiilor partiale. Este, în anumite cazuri, posibila modificarea specificatiei pe durata rularii simulatorului. Simulatorul ofera de asemenea furnituri de baza, cum ar fi: un singur pas, utilizarea punctelor de oprire. Punctele de oprire pot fi setate sau distruse pe parcursul simularii. Trasele simularii pot fi vizualizate ca diagrame MSC.
Limitarile pentru SDL Deoarece o specificatie formala ar trebui sa fie, pe cât posibil, o implementare independenta, limbajele de specificare nu (si nu pot) acopera aspectele de performanta. Pentru a obtine masurile de performanta si/sau declaratii verificate asupra comportamentului în timp a sistemului în discutie, trebuie specificate proprietati cantitative. Altfel, va trebui o deplasare spre implementare pentru a obtine informatii despre proiect. Aspectele de performanta pot sa acopere rezultate cum ar fi caracteristicile de performanta a echipamentelor hardware, concurenta datorata resurselor partajate, algoritmii utilizati pentru manevrarea si programarea datelor, viteze de procesare, capacitatea canalelor de comunicatie, dimensiunea bufferelor, valorile de time-out, si fara a fi la sfârsit încarcarea resurselor si caracterizarea traficului. SDL pur nu accepta modelarea aspectelor cantitative. Daca cerintele functionale ale unei specificatii SDL poate fi examinata într-un stadiu de început al duratei proiectarii sistemului, investigarea proprietatilor cantitative poate fi realizata într-un stadiu mult mai târziu, în procesul de dezvoltare al proiectului, ceea ce duce la costuri excesive relative la corectarea erorilor de proiectare legate de performante. Mai mult, proiectantii se confrunta cu o falie metodologica între analizele functionale si cele cantitative ale sistemului, deoarece metodele pentru analizele cantitative cer transformarea specificatiilor SDL într-un model diferit. O astfel de transformare este costisitoare si expusa la erori. Deoarece, procesul de proiectare a sistemelor complexe este, în mod uzual iterativ, rezultatele analizei performantelor trebuie retransformate în modelul SDL, pentru a putea fi integrat în proiectarea sistemului ceea ce este, din nou, legat de costuri mari suplimentare si este un proces supus la o noua sursa de erori.

*O solutie posibila o reprezinta **QUEST**, ca o paradigma omogena pentru analizele functionale si ale performantei. Abordarea QUEST elimina problemele mentionate anterior, prin integrarea elementelor de proiectare si analiza cantitative si calitative*:

- Sistemul este supus analizei cantitative si calitative, înca din primele stadii. Fluxurile functionale si nefunctionale ale proiectului vor fi detectate cât de devreme posibil, minimizând astfel costul corectiilor;
- Nu sunt necesare transformari ale specificatiilor SDL într-un alt model.

Abordarea QUEST consta din doua componente:

1) Un numar de adnotatii care sunt încapsulate în comentarii formatate special. În acest sens SDL este extins la QSDL. Extensiile limbajului sunt vizibile numai instrumentelor care accepta QSDL. Celorlalte instrumente extensia este opaca.

2) Instrumentul QUEST care transforma o specificatie QSDL într-un model evaluabil automat (denumit evaluator) care furnizeaza metodele pentru automatizarea evaluarii comportamentului sistemului din urmatoarele puncte de vedere:

- Corectitudinea functionala a specificatiilor sistemului, atât prin explora-rea întregului spatiu al starilor, pentru detectarea blocajelor, depasirea cozilor, receptiile semnalelor nespecificate etc., cât si controlul pentru verificarea corectitudinii functionale.
- Verificarea performantelor si a verificarile legate de timp, i.e., validarea cresterii timpului si verificarea asertiunilor definite de utilizator prin modelul de verificare (model checking) în conjunctie cu cerintele de performanta.
- Analiza performantelor tranzitorii si vizualizarea masurilor performantelor.
- Analiza performantelor stationare.

A se nota ca specificatia QSDL care este derivata din SDL original nu adauga nici o functionalitate noua în sistem, dar este de asteptat sa rafineze specificatia originala adica, vor fi investigate aspectele nefunctionale ale specificatiei SDL. Relatia dintre SDL si QSDL este prezentata în figura 4.47. Daca specificatia SDL acopera numai aspectele functionale, specificatia QSDL descrie atât aspectele functionale cât si cele cantitative (i.e., non functionale) ale sistemului care urmeaza a fi investigat.

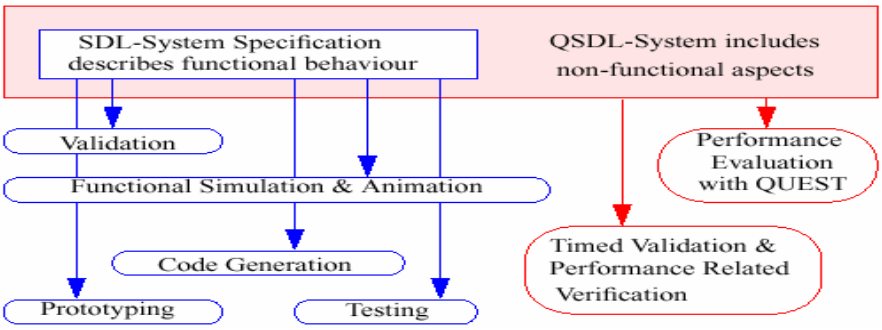


Figura 4.47 Abordarea QSDL
(preluata din manualul de utilizare QSDL)

Statie de asteptare Modelele de asteptare sunt o paradigma foarte raspândita pentru modelarea performantelor. Sunt utilizate pentru a descrie si analiza congestia cererilor multiple pentru resurse restrictive. Un sistem descris cu ajutorul statiei de asteptare poate fi investigat prin tehnici analitice, numerice sau simulative. În mod tipic, se utilizeaza notatia Kendall ?/B/c/K pentru a denota parametrii pentru queueing station.

Intersosirile si distributiile timpului de serviciu sunt date de A si respectiv B, c va denota numarul de servere si K este capacitatea, i.e., numarul de utilizatori pe care o statie de asteptare îi poate pastra. Suplimentar, se poate specifica o disciplina de coada cum ar fi primul sosit primul servit sau o structura de prioritati. În functie de tipul parametrilor este posibil sa fie derivate formule analitice închise pentru timpul de utilizare, de asteptare si de raspuns. Dintre cele mai cunoscute modele se remarca sistemele lui Erlang M/M/I-, M/M/I/K si M/M/m/m, care au fost utilizate cu succes în proiectarea retelelor de telefonie. Exista si alte tipuri de statie de asteptare cum ar fi partajarea procesorului, statii multi coada / multi serviciu si statii cu disciplina de programare. Un exemplu simplu îl reprezinta un singur server cu o disciplina FIFO:

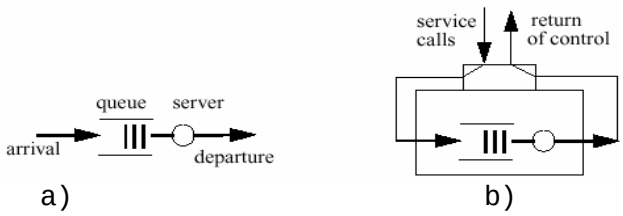


Figura 4.48. Server cu disciplina FIFO
(preluata din manualul de utilizare QSDL)

Abordarea lui QUEST, a statiei de asteptare, este o masina care furnizeaza un serviciu unui mediu. Serviciul furnizat poate fi cerut de entitatile din mediu, prin intermediul unei interfete. Acest punct de vedere este consistent cu proiectare moderna si cu metodologiile de specificare si a fost adoptat cu succes în domeniul modelarii performantelor. Timpul total în care o cerere sta în statie depinde de dimensiunea serviciului cerut, de viteza serverului si, suplimentar, de timpul de asteptare în coada. Dimensiunea serviciului cerut este descris, normal, prin variabile aleatoare, viteza serverului fiind totusi o constanta real pozitiva. Timpul de asteptare în coada este dependent de congestiunea datorata utilizarii concurente a masinii astfel definite. Dupa ce serviciul a fost complet controlul revine entitatii solicitante.

Se va considera urmatorul exemplu: o specificatie SDL care include un protocol pe doua nivele si un mediu formeaza nucleul modelului. Se ataseaza acestui nucleu specificatiile încarcarii (sursa traficului) si specificatiile resurselor (masinilor) pe care modelul le va atasa o întârziere datorata asteptarii, serviciului si transmisiunii. Semnalele emise de catre sursele de trafic realizeaza actiuni în instantele protocolului care, în final, leaga cererile de masini. Reprezentarea cererilor pe masini este prezentata în figura 4.49.

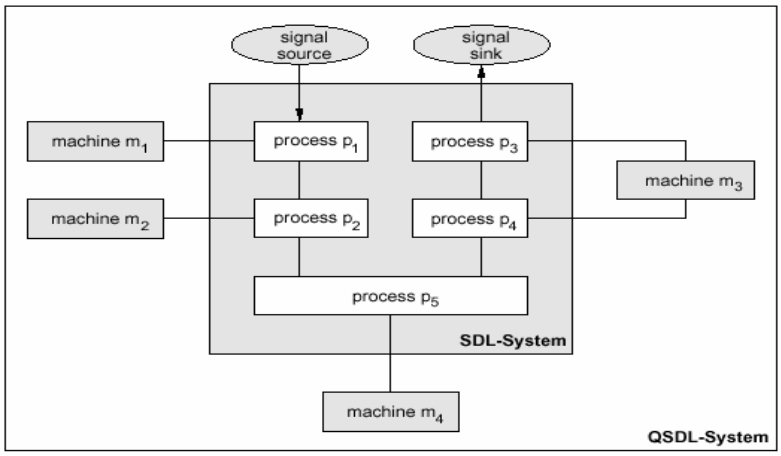


Figura 4.49 Reprezentarea cererilor pe masini
(preluata din manualul de utilizare QSDL)

În respectiva figura apar cele trei utilizari tipice ale masinilor:

- Modelarea paralelismului: instanta p1 si p2 au acces exclusiv la masinile m1 si respectiv m2;
- Modelarea utilizarii secventiale: instantele B3 si B4 partajeaza masina M3;
- Modelarea duratei de timp a transmisie fizice prin mediu, aici modelata prin masina m4

O topica importanta care afecteaza performanta sistemului este comportamentul surselor de trafic. Deci, caracterizarea încarcarii prin parametrii de trafic convenabili este o parte importanta pentru fiecare model cantitativ. QSDL accepta modelarea încarcarii prin diferite furnituri, în particular având disponibile un numar de functii de distributie aleatoare. Încarcarea este impusa unui sistem, adica caracteristicile traficului ale sistemului curent si viitor de telecomunicatii poate fi descris prin generatoare de încarcare ca procese QSDL. Semnalele generate de aceste procese finalmente realizeaza maparea dintre cereri si masini. Prin utilizarea surselor de trafic cu stari multiple definirea comportamentului sursei este foarte flexibil. Se pot descrie caracteristici de trafic foarte diferite ca de exemplu: transfer de fisiere, transmisii de semnale vocale, digitale, sau video. Un exemplu tipic pentru surse cu stari multiple este sursa on/off. În starea on sursa va genera trafic (pachete, celule) cu o anumita rata, iar în starea off nu. Tranzitia dintre stari poate fi descrisa de asemenea prin proceduri aleatoare.

Limbajul QSDL

Se vor introduce specificatiile limbajului QSDL (Queueing SDL) QSDL extinde specificatiile standard ale limbajului SDL printr-un numar de furnituri. De fapt, sunt doua astfel de tipuri de extensii:

- Prima extensie este generica si este încapsulata în comentarii formatate special. Acestea privind alte instrumente SDL standard pentru a fi apelate împreuna cu extensiile QSDL. În aceasta maniera o specificatie SDL poate fi prin furniturile QSDL ramânând înca utilizabila în instrumentele standard SDL.

? Cel de al doilea tip de extensie sunt tipurile de date predefinite declarate în SDL standard. QSDL admite aceste tipuri de date ca fiind predefinite. Pentru a utiliza aceasta extensie cu instrumentele standard SDL, utilizatorul va încarca un set de programe speciale SDL, care declara aceste tipuri de date.

Cea mai importanta extensie sunt resursele bazate pe principiul statiilor de asteptare. QSDL permite introducerea masinilor consumatoare de timp în specificatiile SDL pentru a modela congestionarea proceselor pentru resurse limitate. Dupa adaugarea modelelor de încarcare si dupa definirea unei reprezentari a încarcarii pe masini, specificatiile SDL rezultante contin, în general, toate informatiile necesare pentru generarea unui model evaluabil automat. Mai sunt introduse si alte furnituri pentru a permite SDL modelare si utilizarea diferitelor tehnici de analiza

Tratarea timpului în QSDL Specificatiile SDL nu contin informatii referitoare la timpul necesar pentru executia unei actiuni individuale a proceselor sale. Aspectele legate de timp sunt utilizate numai pentru a modela timeout-urile prin mecanismele de timere a SDL. În QSDL se asociaza durate de timp (deterministe sau probabiliste) si utilizarea resurselor cu anumite elemente ale limbajului QSDL. QSDL dispune de tranzitii temporizate si de stari temporizate.

Tranzitii temporizate O tranzitie SDL este o secventa de actiuni SDL. SDL furnizeaza mai multe tipuri de astfel de actiuni: *output, set, reset, task* si altele. În QSDL exista o actiune suplimentara: *reques*t. Aceasta actiune utilizeaza un anumit serviciu de pe o masina accesabila în sistemul QSDL. Fiecare tranzitie QSDSL poate contine un numar arbitrar de actiuni request. Fiecare request blocheaza procesul apelant pentru o durata definita de timp. Aceasta durata de timp este timpul de raspuns al request-ului. Timpul de raspuns va consta din timpul de serviciu si din timpul de asteptare pe masina. Suma tuturor timpilor de raspuns al request-urilor dintr-o tranzitie este timpul de declansare (firing) al tranzitiei. Timpul de declansare depinde de dimensiunea serviciului pentru diferitele actiuni request, de viteza masinii si de încarcarea masinii.

QSDL ofera mijlocele necesare pentru o modelare mai realista a tranzitiilor cu timp de întârziere mai bune decât alte abordari din context. În special, competitia pentru resurse limitate diferentiaza QSDL fata de celelalte paradigme de modelare a performantei, care de obicei accepta doar tranzitii cu durate de timp deterministe sau durate de timp sub forma de interval. Tranzitiile cu întârziere a modelului QSDL sunt introduse prin procese solicitante. Întârzierea tranzitiei actuale depinde de situatia încarcarii întregului sistem. Aceasta permite o modelare mai realista a consumului de timp.

Stari temporizate Luând în considerare mecanismul asicron de timer al SDL, nu poate fi garantat ca procesele vor fi activate la un moment dat de timp pentru a realiza actiuni critice cu timpul, adica sa raspunda întreruperilor hardware. Aceasta face dificila modelarea unui sistem, în SDL, pentru cerinte de timp real ale hardware-ului. Este de preferat sa existe posibilitatea de activa procesele exact la momentul temporal necesar si sa existe garantarea acestui fapt. Aceasta permite proceselor sa reactioneze pe conditii fixate fara a consuma un semnal din coada de asteptare a procesului. Mecanismul de timer al SDL nu este convenabil pentru aceasta, deoarece semnalele timer sunt stocate în coada de intrare si consumate ca semnale ordinare. SDL ofera pentru activarea spontana a proceselor (semnale independente) conceptele de tranzitie spontana si semnale continue. Cu ambele concepte este posibila activarea unui proces indiferent de prezenta unui semnal de intrare în coada de intrare. Utilizarea semnalelor continue este restrictionata la semnalele continue în a caror definitie prioritatea este mai mica decât a semnalului consumat de tranzitie. Din aceasta cauza aceste semnale nu sunt convenabile pentru modelarea actiunilor critice. Tranzitiile spontane sunt mai convenabile pentru modelarea reactiei procesului indiferent de prezenta de semnal în coada sa de intrare. O specificatie SDL nu are informatii relative la momentul de timp la care se executa o tranzitie spontana.

QSDL introduce constructia "awake" care este utilizata pentru specificarea starilor temporizate. Noul mecanism activeaza tranzitiile spontane ale respectivului proces dupa o întârziere riguros specificata. Aceste întârzieri sunt date ca o expresie de tipul Duration dupa cuvântul cheie AWAKE la definirea starii QSDL. Fiecare stare a sistemului poate avea propria expresie awake. Daca $?_1$ este momentul de timp la care procesul intra în starea temporizata *s* si ? este temporizarea starii *s*, atunci timpul pentru awake al procesului va fi: $?_2 = ?_1 + ?$. La momentul $?_2$ respectivul proces poate executa una dintre tranzitiile spontane ale starii temporizate *s*. Daca în starea *s* sunt activate mai multe tranzitii spontane, atunci alegerea acestora se va face nedeterminist. Pe perioada duratei de timp dintre $?_1$ si $?_2$ nu este posibila nici o tranzitie de timp. În mod diferit de acesta, tranzitiile

consumatoare de semnal ramân valide daca semnalul necesar pentru activarea tranzitie este în coada de intrare a procesului.

Daca un proces iese dintr-o stare temporizata prin executarea unei tranzitii consumatoare de semnal timpul de awake este resetat. Semanticile pentru starile care nu au temporizare awake ramân neschimbate. În consecinta, se face distincti dintre starile temporizate si netemporizate pentru specificatiile QSDL.

Semnale temporizate QSDL ofera posibilitatea de a parametriza fiecare actiune de iesire dintr-un sistem QSDL printr-o expresie de tipul Duration. În acest sens, timpul de întârziere (timpul de transmisie) între emitator si receptor, într-un mediu fizic, modelat la rândul sau, poate fi modelat. Semantica iesirilor temporizate poate sa apara similara cu utilizarea canalelor cu temporizare din SDL, dar difera de acestea prin doua puncte cruciale:

Modelarea traficului si a încarcarii O alta topica importanta care afecteaza performanta sistemului este comportamentul surselor de trafic. Deci, caracterizarea încarcarii prin parametrii de trafic convenabili este o parte a oricarui model evaluabil cantitativ. QSDL accepta modelarea încarcarii prin diferite componente, în particular prin furnizarea unui numar de functii cu distributie aleatoare. Încarcarea trebuie impusa unui sistem, adica caracteristicile traficului ale sistemului de telecomunicatii actual si viitor pot fi descris prin generatoare de încarcare descrise, la rândul lor, ca procese QSDL. Semnalele generate de aceste procese, în final, leaga cererile de masini. Utilizarea surselor cu stari multiple ofera o mare flexibilitate în proiectarea surselor de trafic. Pe aceasta cale pot fi descrise caracteristici foarte diferite ale traficului: transfer de fisiere, transmisii digitale de voce sau video.

Senzori Sistemele critice relativ la timp (ca sistemele în timp real sau protocoalele de comunicatie) sunt considerate a fi corecte numai daca acestea îndeplinesc cerintele de performanta prespecificate (adica, cerintele de calitate a serviciilor – cerinte QoS). Ca o consecinta, nu numai corectitudinea functionala a sistemului trebuie examinata, dar, de asemenea, trebuie examinate si proprietatile nefunctionale relative la performanta comportamentului diferitelor componente ale sistemului care prezinta interes. Definirea cerintelor de performanta nu face obiectul QSDL, *dar scopul acestuia este de a defini masura performantelor sistemului modelat.* Bazat pe masura de performanta definita a unei specificatii QSDL, validarea cerintelor de performanta poate fi realizata pe parcursul executiei sistemului. Valorile de performanta, calculate prin masurile de performanta la momentul rularii acestora, pot fi verificate comparativ cu limite certe si/sau sunt utilizate pentru a influenta comportamentul functional al sistemului modelat.

QSDL ofera masuri de performanta în forma (în limbaj) de senzori. Un senzor poate fi privit ca o colectie de date pe care se calculeaza valorile de performanta la executia sistemului. Fiecare senzor este actualizat de anumite evenimente, cum ar fi consumarea de semnale sau cereri de serviciu care apar pe parcursul executiei unui sistem QSDL. Valorile care sunt utilizate pentru actualizarea unui senzor sunt apelate ca esantioane. Esantioanele sunt asociate unui eveniment de actualizare, de exemplu, evenimentul sosire semnal actualizeaza senzorul care numara semnalele sosite ale tipului de semnal sosit. În QSDL senzorii sunt grupati în trei categorii:

- Masurile generale ale sistemului si blocurilor
- Masurile specifice procesului
- Masurile specifice masinii

Tipurile de baza de senzori

- **Senzori de numarare** – Senzorii de numarare sunt utilizati pentru a numara esantioanele asociate cu un tip de eveniment. Senzorii de acest tip ofera patru tipuri de valori de performanta. Unele dintre masurile senzorilor de numarare sunt legate de un interval de timp. Lungimea acestui interval poate fi specificata pentru un astfel de senzor în declaratia de senzor.

- **Senzori de frecventa** – Senzorii de frecventa sunt o generalizare a senzorilor de numarare si calculeaza frecventa tuturor valorilor esantioanelor înregistrate. Aceasta poate fi realizata pentru un numar arbitrar de valori dintr-un domeniu dat. Domeniile utilizate sunt: tipuri de semnale, tipuri de procese, tipuri de servicii masina si, eventual, altele. Pentru fiecare valoare a domeniului, un senzor de frecventa va pastra un contor individual.

- **Senzori de pontaj** – Un senzor de pontaj calculeaza numarul de masuratori pentru o secventa de valori reale. Unele dintre acestea sunt legate de un interval de timp. Ca si în cazul senzorilor de numarare, lungimea acestui interval poate fi data la declararea senzorului de pontaj. Fie $x_1,\ldots,x_n$ o

secventa de valori reale si $t_1,\ldots,t_n$ timpii în care valorile sunt înregistrate de senzorul de pontaj. Sunt date mai multe masuri oferite prin utilizarea acestui tip de senzor

Senzorii la nivelul procesului Masurile acestui tip sunt specifice unei singure instante proces. În SDL este posibil sa existe un numar arbitrar de instante pentru fiecare tip de proces. Fiecare instanta de proces are propria copie a senzorului, care a fost declarat la definirea tipului procesului. Tipurile senzorilor specifici procesului sunt împartiti în cinci grupe:

a) Senzori de baza pentru masuri definite utilizator (<base sensor type>)

b) Senzori fara parametrii (<process sensor type>). QSDL accepta zece senzori proces diferiti fara parametrii.

c) Senzori care pot avea o lista de parametrii (<process sensor signal type>). Sunt definiti sapte senzori diferiti cu liste de parametrii semnal. Lista de parametrii este optionala pentru toti senzorii acestui tip. Semantica acestora este data în tabela. Daca exista lista optionala de parametrii, senzorul este restrictionat la semnalele tipului (tipurilor) respective. Semnalele care nu figureaza pe respectiva lista vor fi ignorate.

d) Senzori care pot sa aiba parametrii tipul serviciului masina (<process sensor machineservices type>). Senzorii specifici procesului cu lista de parametrii a tipului serviciu masina este prezentata în tabela. Lista de parametrii este optionala pentru toti senzorii acestui tip.

e) Senzori care au parametri de tipul stare (<process sensor state type>). Sunt definiti patru senzori diferiti pentru valorile de stare si sunt enuntati în tabela. În acest caz, parametrii senzorului nu sunt optionali si este permis numai un parametru de stare.

Senzori la nivel masina Senzorii specifici pentru masina sunt utilizati pentru a calcula masurile pentru o singura masina QSDL. Din punct de vedere sintactic un senzor masina este similar cu senzorul proces. Acestea vor diferi numai prin tipul de senzori.

4.4. SPECIFICAREA PROTOCOALELOR SI SERVICIILOR ÎN CONTEXTUL SDL, QSDL

Definitii externe

Abstract Syntax Notation One ASN.1	(Recomandarea ITU-T X.208)
Message Sequence Chart (MSC)	(Recomandarea ITU-T Z.120)
Specification and Description Language(SDL)	(Recomandarea ITU-T Z.100)
Tree and Tabular Combined Notation(TTCN)	(Recomandarea ITU-T X.292)

Definitii interne

Pentru scopurile acestui paragraf se vor aplica urmatoarele definitii:

Domeniul aplicatiei câmpul de activitati în care va opera sistemul care este supus specificarii

Documentare activitatea de creare a unui ETS din descrierea formala SDL si a altor informatii generate pe parcursul acestei metodologii, în concordanta cu anumite standarde pentru format si stil

Formalizare pasul activitatii în care se produce descrierea formala SDL a unui sistem

Descrierea formala SDL o descriere SDL care este conforma cu Recomandarea ITU-T Z.100, care nu contine text SDL informal si care poate fi astfel interpretata de instrumente automate

Descrierea informala SDL o descriere SDL care este conforma cu recomandarea ITU-T Z.100, dar care contine text informal; nu poate fi interpretata de instrumente automate

Validare proces, cu metodele asociate, procedurile si instrumentele prin care se face evaluarea

- faptului ca un standard este complet implementat, conform cu regulile pentru standarde si satisface scopul exprimat în documentul de cerinte pe care este bazat standardul.
- faptului ca o implementare care este conforma cu standardul are exprimata functionalitatea exprimata în documentul de cerinte pe care este bazat standardul.

Modelul validarii versiunea detaliata a unei specificatii, care poate contine si o parte a mediului sau, care este utilizata pentru a realiza validarea formala.

Formalizarea SDL (cu ASN.1 si MSC)

Scopul pasului de formalizare este de a produce o specificatie formala a unui sistem care sa fie utilizata ca baza pentru o ETS si pentru validare. ETSI sustine utilizarea recomandarii ITU-T Z.100 ca formalismul principal pentru descrierea comportamentului cu recomandarea ITU-T Z.120 pentru specificarea relatiilor fluxului de mesaje si recomandarea ITU-T X.208 pentru a descrie continutul de date al mesajelor.

Criterii de startare a formalizarii

Înainte ca formalizarea sa poata avea loc, trebuie sa existe urmatoarele informatii:

1) o descriere a entitatilor care trebuie modelate în recomandarea ITU-T Z.100 (descrierea functionala a entitatii, fluxuri de informatie si/sau MSC-uri, elementele serviciului, descrierea ASN.1 a datelor, descrierea actiunilor etc.)
2) o descriere a elementelor si a numelor pentru elementele principale ale sistemului
3) o descriere a fiecarei interfete normative

Procesul de formalizare este divizat în urmatorii pasi:

- pasi de structura (pasi **S**)
- pasi de comportament (pasi **B**)
- pasi de date (pasi **D**)

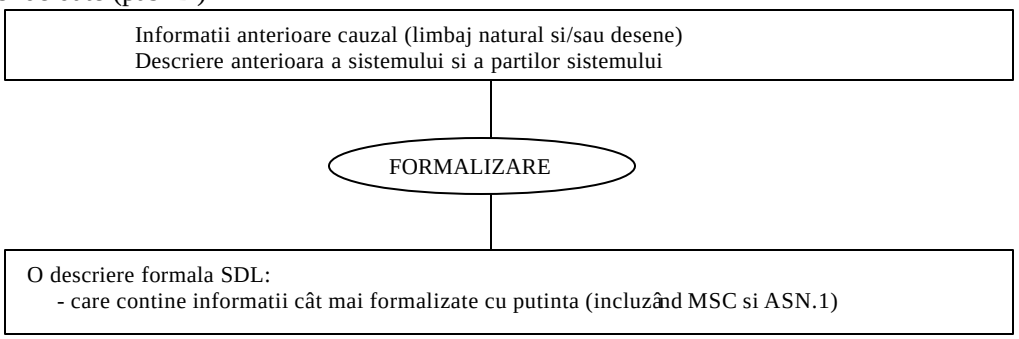


Figura 4. 50. Procesul de formalizare

Scopul pasilor S este de a defini în termeni de recomandare Z.100 a interfetelor interne si externe a sistemului si de a partitiona sistemul în blocuri SDL.

Pasul S1: mediul si frontiera sistemului

Instructiuni:

1) Identificarea frontierelor sistemului care trebui descris si mediul sau
2) Gasirea unui nume adecvat sistemului
3) Definirea unui diagrame SDL cu numele identificat si explicarea sistemului si a relatiilor sale cu mediul, informal, în diagrama SDL sistem

Reguli:

Regula 1: Frontiera sistemului descrie ceea ce va trebui specificat (descris). Entitatile comunicante din interiorul sistemului trebuie descrise, indiferent daca acestea sunt normative sau informative. Entitatile din afara frontierelor sistemul sunt presupuse existente, dar nu pot fi descrise în SDL. Comunicatia este posibila numai prin schimbul de mesaje discrete (semnale pentru SDL)

Regula 2: Unitatea pentru datele de timp trebuie înregistrate, într-un comentariu, în diagrama sistem.

Ghidari:

1) sistemul SDL poate fi un sistem închis, fara interfete. În acest caz, sistemul SDL descrie comportamentul care este subiectul unei ETS si entitatile care comunica cu aceasta. Aceste entitati sunt informative. Interfetele normative pentru ETS pot fi necesar, interfete interne sistemului SDL.
2) Daca MCS-urile au fost deja produse, acestea vor identifica axele corespunzatoare entitatilor considerate ca apartinând sistemului si separat axele entitatilor care apartin mediului exter n. Nu vor fi descrieri formale în specificatiile SDL (sau sunt furnizate numai ca parti informative ale sistemului)

Pasul S2: comunicatiile discrete ale sistemului

Instructiuni:

1) Identificarea entitatilor din afara sistemului a caror comunicare cu sistemul este o topica a specificatiei
2) Identificarea fluxului de informatie în termeni de mesaje discrete pentru a fi comunicate între sistem si fiecare entitate exterioara cu care se comunica
3) Modelarea datelor mesaje ca semnale definite în sistem
4) Statuarea relatiei dintre fiecare semnal si entitatile externe sistemului
5) Statuarea scopului fiecarui semnal printr-un comentariu din definitia semnalului

6) Plasarea semnalelor relationate (de obicei toate) într-o signallist
7) Includerea definitiei semnalului si a signallist în diagrama sistemului.

Reguli:

Regula 3: Definitia textuala a unei diagrame trebuie plasata în simbolurile test din interiorul diagramelor. Plasarea textului SDDL în simbolul text a unei diagrame identifica în mod clar textul ca fiind SDL si, de asemenea, identifica diagrama SDL de care acesta apartine.

Regula 4: Fiecare tip al definitiei (definitia semnalului, definitiile signallist, definitiile de date) trebuie plasat în simboluri text diferite. Daca definitiile textuale sunt mai mult de 50% a diagramei, trebuie avut pagini diferite pentru fiecare tip de definitie.

Ghidari:

1) Numarul de semnale va fi mai mic daca evenimentele relationate sunt comunicate cu un singur semnal ca parametrii (se poate introduce câte un semnal pentru fiecare tip de eroare sau un singur semnal cu parametrii pentru fiecare tip de eroare).
2) La introducerea unui semnal cu parametrii se va identifica sau se va denumi sortul de date corespunzator. Sortul de date trebuie sa corespunda unui tip ASN.1 sau SDL.
3) Definitiile semnalelor sau a signallist sunt de obicei prea mari pentru a fi incluse într-un simbol text pe prima pagina SDL a diagramei sistemului. Se vor adauga pagini suplimentare diagramei sistem pentru a contine aceste descrieri textuale. Nu se vor plasa în diagramele SDL declaratii lungi despre relatiile dintre semnale si mediul exterior sau descrierile care contin desene informale. Aceste declaratii vor fi referite de catre SDL. Definitiile scurte vor putea fi plasate cu definitii SDL adecvate.
4) Pentru a se putea asista documentarea, definitiile signallist sunt plasate înaintea definitiilor pentru semnalele utilizate în respectiva lista. Definitiile de date se vor plasa dupa definitiile de semnale. Definitiile sunt grupate prin plasarea definitiilor relationate în acelasi simbol text.

Pasul S3: Partile sistem

Instructiuni:

1) Identificarea partilor importante ale sistemului si desenarea acestora ca blocuri în sistem
2) Gasirea de nume adecvate pentru fiecare bloc si descrierea fiecarui bloc si a relatiilor sale cu mediul, informal, într-un comentariu din interiorul blocului. Pasii S3 la S6 se vor repeta pentru descompunerea blocului în sub-blocuri. La replicarea pasului termenul de sistem va desemna blocul superior. Diferenta dintre bloc si sistem este ca semnalele (lista semnalelor si datele) utilizate pentru a comunica cu structura superioara sunt definite în afara blocului dat în interiorul sistemului.

Reguli:

Regula 5: Nu trebuie sa fie mai multe de 5 blocuri la nivelul sistem (sau direct încorporate în interiorul unui bloc)

Regula 6: O definitie trebuie sa aiba un scop cât mai simplu care sa poata include toti utilizatorii unui item definit

Regula 7: Daca un bloc (proces sau procedura) este informativ si nu este o parte a blocului informativ superior, acesta trebuie sa fie adnotat ca informativ în diagrama care îl refera sau în diagramele pe care le refera, sau în ambele locuri.

Ghidari:

1) Blocurile sunt entitati care contin un aspect comportamental cu stari interne
2) Blocurile delimiteaza vizibilitatea. Din aceasta cauza, semnalele, sorturile si tipurile care sunt utilizate numai în interiorul unui bloc trebuie sa fie definite în acel bloc, astfel încât informatia este opaca nivelelor superioare si, în consecinta fac aceste blocuri mai usor de înteles. Acest principiu se va aplica de asemenea si pasilor S6 si S7 fiind exprimat, la modul general, în regula 6
3) Un bloc este informativ daca nu are comportamente care sa fie normative. Scopul unui bloc informativ este de a determina completitudinea sistemului, astfel încât functia sistemului sa poata fi:

- înteleasa prin utilizarea acestora si a interactiunilor cu aceste parti informale
- executata si analizata si sa sustina validarea

Un bloc poate fi informativ numai daca toate blocurile din interiorul sau, procese si proceduri (incluzând si procedurile la distanta) sunt informative. Odata ce un bloc (procedura, proces) a fost marcat ca fiind informativ, toate blocurile (procedurile, procesele) din interiorul acestuia sunt informative si nu mai trebuie marcate, ca atare. Informativ nu are

aceiasi semnifi catie cu informal. În descrierea formala a SDL atât partile informative, cât si cele normative trebuie sa fie formale.

4) Când numarul de blocuri direct încapsulate este prea mare, în interiorul sistemului (sau a blocului) o parte dintre acestea se vor grupa si se vor încapsula într-un bloc (pasul 6)

Pasul S4: Caile de comunicare dintre parti

Instructiuni:

1) Identificarea canalelor necesare între blocuri si frontiera diagramei si între blocurile din interiorul diagramei
2) Pentru fiecare canal, se va face identific area directiei de comunicare
3) Se va asocia op lista de semnale pentru fiecare directie a canalului
4) Se va selecta numele semnalului în functie de utilizarea si functia comunicatiei.

Reguli:

Regula 8: Trebuie sa fie un singur canal între doua blocuri

Regula 9: Fiecare canal normativ trebuie sa aiba un comentariu "normativ" atasat

Ghidari:

1) Lista de semnale identificata în pasul S2 corespunde unuia sau mai multor canale legate într-un punct pe frontiera sistemului, cea ce din exterior trebuie privit ca o interfata, pe care pot sosi mai multe canale din blocuri diferite. Pot fi mai multe liste de semnal pentru un singur canal. Fiecare punct de pe frontiera reprezinta comunicarea cu o entitate externa diferita. Pot fi unul sau mai multe canale care conecteaza un as tfel de punct la blocurile din diagrama sistem SDL.
2) Blocurile sunt conectate la alte blocuri si/sau la mediu prin canale, în concordanta cu fluxul de informatie.
3) Cazurile reprezentate tipic, în MSC sprijina identificarea canalelor.
4) Efectele speciale care depind de canale cu temporizare sau fara nu sunt utilizate în mod normal. Canalele se considera cu temporizare; exceptie fac cele care sunt cerute în mod expres fara.
5) Daca o comunicare pe un canal necesita mesaje specifice, cu un format si o codificare specifica, canalul este normativ. Canalele interne sistemului care nu identifica o interfata particulara pentru producerea testelor sau pentru alte scopuri sunt în mod obisnuit informative. Comunicatia pe canalele informative contribuie la comportamentul sist emului, dar pot fi un sistem alternativ cu canale sau comunicatii diferite care sa aiba acelasi comportament.

Pasul S5: Asocierea semnalelor cailor de comunicatie

Instructiuni:

1) Pentru fiecare lista de semnale se identifica semnalele adecvate
2) Definirea si denumirea fiecarui nou semnal

Reguli:

Regula 10: Într-un simbol signallist nu pot sa apara mai mult de trei semnale (sau liste de semnale). Se vor utiliza liste de semnal consistente.

Regula 11: Listele de semnale, semnalele si datele utilizate în toate comunicatiile unui sistem trebuie sa fie definite într-unul sau mai multe simboluri text separat de alte definiti.

Ghidari:

1) Desi SDL permite simbolul lista semnal asociat unui canal pentru a contine în mod explicit lista numelui semnalelor, aceasta nu este recomandata. Lista semnalelor este uzual necesara în mai multe locuri si este mult mai simplu de realizat modificarile într-un singur loc: signalliust
2) Pentru redefinirea unei liste semnal asociate unui canal pe nivel superior, se va utiliza o noua lista, cea ce va implica structura si claritatea SDL-ului. Definitia listei de semnal trebuie sa fie definita în diagramele de nivel superior celui în care este utilizata.
3) Necesitate de semnale noi pentru comunicarea dintre blocuri, necesita a fi definite în unitatea (bloc sau sistem) care contine blocurile comunicante.

Pasul S6: Informatia ascunsa si sub-structurarea

Instructiuni:

1) Se va considera fiecare bloc în interiorul diagramei curente pentru a decide care dintre acestea vor contine blocuri sau procese
2) Daca blocul va contine blocuri, se vor aplica recursiv pasii S3 la S6 blocului, care va fi privit ca (sub)sistem si care va introduce noi semnale necesare, blocuri, canale si liste de semnale
3) În alte conditii se va aplica pasul 7 care va diviza blocul în procese

Reguli:

Regula 12: Diagramele vor fi imbricate prin referire si numai prin exceptie prin încapsulare

Regula 13: Numarul canalelor pentru fiecare bloc nu trebuie sa depaseasca trei.

Ghidari:

1) Daca sistemul este de mari dimensiuni, anumite blocuri vor fi considerate, ele însele, sisteme, putând fi în continuare partitionate în concordanta cu regulile care se aplica sistemelor. Acesta se va produce în blocuri imbricate. O entitate care are ca trasatura principala comportamentul si nu este, în continuare partitionata este o candidata ca proces; entitatea direct încapsulata este un bloc. Entitatea care urmeaza a fi partitionata este uzual un bloc.
2) Este posibil sa nu apara clar faptul ca interiorul unui bloc va fi bloc sau proces, ceea ce va duce la împartirea initiala în blocuri. Daca este dificila partitionarea unui bloc, în continuare, se poate presupune ca este vorba despre un proces.
3) Aplicarea recursiva a acestui pas va duce la un numar de nivele. Chiar daca sistemul este complex nu pot sa apara mai mult de trei nivele de blocuri.
4) Numarul de procese dintr-un bloc este limita la cinci: în cazul unui bloc cu mai multe procese acesta va fi divizat pentru a se obtine blocuri cu un numar mai mic de procese. Interactiunea dintre axele MSC corespunzatoare proceselor poate sprijini identificare clusterelor naturale ale proceselor pentru fiecare bloc nepartitionat
5) Trebuie definit arborele de diagrame a unui bloc

Pasul S7: Constituentii blocului

Instructiuni:

1) Identificarea entitatilor separate cu comportament pentru blocurile selectate în pasul S6 pentru a fi divizate în procese si definirea acestor entitati ca procese ale blocului.
2) Gasirea de denumire adecvate pentru fiecare proces si descrierea acestora si a relatiilor acestora în mediu, informal, într-un comentariu din interiorul referintei procesului
3) Pentru fiecare proces se va defini numarul initial si cel maxim de instante.
4) Se vor utiliza rutele semnal pentru a conecta seturile de procese la canale pe frontiera blocului.

Reguli:

Regula 14: pentru fiecare bloc, cel putin un proces va avea numarul de instante initiale mai mare ca zero, adica va crea alte instante în bloc.

Regula 15: numarul de definitii de procese dintr-un bloc nu este mai mare de 15

Ghidari:

1) Rutele semnal din interiorul unui bloc pot fi derivate de o maniera similara ca pentru canale, specificata în pasii S3 si S4
2) În cazul în care procesele au fost încapsulate într-un bloc care contine numai un proces, blocul nepartitionat va contine o descriere de proces a unui singur tip si relatia cu mediul este derivata din interfetele externe blocului.

Pasul S8: Semnale locale în bloc

Instructiuni:

1) Identificarea semnalelor între procesele locale din bloc
2) Definirea la nivelul blocului a acestor semnale care sunt suplimentare (nu sunt definite în exteriorul blocului)
3) Identificarea tuturor procedurilor importate si exportate ale proceselor din bloc si a definitiilor procedurilor la distanta corespunzatore la care se adauga definirea acestora la nivelul blocului (sau a tipului blocului)

Pasii pentru comportament

Acesti pasi sunt utilizati pentru descrierea comportamentului componentelor în termeni de comunicatie, dar fara a furniza definitii formale pentru datele care sunt transportate prin pasii de date. Aceasta subclauza descrie pasii pentru comportament pentru un proces SDL, dar atât acestia, cât si pasii pentru formalizarea datelor sunt adecvati pentru definirea unui comportament al procedurii SDL.

Pasul B1

Instructiuni:

1) Identificarea alfabetului de intrare al procesului, denumit lista de semnale ale procesului, care este realizat prin identificarea:

- semnalelor care pot fi receptionate de catre proces
- procedurile exportate de catre proces. Acesta este cazul în care procesul actioneaza ca un server într-un model client-server, în care semnalul corespunzator este implicit, iar numele procedurii la distanta poate fi considerat ca parte a alfabetului
- expirarea timerelor este receptionata în aceiasi maniera ca si semnalele originale. Timerele care apartin unui proces trebuie declarate într-un simbol text al procesului. Durata la care este setat un timer trebuie sa aiba un nume simbolic utilizând un sinonim sau o variabila (de tipul durata).

Ghidari:

1) Desi lista de semnale poate fi derivata, uzual, simplu din diagrama blocului încapsulat, scopul acestui pas este de a revizui lista de semnale luând în considerare numai procesul curent. În cazul în care se decide modificarea setului de semnale de la alte procese, diagramele bloc corespunzatoare trebuie schimbate. Procesele care emit semnale vor trebui aduse la zi, adica au fost deja formalizate. O sursa de semnale care nu este reprezentata în diagrama blocului sunt semnalele emise de proces catre sine însusi, sau catre alta instanta a aceleiasi definitii proces.
2) Lista de semnale este utilizata la deciderea comportamentului unui proces, în fiecare stare din pasul B4. Se recomanda pastrarea unei înregistrari a listei semnalelor, adica acestea nu pot fi derivate automat prin instrumentele utilizate pentru SDL.
3) Se utilizeaza, pentru o procedura, semnalele blocului încapsulat, dar pot fi identificate semnale noi care trebuie adaugate listei de semnal a procesului.

Pasul B2: Scheletul proceselor

Instructiuni:

1) Producerea MSC pentru, cel putin, a cazului tipic. Acest fapt este foarte important atât pentru identificarea semnalelor, cât si pentru descrierea comportamentului de baza
2) Producerea unui schelet al procesului prin reprezentarea din MSC, luând în considerare numai cazurile tipice:

   2.1. Se porneste de la simbolul de start al procesului, se construieste un arbore de stari luând în considerare schimbarile de stare normale ale procesului.

   2.2. Se construieste arborele procesului prin ramificarea la nivelul fiecarei stari bazate pe fiecare intrare care este consumata si luata în considerare în MSC si urmând tranzitiile (incluzând iesirile si crearile) la diferitele stari

   2.3. Identificarea unei stari ca fiind diferita de alta stare, daca are un set distinct de semnale, care sunt consumate sau salvate, sau au raspunsuri diferite la un semnal dat.

   2.4.Includerea supervizarii timpului (etare es etare) si intrarea timer corespunzatoare

   2.5. Dupa definirea arborelui, se vor identifica nodurile de revenire si se va prezenta arborele într-un graf
3) Se transpune graful într-o diagrama proces
4) Se va determina daca procesul are doua sau mai multe seturi distincte de interfete pentru parti diferite ale comportamentului care pot fi interschimbate, iar - daca acele comportamente sun independente – procesul se va diviza în mai multe procese.

Reguli:

Regula 16: Tranzitiile spontane nu sunt utilizate în partea normativa a standardelor

Regula 17: MSC trebuie sa fie:

a) trasarea corecta a manevrarii semnalelor de catre sistemul SDL
b) adnotat explicit pentru a indica modul si cauza pentru care difera de comportamentul SDL

Ghidari:

1) Doua procese cu N si respectiv M stari, atunci când se combina ca un singur proces, acesta din urma va avea NxM stari.
2) O procedura sau un task informativ poate avea tranzitii spontane, care starteaza cu none (din SDL) pentru modelarea comportamentului utilizator sau pentru alte evenimente imprevizibile.

Pasul B3: Procesele informale

Instructiuni:

1) Identificarea combinatiilor utilizarii cazurilor
2) Identificarea informatiei pe care procesul o stocheaza si verificarea faptului ca aceasta este implicita în starea procesului sau necesita date interne
3) Utilizarea acestei informatii pentru definirea actiunilor fiecarui proces

4) Adaugarea de task-uri sau proceduri si decizii în tranzitii, utilizând numai text informal pentru task-uri si decizii.
5) Utilizarea unei proceduri se va face prin furnizarea numelui complet, iar ulterior definirea acesteia prin folosirea pasilor B1 la D9

Ghidari:

1) Utilizarea MSC –urilor existente si generarea de MSC pentru a identifica combinatiile utilizatorilor
2) Se va decide utilizarea unui task sau a unei proceduri pentru descrierea procesarii informatiei dintr-o tranzitie, desi aceasta poate fi modificata ulterior.
3) Actiunile care se vor desfasura în mai multe locuri, în proces se vor colecta într-o procedura
4) În cazul în care procedura însasi are stari, trebuie retinut ca numai semnalele mentionate în stare sunt explicitate, toate celelalte semnale ale procesului care a apelat procedura pot fi consumate implicit, daca nu sunt mentionate într-un simbol save.

Pasul B4

Instructiuni:

1) Identificarea pe fiecare stare si pentru fiecare element din lista de semnal (semnal sau procedura la distanta) daca semnalul sau respectiva procedura este intrare prin proces sau este salvat
2) Daca elementul este intrare, se va determina tranzitia (poate fi tranzitia implicita pe aceiasi stare)
3) Se va relua instructiunea 1 si 2 pâna la epuizarea listei de semnale si a tuturor elementelor
4) Se va analiza fiecare proces si apoi combinatiile acestora la nivelul sistemului SDL pentru a verifica proprietati nedorite, iar la nevoie se vor reproiecta.

Reguli:

Regula 18: Toate starile unui proces sunt tangibile din starea de star a procesului

Regula 19: O procedura care este exportata de catre un proces (pentru a fi utilizata ca procedura la distanta) nu trebuie salvata în fiecare stare a procesului

Regula 20: Fiecare semnal primit de catre proces trebuie sa aiba cel putin o intrare legata la o tranzitie nevida.

Ghidari:

1) Matricea semnalelor unei stari poate fi utilizata pentru a verifica actiunea fiecarui semnal în fiecare stare. La identificarea unei noi stari, se va extinde matricea. Aceasta matrice poate utiliza si pentru identificarea a doua stari cu acelasi comportament, care pot fi combinate, sau a doua semnale care au aceiasi stare urmatoare. În aceste cazuri se pot reduce numarul de semnale sau de stari
2) Se poate realiza o diagrama de ansamblu a starii pentru a da o privire generala asupra comportamentului procesului

Pasii de date

Scopul pasilor de date este de a furniza o definitie formala a datelor utilizate în procese. Fara date formale, orice decizie asupra comportamentului sau a operatorilor utilizati în expresii vor avea rezultate incerte

Pasul D1 Parametrii semnalului

Instructiuni:

1) Identificarea valorilor transportate prin semnale, începând cu semnalele la nivel sistem
2) Analiza sorturilor de date predefinite pentru a reprezenta valorile identificate
3) Extinderea definitiei semnalului cu sortul de date
4) Identificarea si definirea a noi sorturi de date, daca acesta este necesara
5) Semnalele de expirare a timerelor pot avea parametrii ca orice alt semnal. Sortul de parametrii trebuie sa fie definiti la definirea timerelor.

Pasul D2 Parametrii procedurilor si proceselor

Instructiuni:

1) Identificarea sorturilor de date necesare pentru parametrii procesului (procedurii)
2) Statuarea rolului fiecarui parametru printr-un comentariu în antetul procesului (procedurii)

3) Parametrii procedurii pot fi definiti ca IN sau ca IN/OUT. IN trebuie utilizat pentru a transporta valori de la procesul apelant la procedura apelata. IN/OUT se va utiliza pentru ambele sensuri.

Pasul D3 Variabilele semnal

Instructiuni:

1) Adaugarea de parametrii pe intrare în concordanta cu definitia semnalului
2) Definirea variabilelor cerute pentru valorile de intrare
3) Statuarea rolului fiecarei variabile într-un comentariu

Pasul D4 Tranzitii formale

Instructiuni:

1) Înlocuirea textului informal din taskuri, decizii si raspunsuri cu asignari formale, expresii formale, expresii de domenii formale si apeluri de proceduri si identificarea oricarui nou operator utiliza în expresiile formale care trebuie adaugat sorturilor de date din pasul D6
2) Definirea de variabile suplimentare si sinonime, daca este necesar
3) Definirea de proceduri suplimentare daca este necesar
4) Adaugarea de parametrii apelurilor procedurii în concordanta cu definitia acesteia

Pasul D5 Iesiri si crearea de parametrii

Instructiuni:

1) Adaugarea expresiilor pentru iesiri utilizând variabilele si sinonimele introduse
2) Adaugarea parametrilor actuali pentru a crea actiuni

Pasul D6 Semnaturile datelor

Instructiuni:

1) Identificarea si definirea sorturilor de date si valori (sinonime) care trebuie definite
2) Daca unele dintre valori sunt identificate ca fiind dependente de instalarea actuala a sistemului, acestea se vor exprima prin utilizarea sinonimelor externe
3) Pentru fiecare sort de date solicitat se va crea newtype si syntype (din SDL) cu numele complet sau se defineste un tip ASN.1.
4) Identificarea fiecarui nou tip de operator care trebuie definit. Daca nu mai sunt operatori noi procesul de formalizare se va considera complet.

Pasul D7 Descrierea datelor informale

Instructiuni:

1) Adaugarea axiomelor informale în formatul textului informal pentru definirea de noi tipuri.

Pasul D8 Descrierea datelor formale

Instructiuni:

1) Pentru semnatura fiecarui operator se va adauga definitia unui operator, în formatul unei diagrame operator, daca acesta este posibil. Daca toti operatorii pot fi definiti pe aceasta cale atunci formalizarea este completa
2) În caz contrar, se vor formaliza axiomele prin înlocuirea axiomelor informale cu unele formale care statueaza proprietatile esentiale ale operatorilor.
3) Textul informal al axiomelor va fi utilizat pentru comentariu la axiomele formale

Pasul D9: Completarea formalizarii datelor

Instructiuni:

Se vor adauga axiome (ca definitii operator) noilor tipuri de date, pâna când acestea vor fi complete (i.e. pâna când toate expresiile care contin operatori non-constructor si literarele pot fi rescrise în expresii.