Cuvintele sunt cu adevărat mijlocul de comunicare cel mai puţin eficient.
Ele sunt cele mai expuse la interpretări greşite şi cel mai adesea prost înţelese.

Neale Donald Walsch

Prezentare
Unified Modelling Language

Conţinut:

1. Ce este limbajul UML.
2. Scopul utilizării UML.
3. Istoric UML. Standardizare.
4. Unelte UML.
5. Prezentarea limbajului
6. Exemple de utilizare
7. Referinţe

1. Ce este limbajul UML ?

- UML (Unified Modelling Language) este, aşa cum îi spune şi numele, un limbaj care se foloseşte la modelarea activitatilor în general, a celor de business, a specificaţiilor software şi oriunde există posibilitatea ca limbajul uman sa dea naştere la interpretări greşite.
- Este utilizat în inginerie, în special în ingineria software. Este in mod natural orientat pentru programarea OO dar poate fi utilizat uşor şi pentru limbajele procedurale.
- „O imagine exprimă cât o mie de cuvinte” – de aceea se utilizeaza pictogramele standardizate numite şi diagrame pentru a reprezenta ceea ce dorim să comunicăm.
- Limbajul UML ne permite realizarea mai multor view-uri, ca nişte fotografii din diverse unghiuri, ale unei realităţi, astfel încât această realitate să fie surprinsă prin toate aspectele ei relevante.
- În software vom utiliza două puncte de vedere necesare unei descrieri suficiente a realităţii: (1) structural şi (2) comportamental (behavioral).

2. Scopul utilizării UML

- Utilizatorii au la îndemână un limbaj expresiv, vizual, astfel încât să poată dezvolta şi schimba modele comprehensive;
- Este extensibil, dar şi specializat;
- Este independent de orice limbaj de programare particular;
- Constitue o bază formală pentru înţelegerea limbajelor de modelare;
- Încurajează creşterea pietei de unelte OO;
- Suportă concepte înalte de dezvoltare, cum ar fi: colaborarea, pattern-uri, cadre de lucru şi componente;
- Integrează experienţe de „bună practică”.

3. Istoric UML si standardizare

- Dezvoltarea UML a început la sfârşitul anului 1994, cand Grady Booch şi James Rumbaugh de la Rational Software Corporation au început sa-şi unifice munca.
- În toamna lui 1995 Ivar Jacobson si compania sa Objectory Company s-au reunit cu Rational rezultand astfel metoda OOSE (Object-Oriented Software Engineering).
- În 1995 s-a decis standardizarea limbajului de modelare la care ei lucrau şi s-a stabilit un acord pentru îndeplinirea standardelor metodologiilor sub tutela OMG (Object Management Group)
- UML se răspandeşte tot mai mult in 1996, astfel Rational stabileşte parteneriat cu cateva firme pentru a defini un UML 1.0 puternic: Digital Equipment, HP, IntelliCorp, IBM, Microsoft, Oracle, Rational Software.
- A fost astfel definit un UML puternic care a fost standardizat în ianuarie 1997 de către OMG.
- Alte companii, printre care şi IBM au trimis separat un raspuns la RFP (Request For Proposal).
- Acestea se unesc cu partenerii UML şi astfel apare UML 1.1. Faţa de UML 1.0 avea mai multe imbunataţiri, printre care şi claritatea semantică şi alte contribuţii de “best practices” de la noii parteneri şi se standardizează în 1997.
- Acum s-a ajuns la versiunea 2.5 (toamna 2018), care este standardizată.

4. Unelte UML

- Firme proprietare:
- De la Microsoft, în pachetul Microsoft Office Professional, există si Visio.
  - Este destul de uşor de învăţat, se bucură de un pachet larg de template-uri, precum şi elemente predefinite specifice modelării în domenii ca: Inginerie Mecanică, activităţi de business, planuri de clădiri, programarea activităţilor în proiectare, software, pagini web, etc.
- De la Visual Paradigm, avem o suita de unelte printre care:

  Visual Paradigm UML cu template-uri specifice pentru:
  - modelarea OOP;
  - modelarea bazelor de date;
  - activităţi de business;
  - colaborarea în cadrul echipei;
  - integrare buna cu IDE-uri ca Eclipse, IntelliJIdeea, JBuilder, NetBeans, etc
  - interoperabilitate crescuta, putând să inglobeze fără efort digrame facute în Visio, sau alte unelte.
  - Foarte bun tutorial, audio-video, foarte bună documentaţie.

4. Unelte UML

- Enterprise Architect de la Sparx Systems
  - Foarte complex, destul de greu de utilizat, dar bine documentat;
  - generează o serie de diagrame pornind de la cod Java, C#, Visul C++
  - se bucura de un larg set de elemente specifice standardului UML
  - unul dintre cele mai bune tutoriale găsite pe Internet
- Unele plug-ins care functionează în IDE-uri precum IntelliJIdeea, Eclipse, etc
  - Pentru IntelliJIdea se pot genera automat Class Diagram si Sequence Diagram pornind de la clasele Java.
  - Face si reverse engineering, pornind de la pachete si biblioteci Java.
- Alte produse:
  - Rational Software Architect, de la IBM;
  - Rational Rose XDE în tradiţia Rational Rose, suportă UML 1.x;
  - SmartDraw pentru Windows

4. Unelte UML

Non proprietare:

Unelete incorporate în IDE-uri Open Source:

•Eclipse cu **Eclipse Modeling Framework** (EMF) şi UML

•**AmaterasUML**, pentru Java şi Eclipse. Poate face şi reverse engineering din cod Java (licenţa BSD).

•**UMLDesigner** - pentru .NET, suportă generare de cod C#.

•Alte unelete free pe Internet, dar nu se bucura de complexitatea şi interactivitatea celor de mai sus (**Star UML**);

5. Prezentarea limbajului. Standard UML 2.x

Defineşte 13 diagrame:

- Diagrame pentru pachete (package diagrams);
- Diagrame de clasă sau structurale (class diagrams);
- Diagrame pentru obiecte (object diagrams);
- Diagrame pentru structuri compuse (composite structure);
- Diagrame pentru componente (component diagrams);
- Diagrame de desfasurare (Deployment diagram);
- Diagrame pentru cazuri de utilizare (use case diagrams);
- Diagrame de activitaţi (Activity diagrams);
- Diagrame pentru starea maşinii (State machine diagrams);
- Diagrame de comunicare (Communication diagrams);
- Diagrame de secvenţă (Sequence diagram);
- Timing diagram;
- Diagrame de interactiune (Interaction Overview Diagrams).

Reutilizarea Codului

- În programarea orientată obiect reutilizarea codului se poate realiza prin:
  - ¤Compunere (Agregare) – includerea de obiecte în cadrul altor obiecte
  - ¤Moştenire – posibilitatea de a defini o clasă extinzând o altă clasă deja existentă
  - ¤Clase şi funcţii template

Tipuri de agregare

- Există două variante de agregare diferențiate de relația dintre agregat și subobiecte

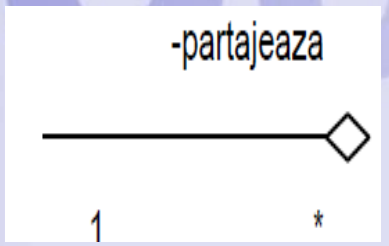
¤Compoziția – subobiectele agregate aparțin exclusiv agregatului din care fac parte, iar durata de viață coincide cu cea a agregatului

¤Agregarea propriu-zisă - subobiectele au o existență independentă de agregatele ce le partajază. Mai mult, un obiect poate fi partajat de mai multe agregate

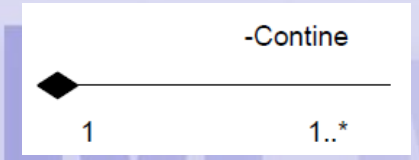
Relaţiile

- Toate relaţiile din cadrul diagramei de clase sunt reprezentate grafic printr-o succesiune de segmente orizontale sau verticale care leagă o clasă de alta.
- Relaţiile principale pot fi:
  - ¤ asocieri
  - ¤ generalizari
  - ¤ dependenţe
  - ¤ relaţii de rafinare.
- Asocierea este o conexiune între clase, ceea ce înseamnă o legătură semantică între obiectele claselor implicate în relaţie.

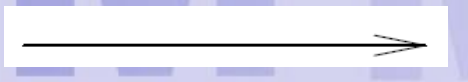
Notatii



Reprezentarea grafică în UML a unei agregări partajate



Reprezentarea grafică în UML a unei compoziţii

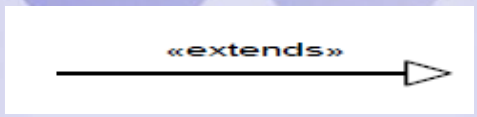


Reprezentarea grafică în UML a unei tranziţii

Reprezentarea grafică în UML a unei dependenţe

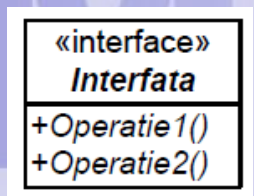
Reprezentarea grafică în UML a unei relaţii de derivare

Notatii



Reprezentarea grafică în UML a unei relaţii de extensie

Reprezentarea grafică în UML a unei relaţii de realizare



Simbolul folosit în UML pentru reprezentarea interfeţelor

«Stereotip» Nume_MetaClasa::**Nume_Clasa**
+Atribut public : Tip_Data = Valoare_Initiala #Atribut protejat : Tip_Data = Valoare_Initiala -Atribut privat : Tip_Data = Valoare_Initiala
+Operatie publica(in Parametru_1 : Tip_Data = Valoare_implicita, out Parametru_N : Tip_Data) : Tip_Data #Operatie protejata(in Parametru_1 : Tip_Data, inout Parametru_i : Tip_Data, out Parametru_N : Tip_Data) : Tip_Data -Operatie privata() : Tip_Data

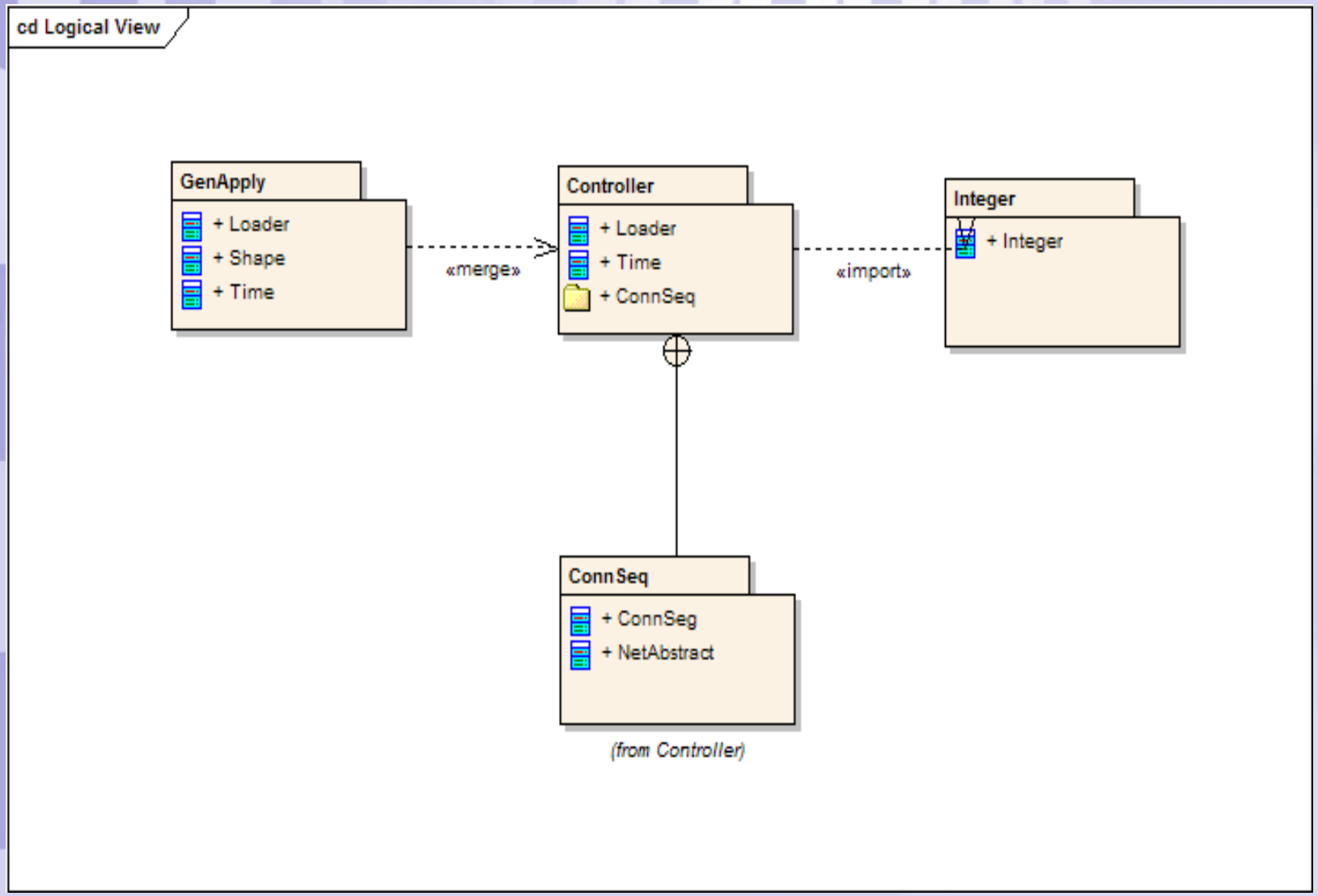
Simbolul folosit în UML pentru reprezentarea claselor

a. Diagrame pentru pachete

- Sunt utilizate pentru a reflecta organizarea pachetelor şi a elementelor sale.
- Cel mai des sunt utilizate în organizarea diagramelor use case sau class diagrams.
- Reprezintă fie relaţii logice, fie fizice;
- Se vor include în acelaşi pachet clase legate prin relaţia de compunere, colaborare, care moştenesc aceeaşi clasă de bază, etc.
- *Conectori:*

  - package merge –generalizare între elementele din sursă şi cele din destinaţie cu acelaţi nume;
  - package import – sunt referite de pachetul sursă;
  - conectori de includere (nesting connectors)- sursa este complet inclusă în destinaţie.

a. Diagrame pentru pachete(2)



b. Diagrame de clasă (class diagram)

- Reprezintă baza pentru orice sistem OO. Este o viziune statică asupra modelului sau a unei părţi din el, cu atribute şi metode.

Relaţii reprezentate:

- agregarea uşoară – arată utilizarea, nu neapărat conţinerea
- agregarea compusă - arată proprietatea sau conţinerea într-o anumită clasă

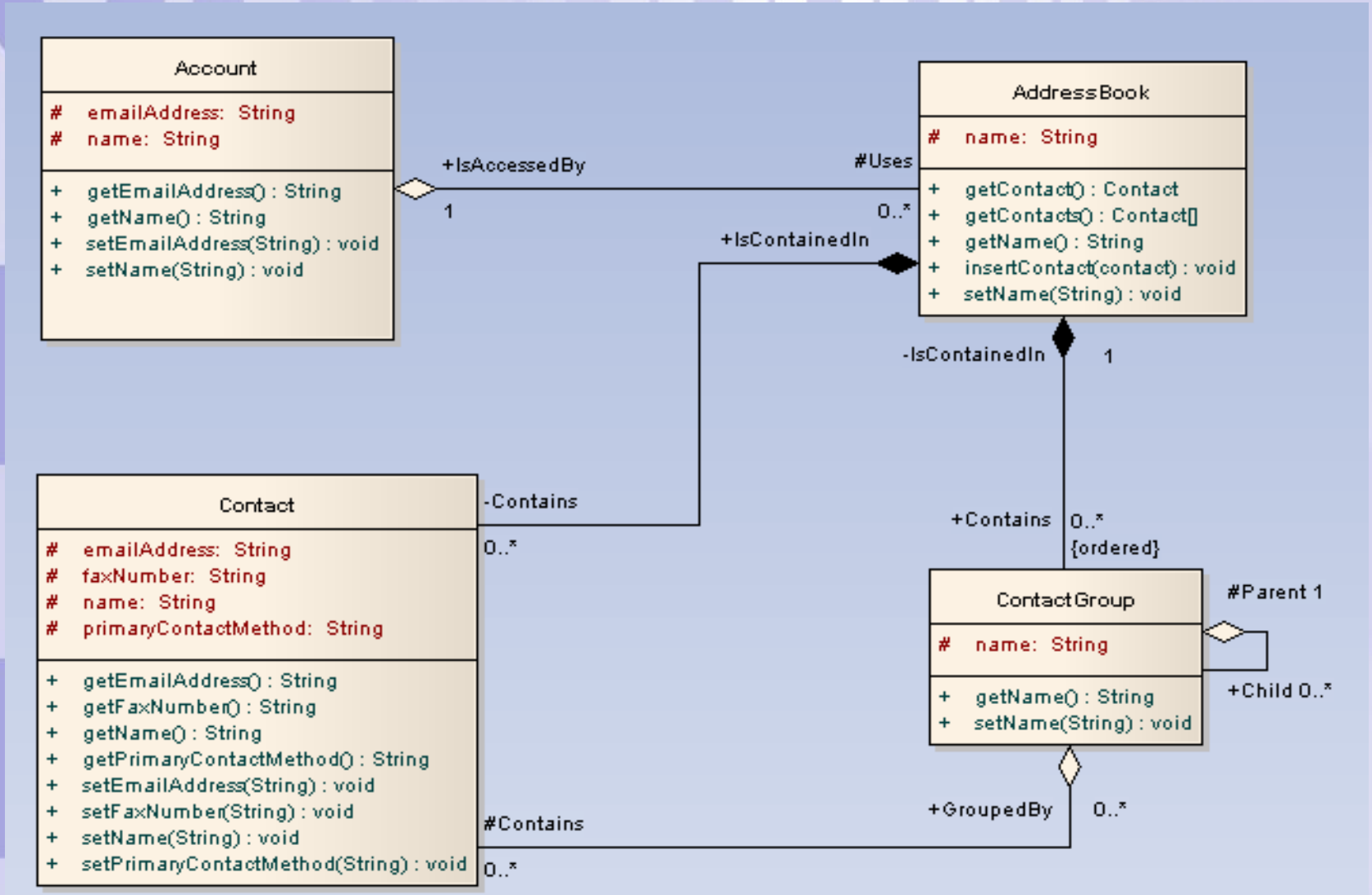
Notaţia pentru atribute: + public, - privat, # protected, ~ friendly.

Cazuri particulare: Interfeţe şi Tabele ale bazelor de date

Relaţii exprimate prin conectori:

- Asocierea- usual implementată ca o variabilă de instanţă in una din clase. Poate cuprinde numele rolului, cardinalitatea, direcţia şi constrângeri.
- Generalizarea – indică moştenirea.
- Agregarea- o clasă compusă din alte clase
- Agregarea compozită – atată dependentă puternică. De regulă, când părintele este şters, se şterg şi părţile dependente
- Asocierea – conector descris prin atribute şi relaţii. De exemplu, un angajat poate lucra concomitent la mai multe proiecte odata şi poate avea caracteristici diferite.
- Dependenţa – este din ce în ce mai puţin folosită, se vor folosi stereotipii, sau alţi conectori specifici;
- Traces – este adesea utilizată pentru a urmări schimbarile sau cerinţele modelului.
- Realizarea – sursa implementează sau realizează destinaţia
- Înglobarea - sursa înglobează destinaţia – caz tipic: inner classes.

b. Diagrame de clasă (class diagram)



Specificarea Atributelor

- Sintaxa specificării atributelor din compartimentul atributelor este urmatoarea:

vizibilitate idAtribut : tip = valoare_implicitia

unde:

¤ vizibilitate reprezintă protecția atributului și poate să aibă una din următoarele valori

+ = public,

- = privat si

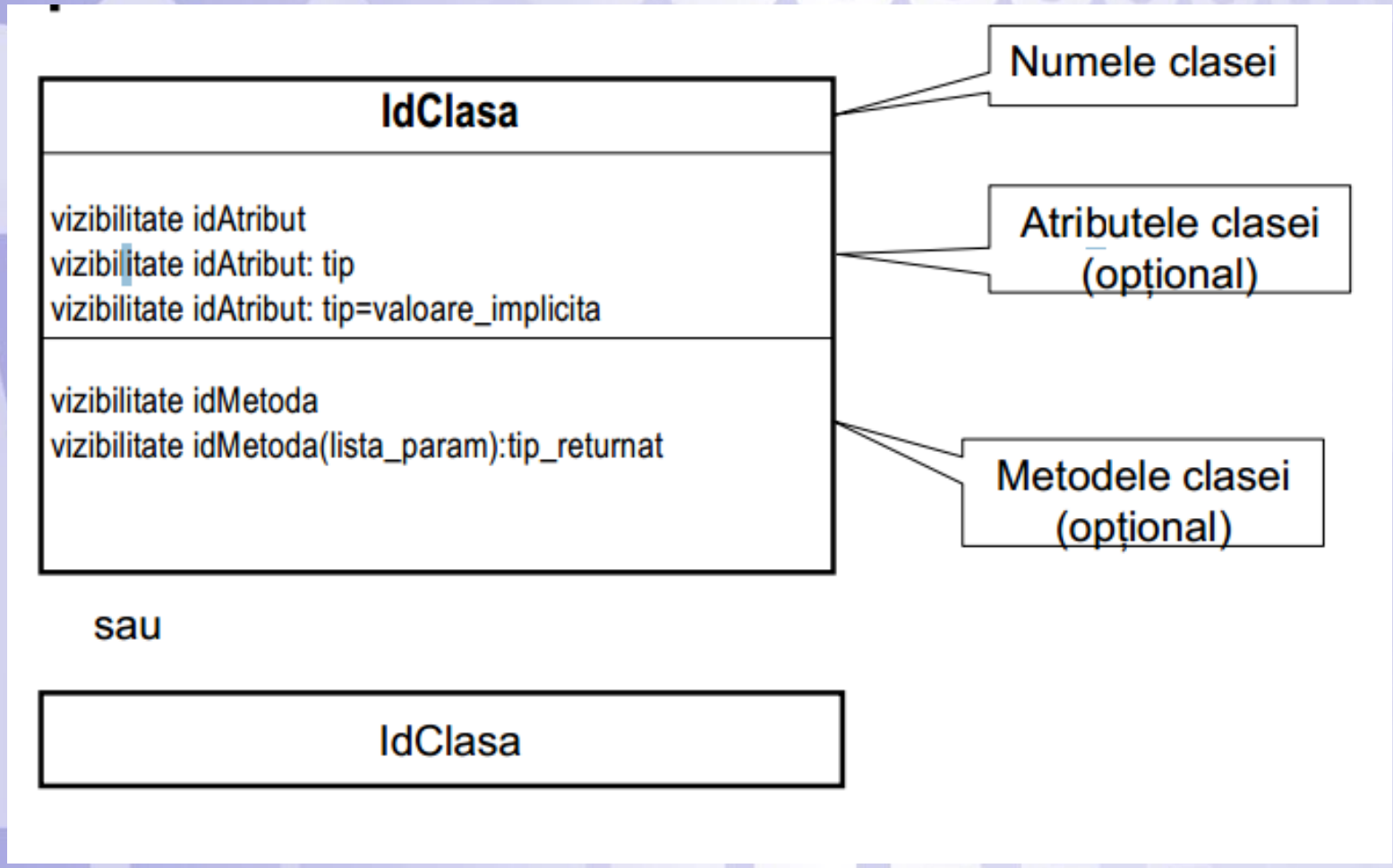
= protected (optional)

¤ idAtribut este identificatorul atributului

¤ tip – este tipul acestuia

¤ valoare_implicita reprezintă valoarea inițială a atributului (optională)

Reprezentarea generala a unei clase



Specificarea Metodelor

- Sintaxa specificarii metodelor sau operatiilor din compartimentul metodelor este urmatoarea:

  vizibilitate idMetoda(idP1:tip1, ..., idPn:tipn) : tip_returnat

  unde:

  ¤ vizibilitate reprezintă protecția metodei. Poate să aibă una din următoarele valori

  + = public,

  - = privat si

  # = protected (optional)

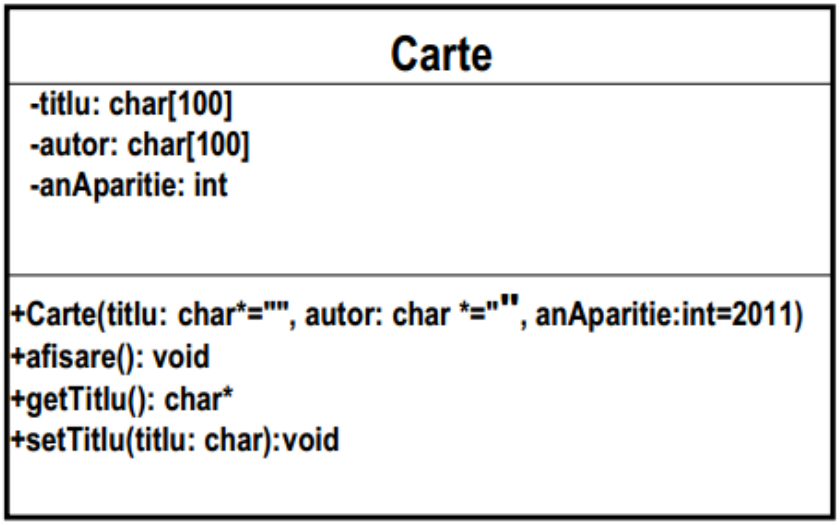
  ¤ idMetoda este identificatorul metodei

  ¤ idP1,..., idPn sunt parametri metodei

  ¤ tip1, ..., tipn sunt tipurile parametrilor

  ¤ tip_returant reprezintă tipul valorii returnate de metodă

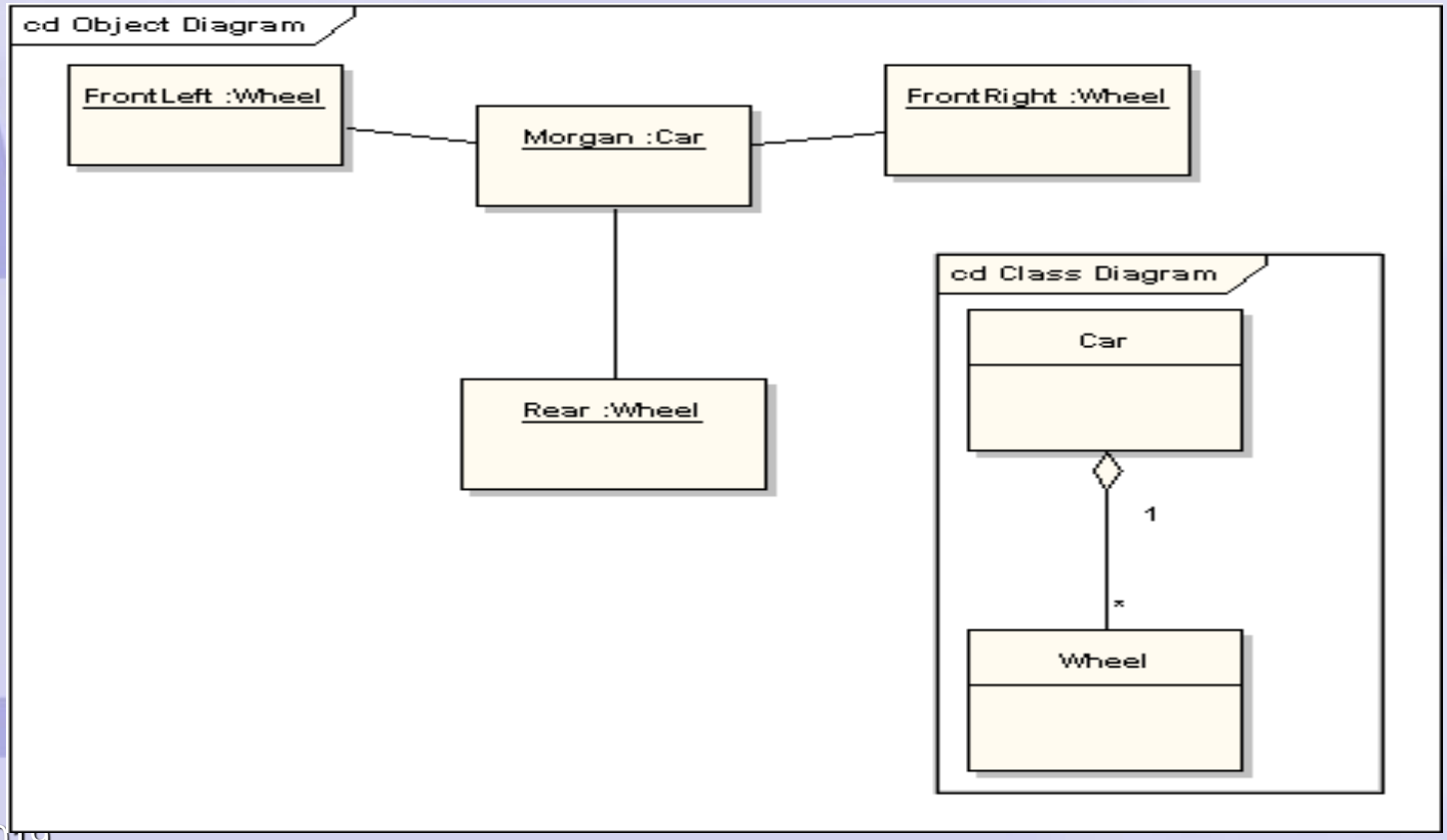
Diagrame de Clase. Exemplu




```
class Carte{
  private:
   char titlu[100];
   char autor[100];
   int anAparitie;
  public:
   Carte(char *titlu="", char *autor="", int an=2011);
   void afisare();
   char* getTitlu();
   void setTitlu(char *titlu);
};
```

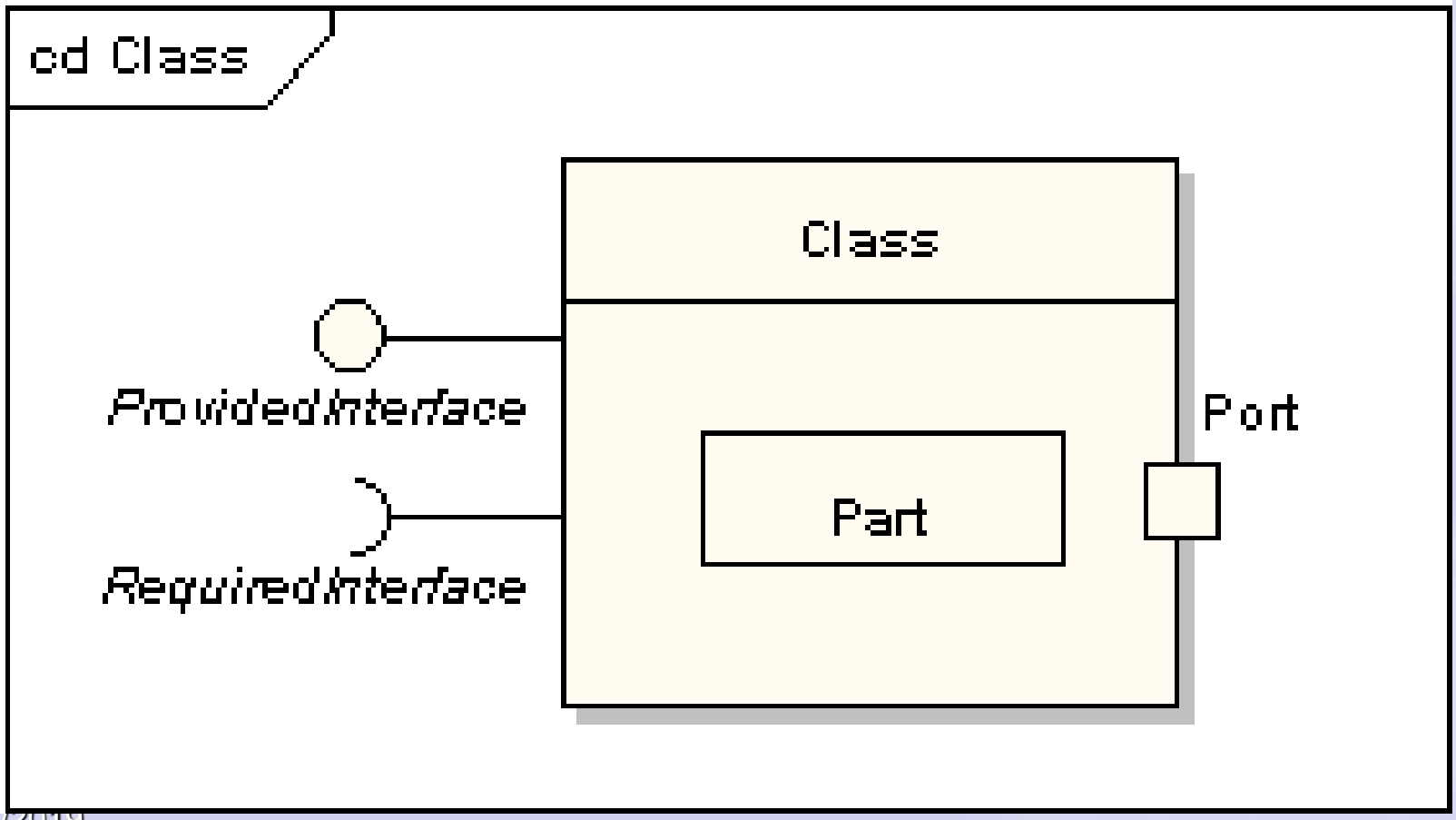
c. Diagrame pentru obiecte

- **Poate fi considerat un caz special de diagramă de clasă. Are scopul de a sublinia relaţiile între clase la un moment dat. Obiectul nu este reprezentat cu atribute şi este calificat cu numele clasei.**
- ***Object:class***
- **Reprezintă o instanţă la un moment dat.**



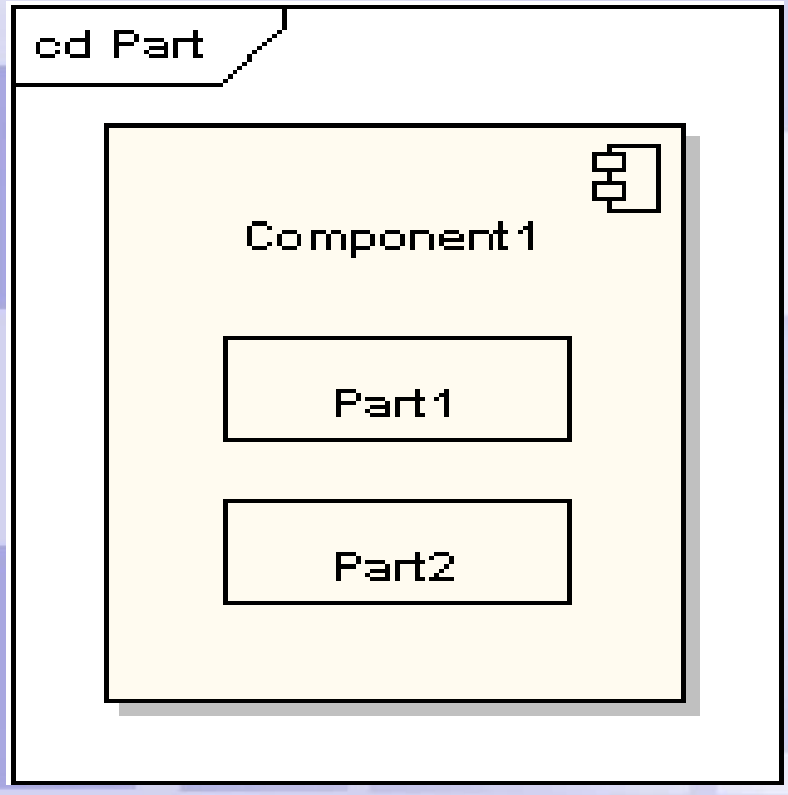
d. Diagrama pentru structuri compuse

- Reprezintă structura internă a unui clasificator, incluzând punctele sale de interacţiune cu alte părţi ale sistemului.
- Arată configuraţia şi relaţia dintre părţi care împreună formează comportametul clasificatorului care le conţine.



d. Diagrama pentru structuri compuse

- **Part- un element de una sau mai multe instanţe care sunt în proprietatea instanţei clasificatorului. De ex: un grafic este reprezentat de elemente grafice care pot fi reprezentate prin părţi. O parte este reprezentată grafic prin:**



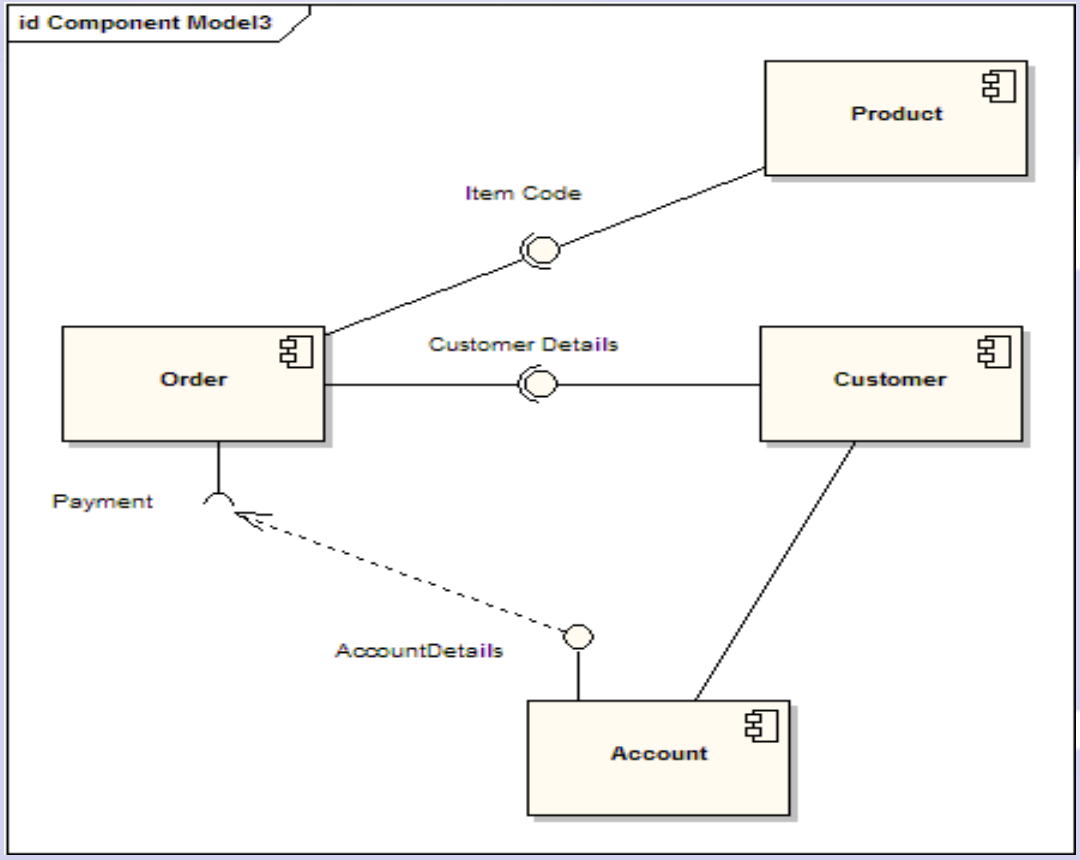
- **Part: este partea vizibilă a instanţei clasificatoare. Part_i definesc interacţiunea dintre clasificator şi mediu. Poate fi vazut şi ca un serviciu pe care clsificatorul îl poate furniza, dar şi ca serviciu pe care-l poate cere mediului.**
- **Interfaţa: similară cu clasa, dar nu are implementari ale operaţiilor şi toate atributele sunt constante.**

d. Diagrama pentru structuri compuse(3)

- ***Conectori:***
- ***Delegate*** – defineşte funcţionarea internă a unui port extern sau interfaţă. Leagă contactul extern (portul, interfaţa) cu realizarea lui internă prin comportamentul unei parţi componente.
- ***Colaborarea*** – set de roluri care sunt utilizate colectiv pentru a ilustra o funcţionalitate specifică.
- ***Role binding*** – este desenată de la o colaborare catre un clasificator care îndeplineşte rolul.
- ***Reprezentarea*** – este desenată de la o colaborare catre un clasificator şi arată că colaborarea este utilizată în clasificator.
- ***Occurrence*** - este desenată de la colaborare catre clasificator pentru a arăta ca colaborarea reprezintă clasificatorul.

e. Diagrame de compunere

- Ilustrează piese soft, embedded controllers, care de fapt impreuna reprezinta sistemul.
- Uzual, o componentă este implementată prin una sau mai multe clase sau obiecte în timpul execuţiei.
- Similare cu package diagrams, dar diagramele de compunere sunt mai bogate semantic.
- Toţi membrii sunt privaţi, iar la package toţi membrii sunt publici.

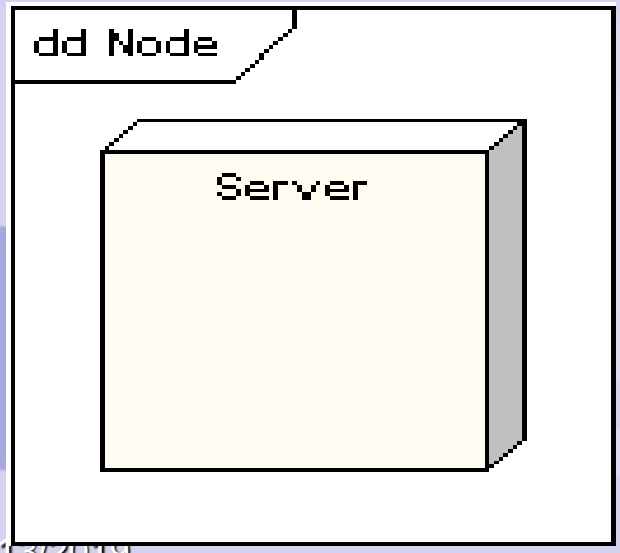


- *Conectori:*
- Assembly connector- leagă o interfaţă cerută de o componentă cu interfaţa expusă de celalată componentă.

f. Diagrame de deployment

- Descriu arhitectura de funcţionare a unui sistem, mai sunt numite si diagrame de punere in functiune.
- Arată configuraţia hard prin noduri şi modul în care elementele soft sunt mapate în aceste noduri.
- Nod - element hard sau soft;
- Instanţa unui nod- este reprezentată subliniat;
- Sterotipii de noduri: nume standard pentru anumite noduri des utilizate

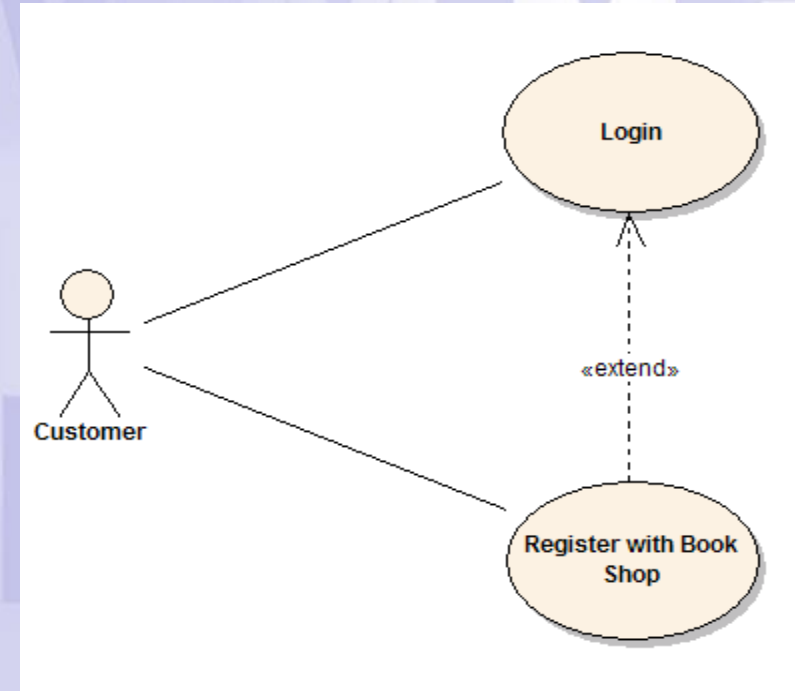
  <<Cd-Rom>>, <<PC Client>>, <<Storage device>>
- Artefacte: este un produs al procesului de dezvoltare soft. Poate include modele, rapoarte, manuale, fişiere sursă, etc



•Pot fi văzute prin relaţia de asociere sau containere.
•Asocierea reprezintă căile de comunicare dintre noduri.
•Alte noduri ne arată ca conţin elemente înglobate (soft în hard).

g. Use case diagram

- Acest model capturează cerinţele sistemului. De obicei de aici se porneste modelarea unei aplicatii.
- Use cases au menirea de a comunica cu utilizatorii şi alti „jucatori” care iau parte la ceea ce sitemul trebuie să facă.
- *Use cases*
- Este o singura unitate logica de executie. Este vederea de foarte sus a comportamentului unui sistem vazut de cineva din afara.



- *Definiţiile use cases includ:*
  - Numele si descrierea: cuprind cererile funcţionale formal definite pentru utilizatorul final. O cerere este văzută ca un contract sau promisiune pe care use case va executa o acţiune sau va da valoare sistemului.
  - Constângeri:
    - O constângere este o condiţie restrictivă sub care diagrama use-case acţionează. Acestea pot fi pre, post sau condiţii invariante.
  - Scenarii:
    - Este o descriere formală a fluxului de evenimente care au loc in timpul execuţiei unei instanţe use case. Defineşte o secvenţă specifică între sistem şi actorii externi.
  - Incluziunea use cases: acestea pot conţine funcţionalitatea altor use cases.
  - Use cases extinse (extended use cases): Acestea pot extinde comportamentul altora, sunt în general utilizate în circumstanţe excepţionale.
  - Puncte de extensie: un punct în care un use cases se extinde poate fi definit prin adaugarea unui punct de extensie.
  - Limitele sistemului: este utilizată de obicei când use cases sunt in sistem şi actorii sunt în afara sistemului.

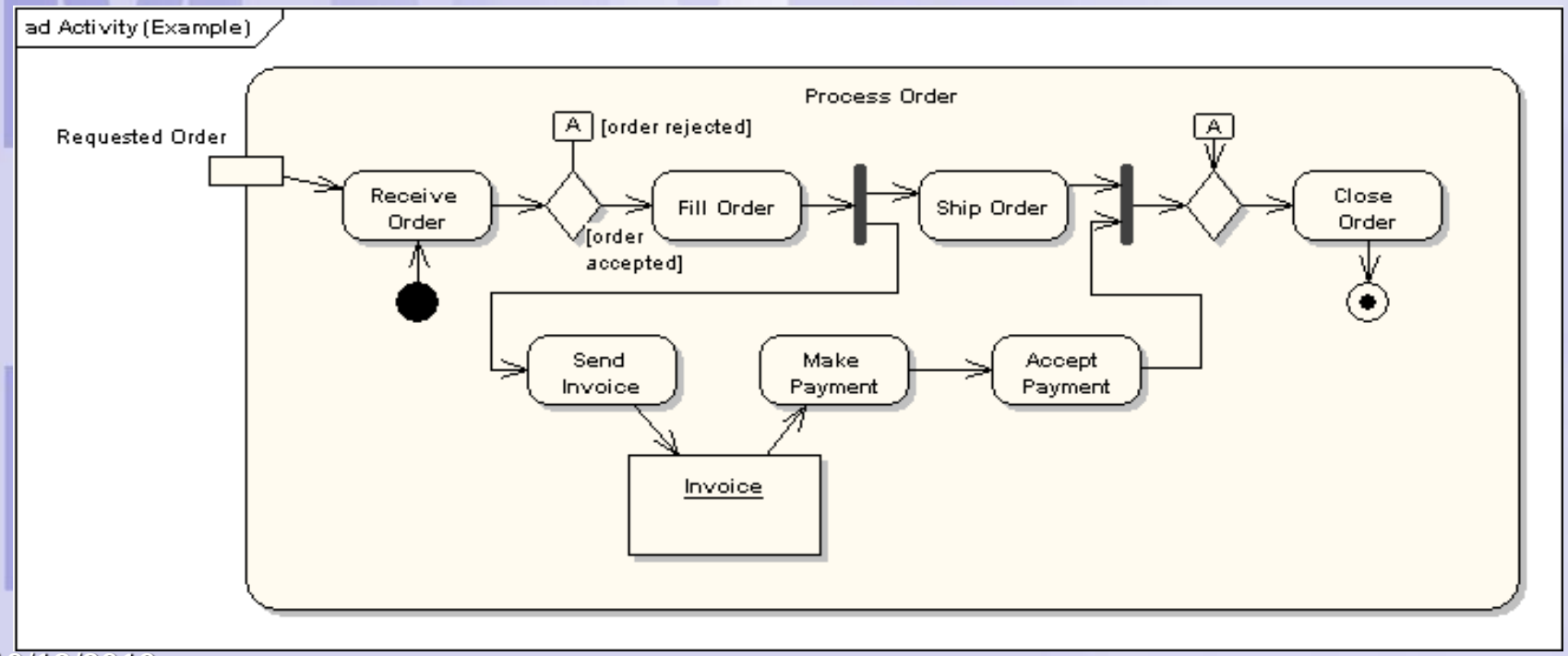
h. Diagrama activităţilor

- Diagrama activităţilor este utilizată pentru a afişa o secvenţă de activităţi.
- Arată fluxul de lucru de la punctul de start către punctul de finalizare, detaliind căile decizionale care se vor lua în funcţie de evenimentele conţinute în activităţi.
- Este foare mult utilizată în modelarea proceselor de business, în detalierea situaţiilor care pot apare în execuţia unor activităţi.

- *Elemente:*
  - Activităţi – specifică o secvenţă parametrizată a comportamentului;
  - Acţiuni – reprezintă un singur pas intr-o activitate;
  - Constrângeri ale acţiunilor – pre sau post condiţii;
  - Controlul fluxului – arată direcţia de control de la o acţiune la alta.
  - Nodul iniţial, nodul final, nodul de sfârşit de flux
  - Flux de obiecte – este o cale pe care o parcurg obiectele. Trebuie să aiba cel puţin un obiect la unul din capete
  - Noduri de decizie şi noduri de unificare- pot avea nume şi reprezintă luarea a doua decizii paralele şi reunificarea fluxului după efectuarea activităţilor în paralel.

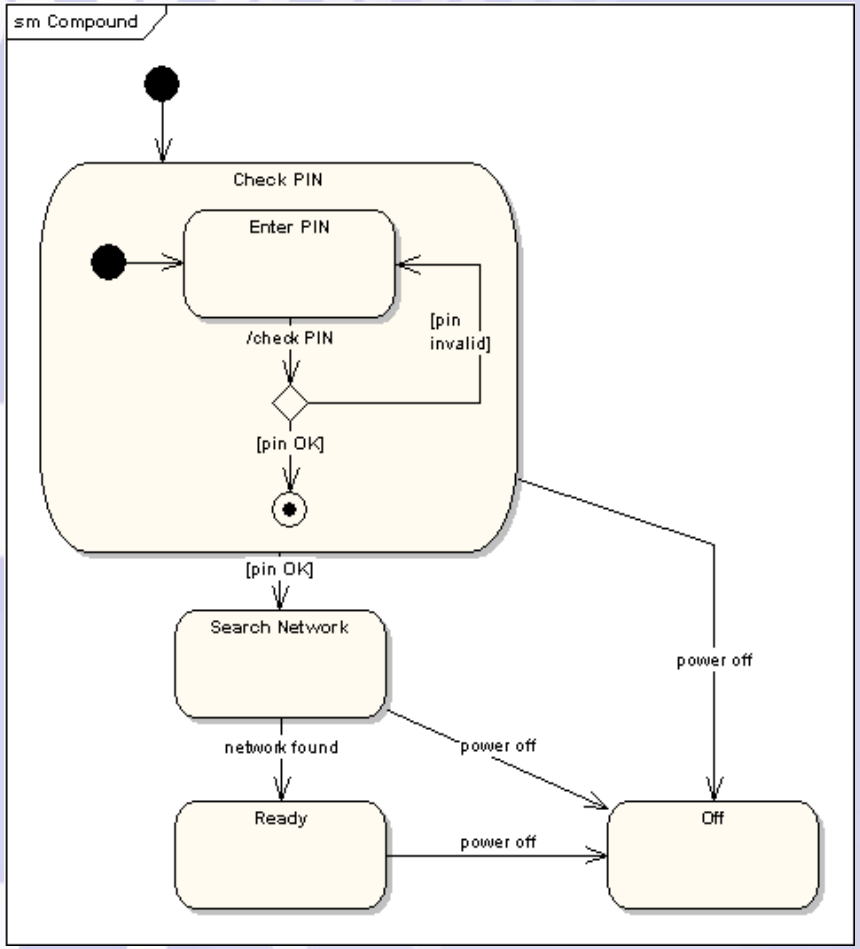
h. Diagrama activităţilor

- Noduri despăţitoare şi noduri de reunire – indică începutul şi sfârşitul unor fire de executie de control.
- Noduri de expansiune - arată activităţi repetitive. Sunt marcate prin cuvintele iterativ, parallel sau stream.
- Tratarea excepţiilor-
- Partiţii – sunt prezentate culoare de separare a activităţilor, in funcţie de anumiţi factori.



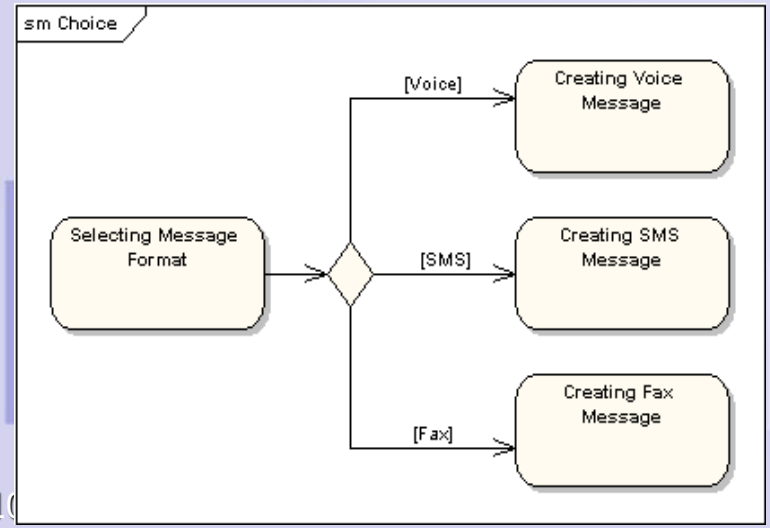
i. Diagrame de stare a maşinii

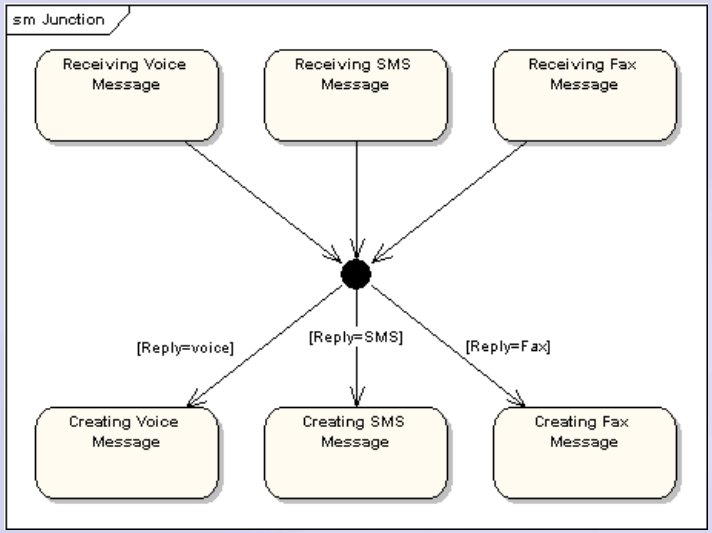
- Modelează comportamentul unui singur obiect, specificând secvenţa de evenimente prin care trece obiectul de-a lungul duratei de viaţă.
- Stările – pot fi stări simple, compuse (o diagramă conţine şi sub-diagrame), stările iniţiale şi finale
- Tranziţii - sunt treceri de la o stare la alta. O tranziţie poiate avea un trigger (declanşator), condiţie (guard) şi un efect. Trigger-ul declanşează tranziţia: un semnal, eveniment, sau o perioadă de timp.



i. Diagrame de stare a maşinii

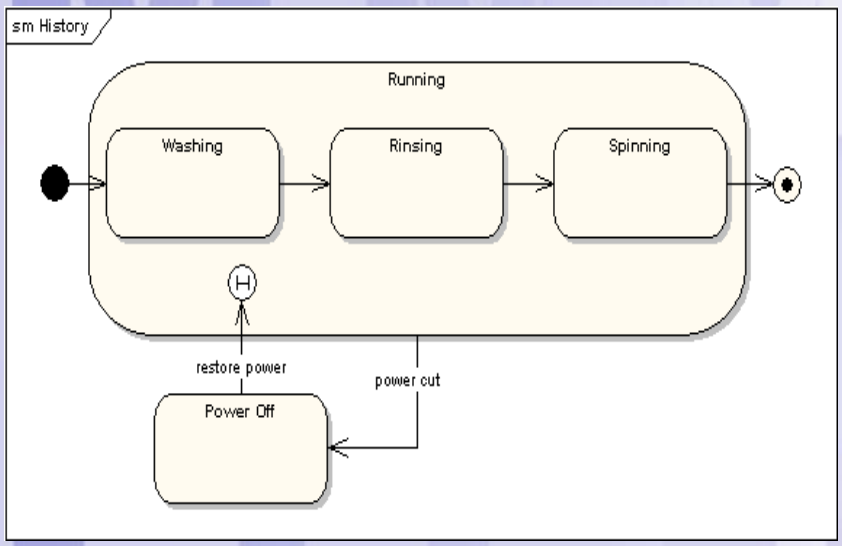
- Condiţia trebuie să fie adevărată pentru declaşarea tranziţiei, iar efectul este o acţiune invocată direct de obiect cand starea provocată de tranziţie a fost atinsă.
- Auto-tranziţie – tranziţie care se auto-returnează. De ex. după un anumit interval de timp se petrece ceva.
- Alegere pseudo-stări – atunci cand o tranziţie se desparte în mai multe tranziţii. După pseudo-stare, starea depinde de opţiunea aleasă.
- Joncţiune pseudo-stări- leagă mai multe tranziţii. Joncţiunile sunt fără semantică. O joncţiune este statică, spre deosebire de alegere pseudo-stare care îndeplineşte o condiţie dinamică.

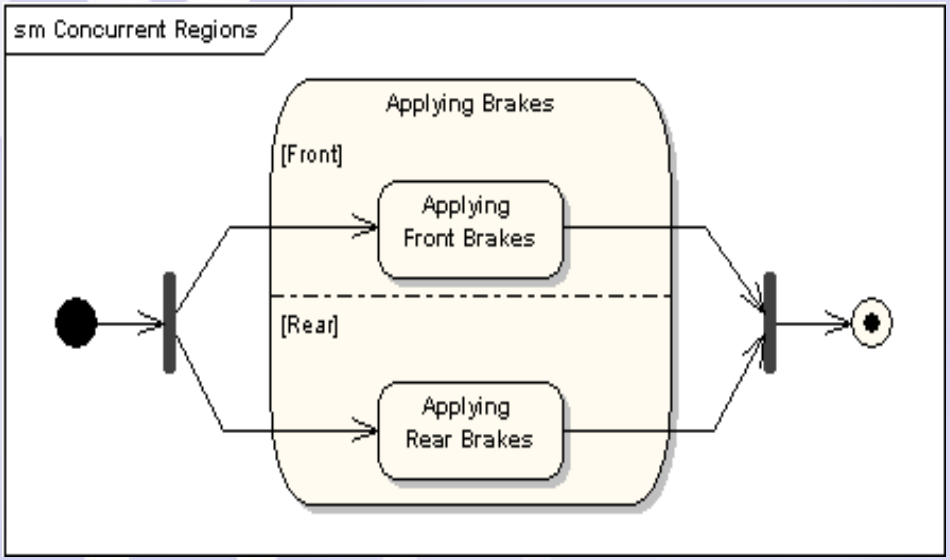




i. Diagrame de stare a maşinii

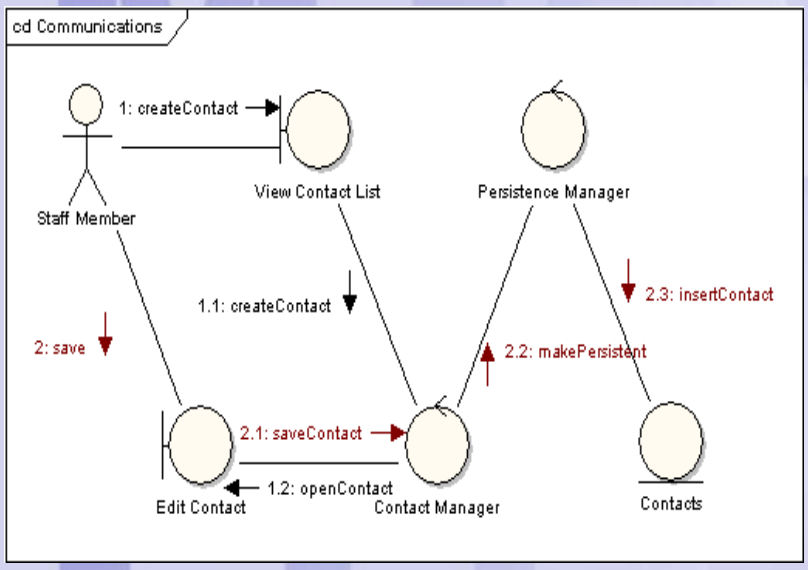
- History states- este utilizată cand de doreşte să se memoreze starea de dinainte a maşinii cand aceasta a fost întreruptă.
- Regiuni concurente- o stare poate fi descompusă în substări care există şi se execută concurent.

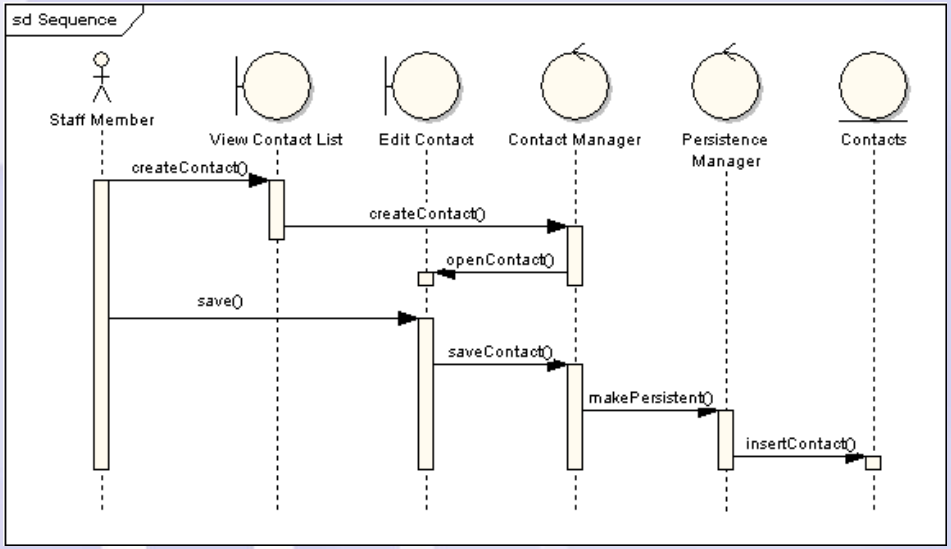




j. Diagrame de comunicare (diagrame de colaborare)

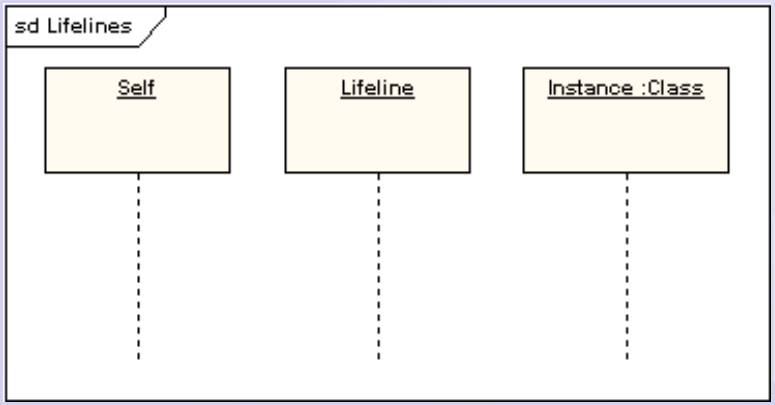
- Sunt diagrame de interacţiune care arată aceleaşi informaţii ca o diagramă de secvenţă, dar se axează pe relaţiile între obiecte.
- Mesajele dintre obiecte sunt numerotate. Următoarele două scheme reprezintă o aceeaşi informaţie:

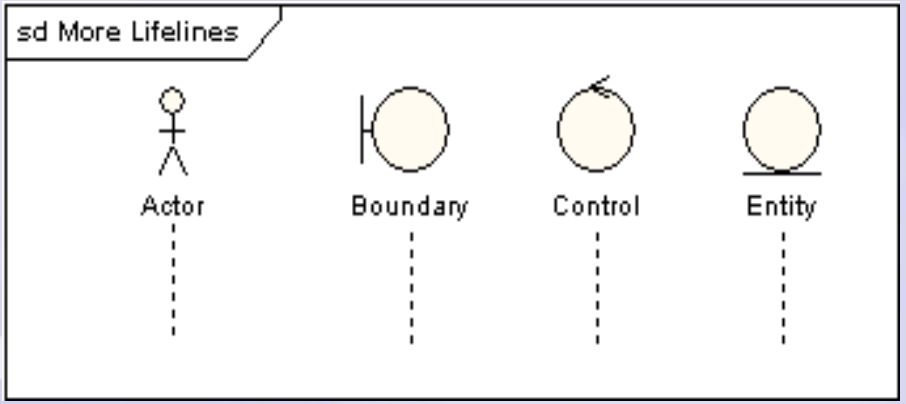




k. Diagrame de secvenţă

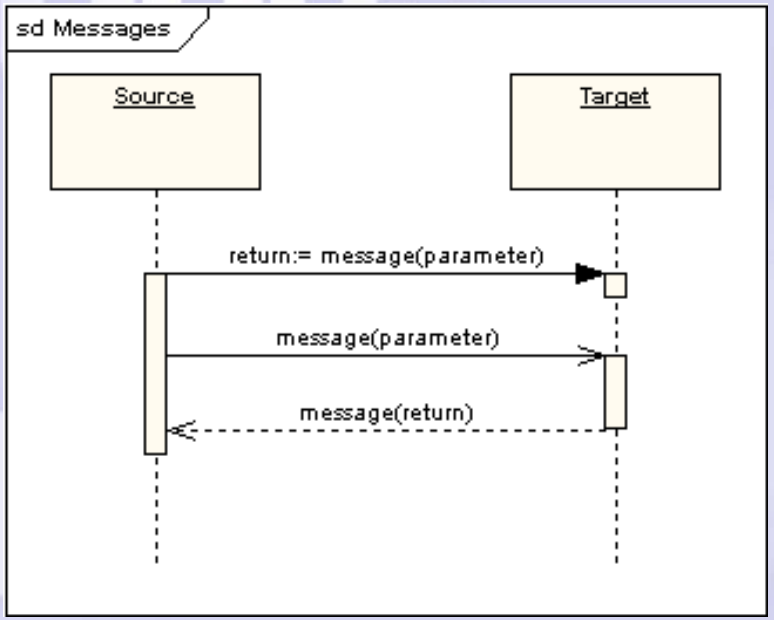
- Sunt o formă de interacţiune care arată obiectele pe durata lor de viaţă, cu interacţiunile lor în timp reprezentate ca mesaje desenate ca săgeţi de la sursă la destinaţie.
- Este o reprezentare foarte intuitivă a modului în care obiectele comunică unul cu altul.
- Lifeline - este un participant într-o diagramă de secvenţă. Numele “self” ne arată ca obiectul este un clasor care conţine diagrama de secvenţă.
- O digramă de secvenţă poate fi în proprietatea unui use-case.
- Elemente: limite, control şi entităţi.





k. Diagrame de secvenţă

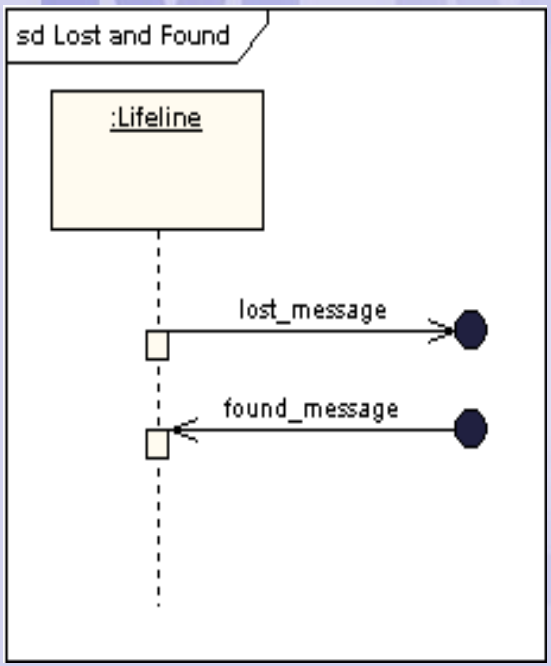
- **Mesajele:**
  - **Complete**
  - **Pierdute şi găsite**
  - **Sincrone sau asincrone**
  - **Apeluri sau semnale**

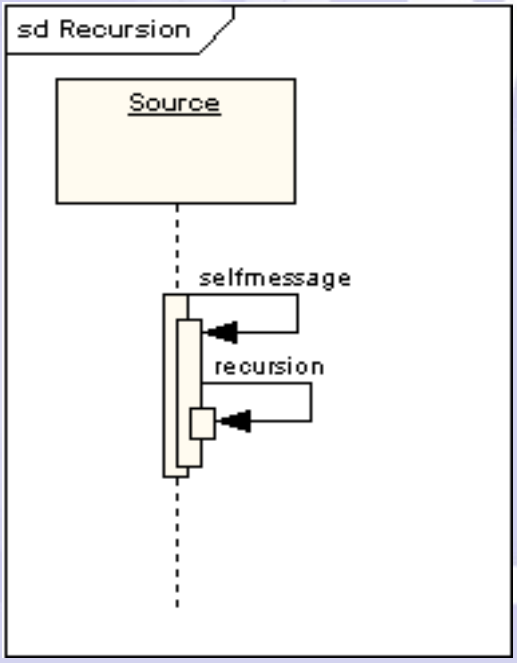


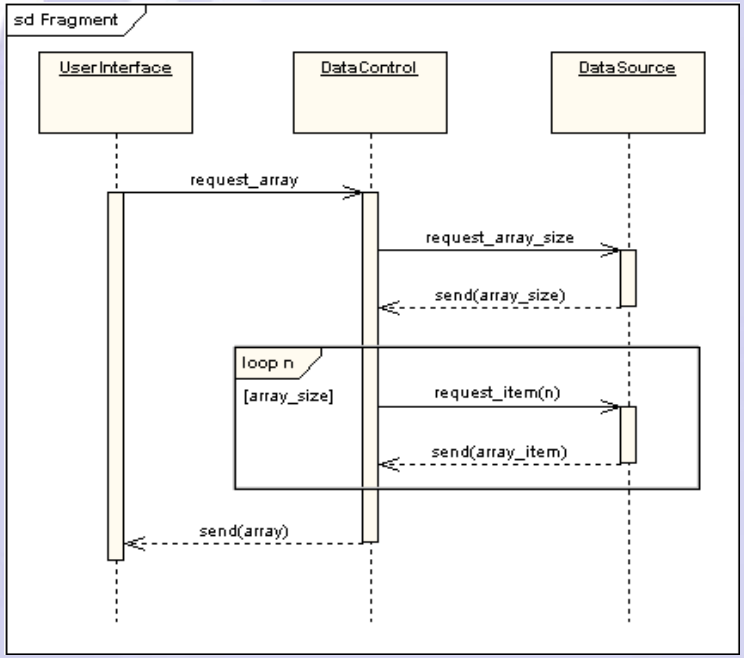
- **Ocurenţa: timpul de executie sau de activare a unui control.**
- **În exemplul de mai sus avem un obiect sursă care trimite două mesaje şi primeşte două replici; al doilea este destinaţia care recepţionează un mesaj sincron şi returnează o replică, al treilea este obiectul destinaţie care recepţionează un mesaj asincron şi returnează o replică.**

k. Diagrame de secvenţă

- Automesajul – este un apel recursiv al unei operaţii sau a unei metode apeland o altă metodă aparţinând aceluiaşi obiect.
- Mesaje pierdute şi găsite
- Pierdute- sunt mesaje care fie că pleacă dar nu ajung la destinaţie, sau care merg catre o destinaţie care nu este arătată in diagrama curentă.
- Găsite- care vin de la un expeditor necunoscut sau care nu se arată în diagrama curentă.





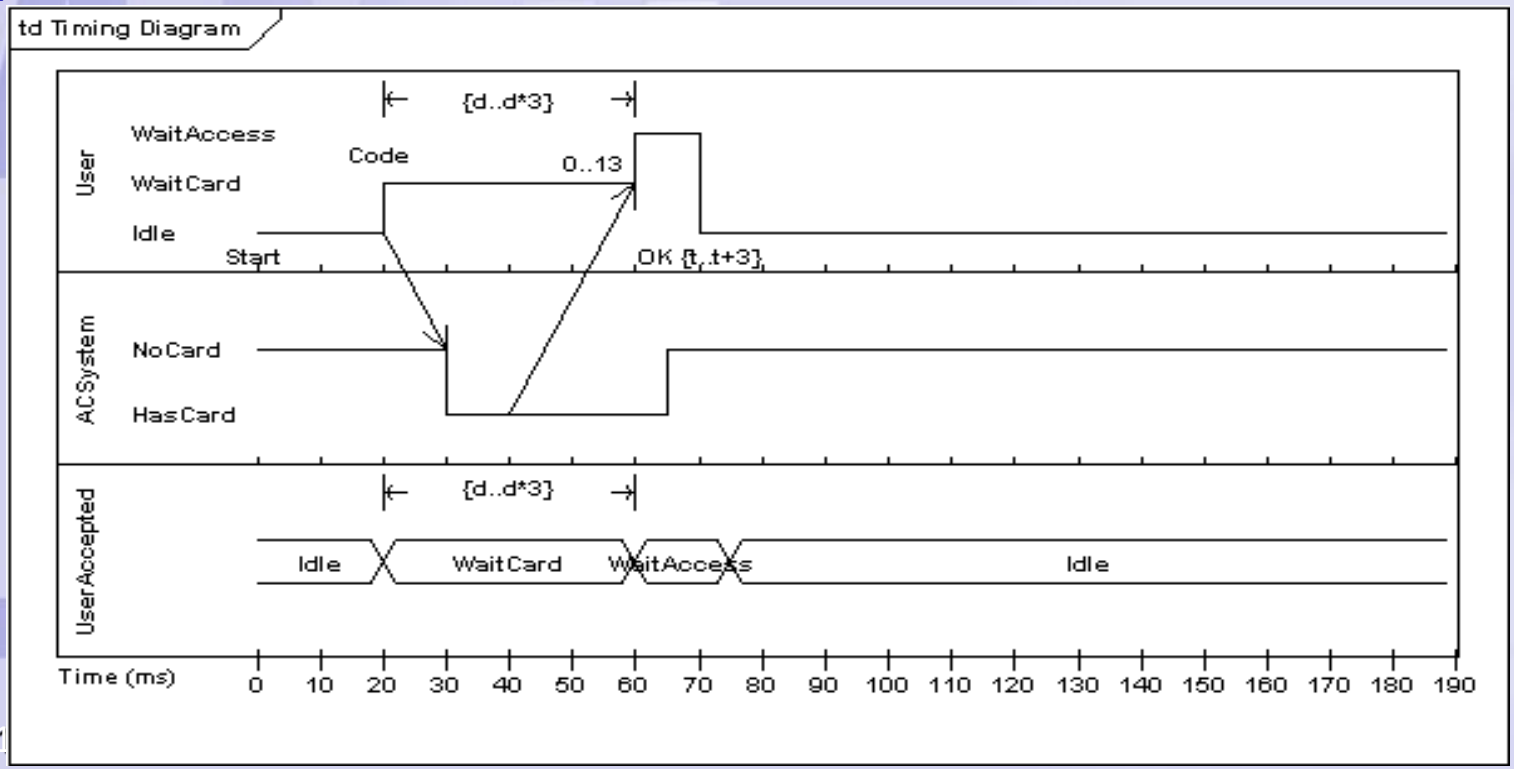


k. Diagrame de secvenţă

- Combinarea fragmentelor: Aceste diagrame nu sunt facute sa arate complexitatea algoritmilor. Dar cateodată, acest lucru este necesar ţi se face prin adăugarea de combinaţii de fragmente. Un fragment combinat este una sau mai multe secvenţe de execuţie închise în cadre şi care se execută sub anumite circumstanţe specificate prin nume. Acestea pot fi:
- Notaţii:
- “alt” pentru fragmende alternative if..then…else
- “opt” pentru modelul switch
- “par” pentru secvenţe paralele la procese concurente
- “strict” pentru o serie de mesaje care trebuie procesate într-o ordine dată
- “neg” pentru mesaje invalide
- “assert” care arată ca orice secvenţă care nu arată ca un operand al aserţiunii este considerat invalid
- Mesaje repetitive:
- Porţi- sunt puncte de conexiune între mesajul din fragment cu mesajul din afara fragmentului

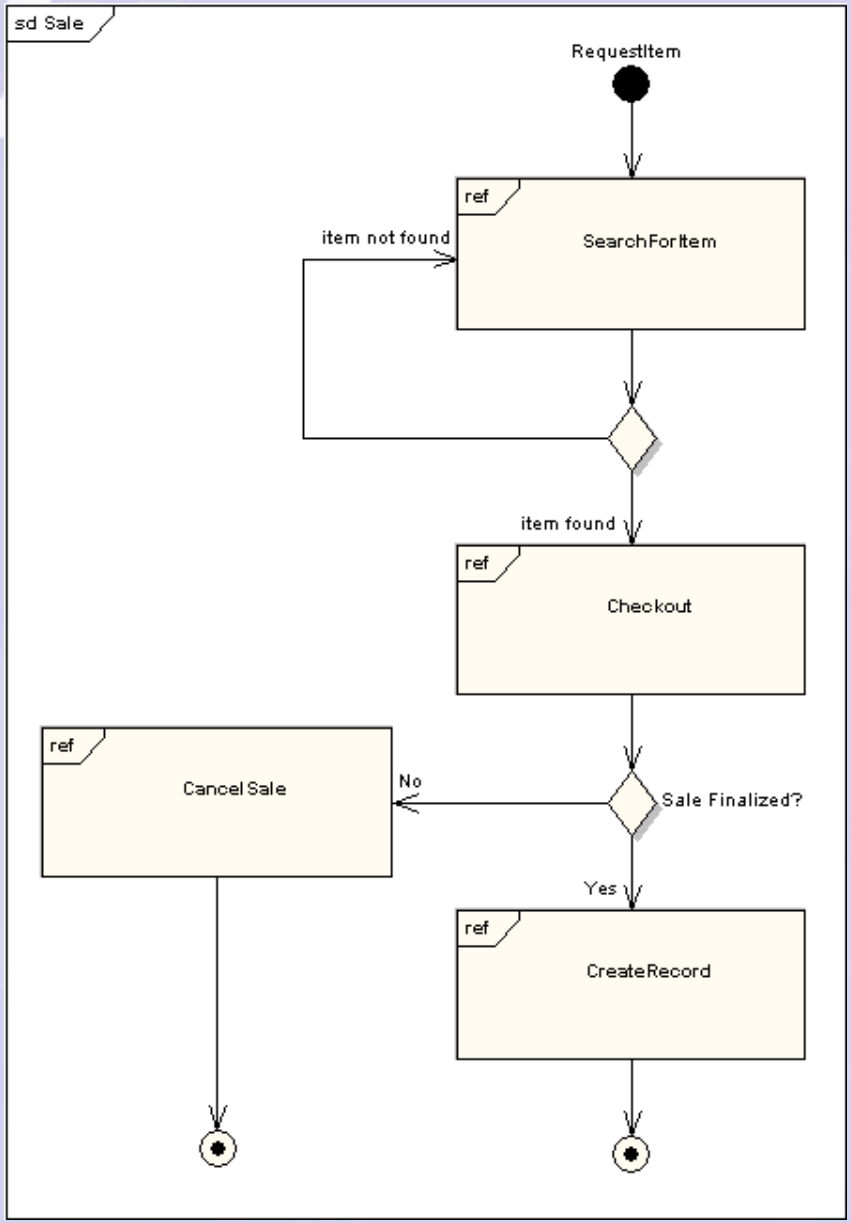
I. Diagrame de timing

- Sunt utilizate pentru a arăta schimbările in stare sau valoare a unul sau mai multe elemente în timp. Pot arăta şi interacţiunea dintre evenimentele dependente de timp şi constrângerile care le guvernează.
- Linia de stare - arată schimbările în starea obiectului într-o perioadă de timp;
- Linia de valoare - schimbările în valorile unui element într-o perioadă de timp;



m. Diagrama generală de interacţiune

- Este o formă de diagramă de activităţi în care nodurile sunt reprezentate de diagrame de interacţiune.
- Diagrame de interacţiune- pot include:
  - secvenţe,
  - comunicări,
  - interacţiuni şi
  - diagrame de timing.



6. Exemple de utilizare

Problema de rezolvat:

- **În cadrul proiectului KeyToNature (www.Key2Nature.eu) care este un proiect de eContent *plus*, se urmăreşte înglobarea şi reutilizarea resurselor digitale, a uneltelor de identificare, a celor de eLearning in scopul cunoaşterii, a educaţiei în domeniul biodiversităţii.**

- **Una din problemele spinoase tehnice de rezolvat este:**
  - **cum reprezentăm resursele digitale;**
  - **cum utilizăm unelte gata făcute în scopul de a identifica o entitate biologică sau a învăţa;**
  - **cum putem stoca impreună atât uneltele cat şi resursele;**
  - **cum putem utiliza cele mai înalte tehnologii pentru a ne atinge scopurile.**
- **S-a utilizat in momentul realizarii pentru construirea diagramelor doua unelte UML:**
  - **Enterprise Architect 7.1. de la Sparx Systems – versiune trial;**
  - **Plug-ins pentru IntelliJIdea 7.0.3.**

Diagrama use cases

- S-a reprezentat în următoarea diagramă de use-case cazul unui student-elev la biologie care are de făcut un referat.

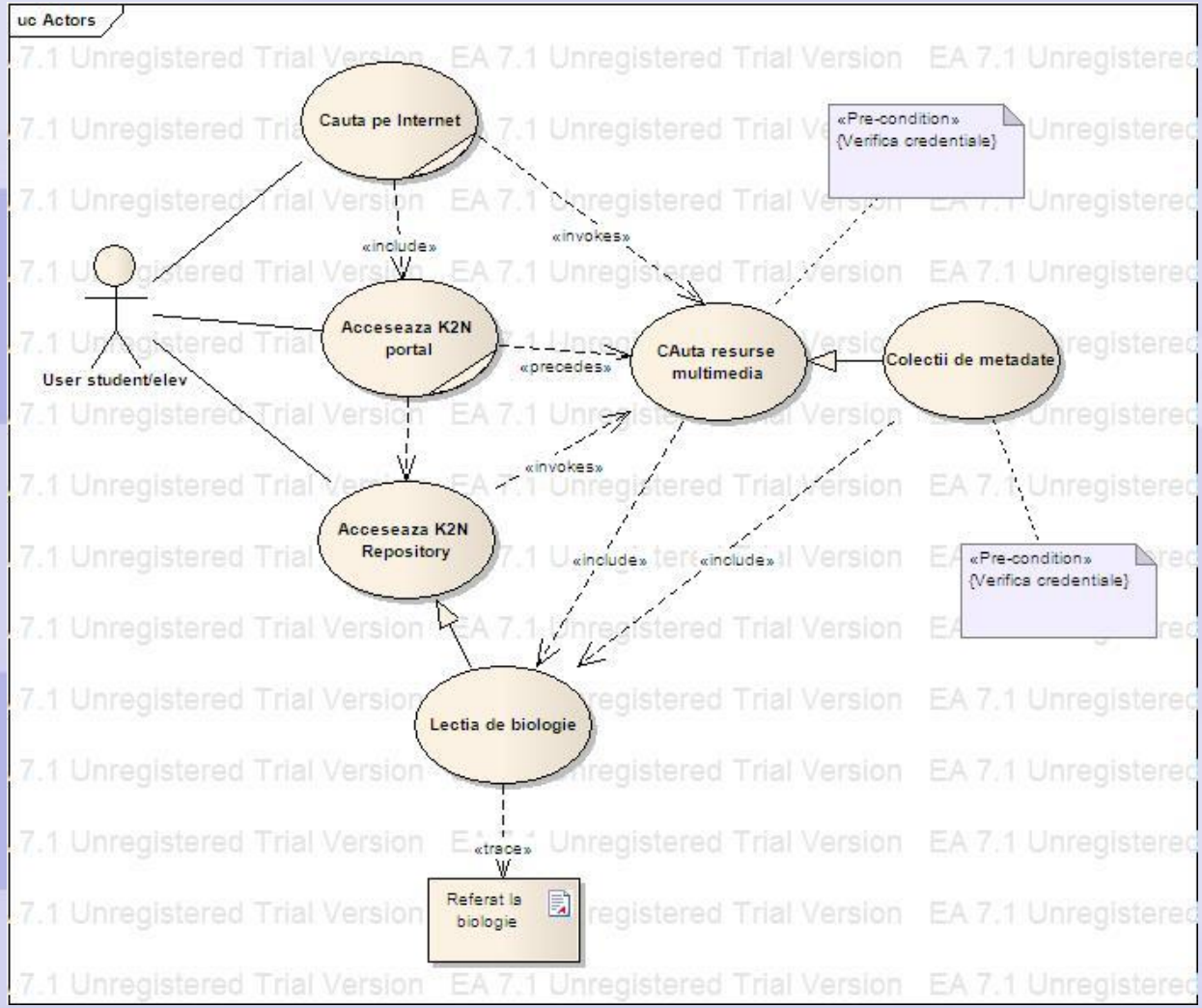


Diagrama de activitati

- Parte din aceast use-cases diagram – căutarea în repository şi care face parte dintr-o diagramă de activităţi poate fi reprezentată astfel:

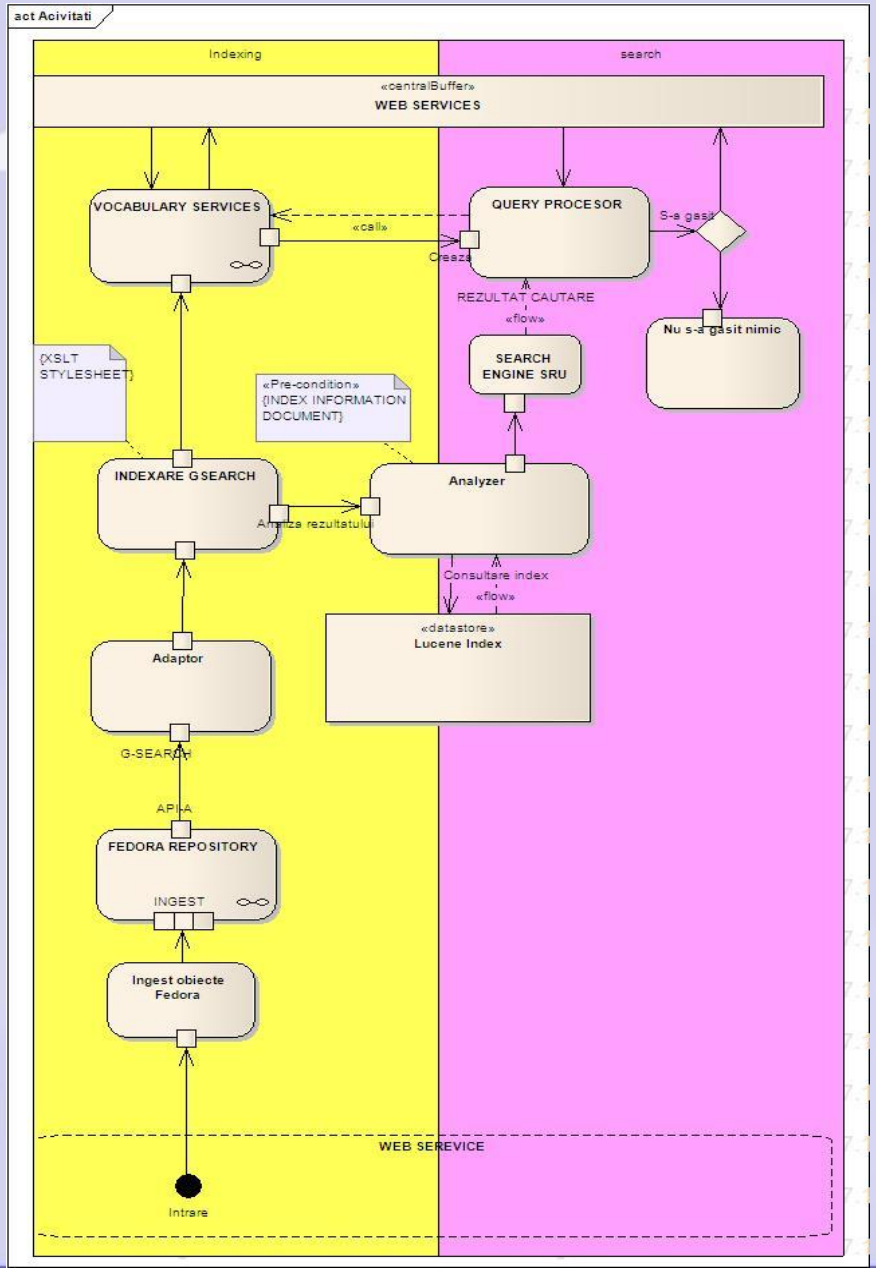
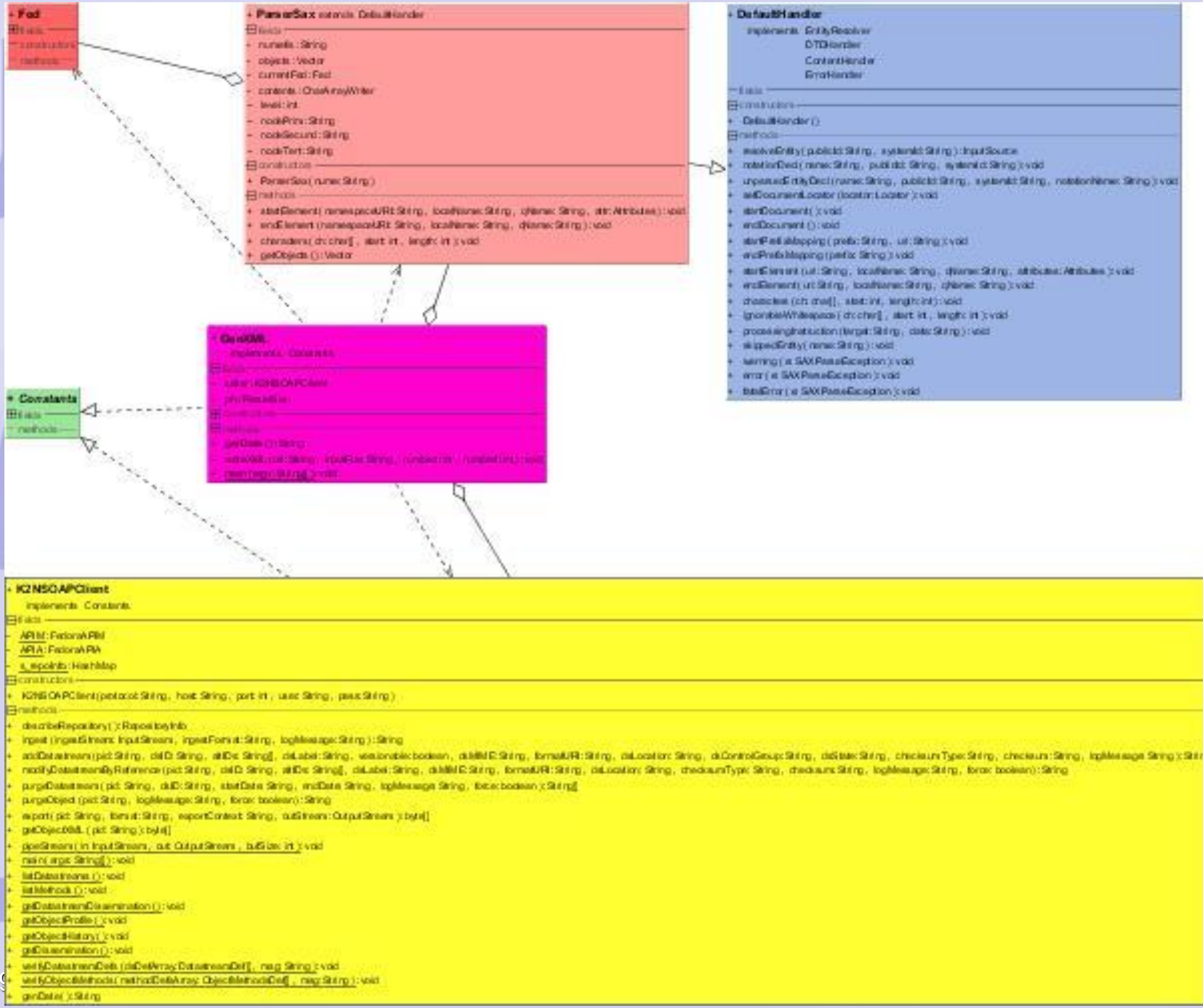


Diagrama claselor

- Partea care reprezintă sarcina impusa în acest moment se poate descrie în cuvinte astfel:
- Se va programa un soap client care să realizeze ingest-ul în repository din surse de date eterogene:
  - fişiere .xml,
  - baze de date Access,
  - formate standard de metadate RDF,
  - alte surse.
- Ingest-ul metadatelor din fişiere .xml format general:
- Diagrama claselor este:

U
M
L



Fed
ParserSax extends DefaultHandler
numefis : String
objects : Vector
currentFed : Fed
contents : CharArrayWriter
level : int
nodePrinc : String
nodeSecund : String
nodeTert : String
ParserSax(numer: String)
startElement(namespaceURI: String, localName: String, qName: String, attr: Attributes) : void
endElement (namespaceURI: String, localName: String, qName: String) : void
characters(ch: char[] , start: int , length: int) : void
getObjects () : Vector
DefaultHandler
implements EntityResolver
DTDHandler
ContentHandler
ErrorHandler
DefaultHandler ()
resolveEntity (publicId: String , systemId: String) : InputSource
notationDecl (name: String , publicId: String , systemId: String) : void
unparsedEntityDecl (name: String , publicId: String , systemId: String , notationName: String) : void
setDocumentLocator (locator: Locator) : void
startDocument ( ) : void
endDocument () : void
startPrefixMapping (prefix: String , uri: String) : void
endPrefixMapping (prefix: String) : void
startElement (uri: String , localName: String , qName: String , attributes: Attributes) : void
endElement (uri: String , localName: String , qName: String) : void
characters (ch: char[] , start: int , length: int) : void
ignorableWhitespace (ch: char[] , start: int , length: int) : void
processingInstruction (target: String , data: String) : void
skippedEntity (name: String) : void
warning (e: SAXParseException) : void
error (e: SAXParseException) : void
fatalError (e: SAXParseException) : void
GenXML
implements Constants
Constants
K2NSOAPClient
implements Constants
APIM : FedoraAPIM
APIA : FedoraAPIA
s_repoInfo : HashMap
K2NSOAPClient(protocol: String , host: String , port: int , user: String , pass: String)
describeRepository( ) : RepositoryInfo
ingest (ingestStream: InputStream , ingestFormat: String , logMessage: String) : String
purgeObject (pid: String , logMessage: String , force: boolean) : String
getObjectXML (pid: String) : byte[]
main(args: String[]) : void
listDatastreams () : void
listMethods () : void
getDatastreamDissemination () : void
getObjectProfile ( ) : void
getObjectHistory ( ) : void
getDissemination () : void
genDate( ) : String

10/13/201

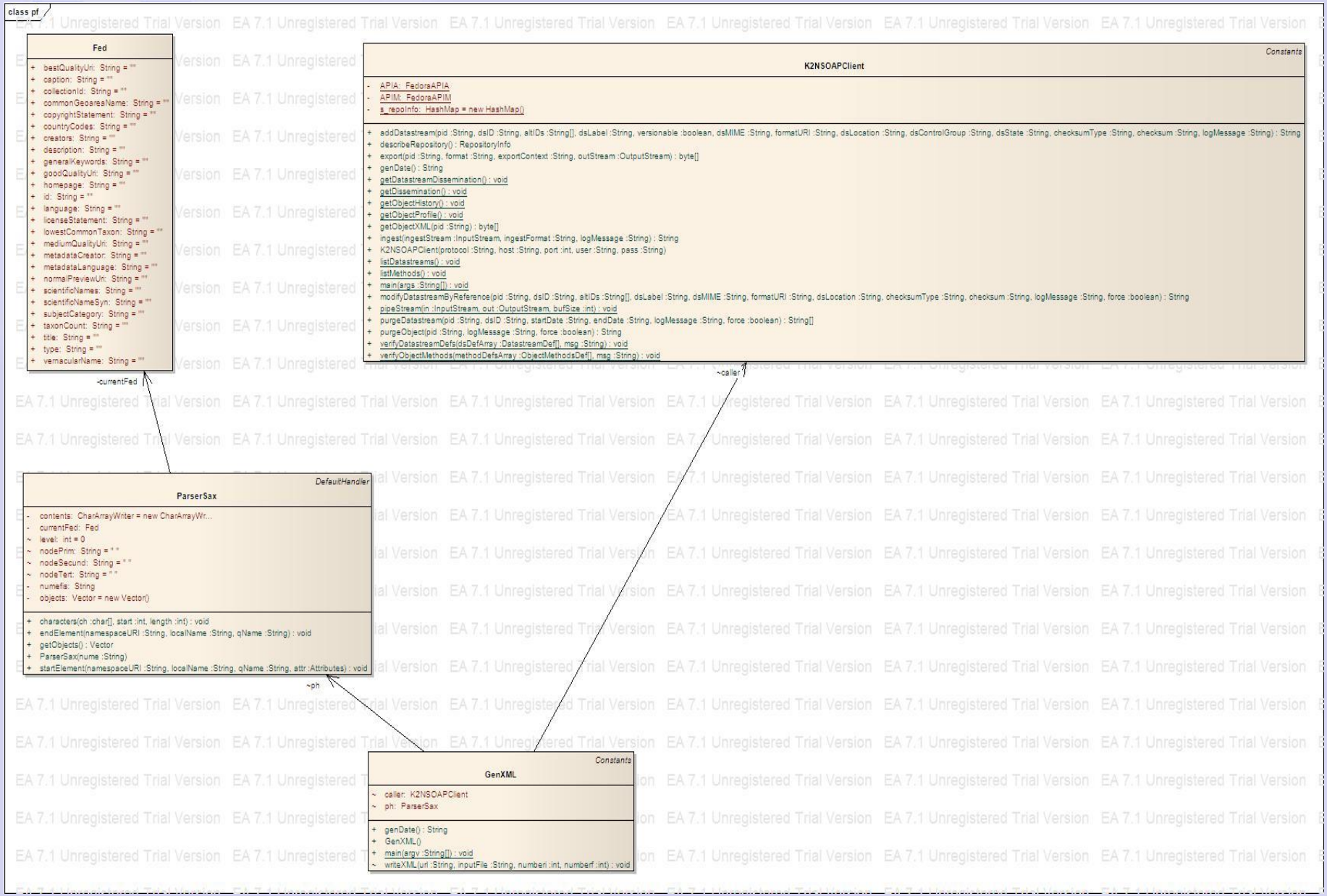
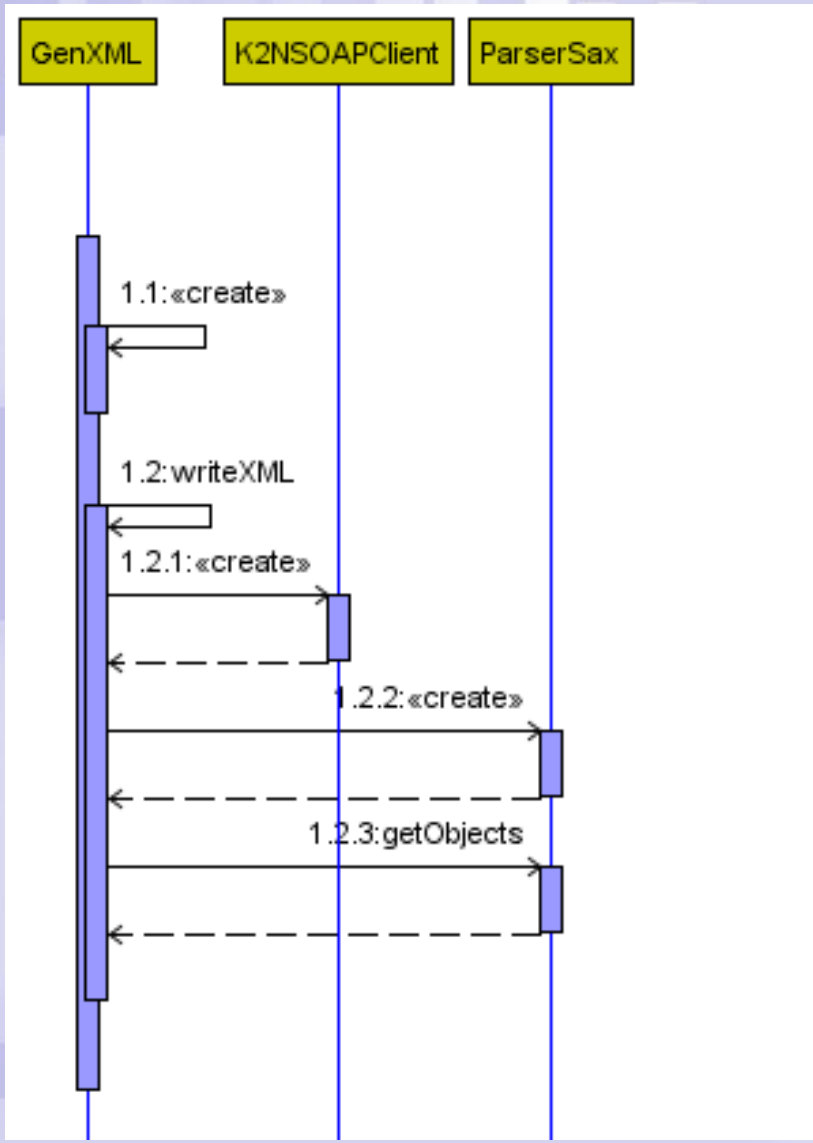


Diagrama de secventa



7. Referinte:

- Tutorial UML 2.x. de la Sparx Systems
http://www.sparxsystems.com.au/uml-tutorial.html
- Articole despre UML în:
http://www.techit.ro/tutorial_uml.php
- Learn UML de la Visual Paradigm
http://www.visual-paradigm.com/product/vpuml/demos/
- Wikipedia
http://en.wikipedia.org/wiki/UML_2
- Key2Nature portal
www.Key2Nature.eu
- Lucene full text search engine
http://lucene.apache.org/
- Fedora website
http://fedora-commons.org/