Universitatea Politehnica Bucuresti
Facultatea de Electronica, Telecomunicatii si Tehnologia Informatiei

Inginerie software bazata pe componente

Manole Laurentiu 442A
Mardare Oana - Viorica 441A
Hurmuzache Ciprian-Constantin 441A
Sarbu Lucia-Ioana 441A
Moraru Diana-Teodora 441A

Cuprins:

1. INTRODUCERE

O componenta este un obiect software, creat pentru a interactiona cu alte componente si care sa incapsuleze o anumita functionalitate sau un set de functionalitati. O componeta are o interfata bine definita si este in conformitate cu toate componentele care se afla in interiorul aceleiasi arhitecturi.[1]

Scopul prinicipal al ingineriei software bazate pe componente este de a creste productivitatea, calitatea, si timpul de livrare in cadrul dezvoltarii software cu ajutorul atat a standardizarii componentelor cat si a automatizarii productiei. O paradigma importanta utilizata aici este de aceea de a construi sisteme software din componente standard in locul „inventarii rotii” de fiecare data. Acest lucru necesita gandirea in termeni de familii de sisteme in detrimentul sistemelor singulare. O alta paradigma este aceea de a inlocui cautarea manuala, adaptarea si asamblarea componentelor cu a unora automatizate la cerere. Ingineria software bazata pe componente cauta sa integreze abordarile domeniului ingineresc cu abordarile bazate pe componente si cu abordarile generale[2].

O problema importanta a ingineriei software bazata pe componente este gasirea notatiilor potrivite pentru descrierea sistemelor. O buna notatie face ca documentarea proiectarii bazata pe componente sa fie clara, cu proprietati foarte bine definite si permite o automatizare a analizei sistemului.

O notatie buna in descrierea sistemelor bazate pe componente este aceea folosind obiectele. Fiecare componenta poate fi reprezentata de o clasa, interfata componentei poate fi reprezentata de interfata clasei si interactiunile dintre componente pot fi definite in termini de asociere.

Modelarea cu obiecte a sistemelor bazate pe componete are numeroase caracteristici bune. Notatiile obiectelor sunt familiare majoritatii inginerilor software. Acestea ofera o mapare directa a implementarilor si sunt suportate de programe comerciale.

Dar notatiile prin intermediul obiecteleor au si un cateva inconveniente. In primul rand acestea ofera o singura forma de implementare primitiva – metoda invocarii. Acest lucru face dificila reprezentarea unui numar mare de interactii intre componente. In al doilea rand acestea au un suport mic in descrierea ierarhica,

facand dificila descrierea sistemelor o data cu cresterea nivelului de detaliu. In al treilea rand, nu au suport pentru definirea familiilor de sisteme. In timp ce acestea pot fi utilizate pentru descrierea modelelor si definirea unui vocabular cu tipuri de obiecte, nu au suport sintactic explicit pentru caracterizarea unei clase de sistem in sensul definirii unor constrangeri pe care fiecare membru al familiei trebuie sa-l observe. In al patrulea rand, nu ofera suport direct pentru caracterizarea si analiza proprietatilor nonfunctionale. Acest lucru face greu de calculat proprietati critice de proiectare a sistemului, cum ar fi performanta si siguranta.

O alternativa care trece de aceste probleme este aceea a utilizarii unui limbaj de descriere arhitecturala (ADL). De-a lungul timpului au fost facute numeroase limbaje de descriere a nivelului inalt al sistemelor software complexe, care reprezinta structura globala ca o colectie de componente care interactioneaza si care permit inginerilor software sa gandeasca proprietatile sistemelor la un nivel mare de abstractizare. Proprietatile uzuale pe care se pune baza sunt cele legate de protocoalele de interactiune, largimea de banda si latenta, locatiile unde sunt centralizate datele, conformitatea cu standardele arhitecturale si anticiparea dimeniunii evolutiei.

Fiecare ADL se bazeaza pe diferite aspecte ale arhitecturii, cu trecerea timpului in interiorul comunitatii de cercetare in domeniul arhitecturilor software au aparut fundamente generale privind reprezentarea arhitecutrala. Unul dintre aspecte a fost dezvoltarea unui limbaj de descriere arhitecturala de generatia a doua, numit ACME, care poate fi folosit ca o reprezentare comuna a arhitecturilor software si care permite integrarea diverselor colectii de unelte de analiza arhitecturala independente.

2. ARHITECTURI

2.1 Descrierea Arhitecturala

Arhitectura software a unui sistem ii defineste structura acestuia la un nivel inalt, prezentatnd organizarea generala ca o colectie de componente care interactioneaza. Proiectarea arhitecturala a jucat mereu un rol important in determinarea succesului unui sistem software bazat pe componente complex:

alegerea unei arhitecturi apropiate poate face ca produsul sa satisfaca cerintele si sa fie foarte usor de modificat o data cu aparitia unor noi cerinte, in timp ce alegerea gresita a arhitecturii poate duce la rezultate dezastroase.

In ciuda importantei acesteia pentru inginerii software de sistem, in cele mai multe cazuri proiectarea arhitecturala este facuta ad-hoc si informal. Ca rezultat modelele arhitecturale sunt de cele mai multe ori foarte putin intelese de catre dezvoltatori, alegerile arhitecturale fiind bazate mai mult pe instinct decat pe principii ingineresti solide iar astfel de modele arhitecturale nu pot fi analizate in privinta consistentei.

Ca un raspuns la aceste probleme un numar de cercetatori din domeniul industrial si academic au propus folosirea unor notatii formale pentru reprezentarea si analiza modelelor arhitecturale. Cunoscute in mod generic ca „Limbaje de descriere arhitecturala” (ADL), aceste notatii ofera in mod uzual atat schite conceptuale cat si o sintaxa concreta pentru caracterizarea arhitecturilor software. Acestea ofera de asemenea unelte de analiza, compilare si simulare a descrierilor arhitecturale scrise in limbajele asociate.

Exemple de ADL-uri: Aesop, Adage, C2, Darwin, Rapide, SADL, UniCon, Meta-H si Wright. In timp ce aceste limbaje sunt concentrate pe proiectarea arhitecturala, fiecare ofera cateva capabilitati distinctive: Adge suporta descrierea arhitecturala a structurilor folosite in navigatia si ghidarea din domeniul aviatiei; Aesop suporta folosirea mai multor stiluri arhitecturale; C2 suporta descrierea sistemelor cu interfata pentru utilizator cu ajutorul stilului bazat pe evenimente; Darwin suporta analiza sistemelor bazate pe distribuirea de mesaje; Meta-H ofera suport pentru proiectantii de sisteme software in timp real pentru aviatie; Rapide permite simularea modelelor arhitecturale si are si instrumente de analiza pentru rezultatele simularilor; UniCon are un compilator de nivel inalt pentru modelele arhitecturale care suporta combinatii intre componente heterogene si tipuri de conectori; Wright face analiza interactiunilor dintre componentele folosite in cadrul arhitecturii.

2.2 Elemente constitutive

Desi exista o diversitate considerabila intre capabilitatile diferitelor ADL-uri toate impart aceeasi baza conceptuala care determina anumite preocupari fundamentale pentru descrierea arhitecturala. Principalele elemente constitutive ale acestei baze sunt[1]:

- *Componentele* reprezentate de elementele computationale primare si de magaziile de date ale sistemului. Exemple de componente tipice sunt clientii, serverele, filtrele, obiectele si bazele de date. In majoritatea ADL-urilor componentele pot avea mai multe interfete, fiecare interfata definind un punct de interactiune dintre componenta si mediul acesteia.

- *Conectorii* reprezentati de interactiunile dintre componente. Din punct de vedere computational, conectorii mediaza comunicarea si coordonarea activitatilor dintre componente. Exemple de astefel de conectori ar fi simplele forme de interactiune pentru apeluri de procedura sau evenimente de broadcast. Dar conectorii pot deasemenea reprezenta interactiuni mult mai complexe cum ar fi protocoale client-server sau o legatura SQL dintre o baza de date si o aplicatie. Conectorii au deasemenea interfete care definesc rolurile jucate de diferiti participanti la o interactiune.

- *Sistemele* reprezinta configuratii ale componentelor si conectorilor. In ADL-urile moderne o proprietate cheie in descrierea sistemelor este aceea ca topologia totala a sistemului este definita independent de componentele si conectorii care creeaza sistemul. Sistemele pot fi deasemenea ierarhice: componentele si conectorii pot reprezenta subsisteme care au arhitecturi proprii.

- *Proprietatile* reprezinta informatia semantica despre sistem si componentele sale structurale. Dupa cum a fost precizat si mai devreme, fiecare ADL se concentreaza pe diferite proprietati, dar toate pot oferi anumite proprietati extra-functionale cu ajutorul unor unelte de analiza a acelor proprietati.

- *Constrangerile* se refera la modele arhitecturale care trebuie sa ramana adevarate chiar daca acestea evolueaza de-a lungul timpului. Constrangerile tipice includ restrictii asupra unor valori permise ale unor proprietati, topologiilor, si asupra vocabularului modelelor. De exemplu o arhitectura poate avea constrangeri asupra numarului de clienti pentru un anumit server, astfel incat acesa sa fie mai mic decat o anumita valoare maxima.

- *Stilurile* reprezinta familii de sisteme conexe. Un stil arhitectural tipic isi defineste un vocabular al tipurilor de elemente folosite si al regulilor care pentru alcatuirea lor.

La baza ADL-urilor sta o harta naturala de descriere a nevoilor sistemelor bazate pe componente. In primul rand, ADL-urile permit descrierea alcatuirii componentelor in mod precis, facand explicit modul in care acele componente comunica intre ele. In al doilea rand suporta utilizarea componenteleor cu interfete multiple, o caracteristica cheie a multor sisteme bazate pe componente. In al treilea rand, suporta descrierea ierarhica si incapsularea subsistemelor devenind componente intr-un sistem mai mare. In al patrulea rand, suporta specificarea si analiza proprietatilor nonfunctionale. In al cincilea rand, multe ADL-uri ofera un loc explicit pentru descrierea detaliata a semanticii folosite de infrastructura (prin specificarea tipurilor de conectori). In al saselea rand, ADL-urile permit definirea de constrangeri asupra compozitiei sistemelor care dau detalii despre ce tipuri de componente sunt permise si ce nu. In ultimul rand, stilurile arhitecturale permit diferentierea precisa dintre tipurile de standarde de integrare ale componentelor.

Pentru a putea intelege cum o descriere arhitecturala permite caracterizarea sistemelor bazate pe componente, trebuie sa descriem un tip particular de ADL-uri, numit ACME. Ca un ADL de generatie a doua, ACME are proprietatea distinctiva de fi creata pe experientele celorlalte ADL-uri, oferind intr-un limbaj simplu elemente esentiale de proiectarea arhitecturala si care suporta componente arhitecturale mult mai complexe. In particular, ACME cuprinde ontologia arhitecturala descrisa mai sus, oferind o semantica extensibila si un set de unelte complex pentru analiza arhitecturala si integrarea acelor unelte dezvoltate independent.

ACME suporta definirea a patru aspecte distincte ale arhitecturii. Primul este structura – organizarea unui sistem din partile sale constituente. Al doilea aspect este reprezentat de proprietatile de interes - informatii despre sistem sau despre partile care permit abstractizarea intregului comportament. Al treilea aspect este reprezentat de constrangeri – orientari despre cum se va schimba arhitectura o data cu trecerea timpului. Al patrulea aspect il reprezinta tipurile si stilurile – definirea claselor si familiilor de arhitecturi.

Succesul oricarui limbaj de descriere a sistemului este in cele din urma definit de impactul pe care il are in proiectarea sistemelor software complexe. Desi ACME este inca la inceput, o data cu trecerea timpului acesta a inceput joace roluri cheie in cominitatea arhitecturi software.

Primul rol este acela de limbaj de descriere arhitecturala. Ca un ADL de generatia a doua, ACME a incercat sa cuprinda elementele esentiale ale modelarii arhitecturale. Prin urmare limbajul prezinta un nucleu relativ simplu de concepte pentru definirea structurii sistemului, la care se adauga abilitatea de a extinde acelor concepte utilizant proprietati, constrangeri, tipuri si stiluri care sunt potrivite cu contextul in care sunt utilizate. Un numar de studii de caz au fost facute utilizand ACME, pentru testarea unor sisteme de ghidare a rachetelor si sisteme de localizare globala folosite de Departamentul de Aparare al Statelor Unite[3].

Al doilea rol este unul de baza pentru proiectarea arhitecturii si pentru uneltele de analiza. Pana in prezent o multime de unelte si trei medii de proiectarea au fost construite care suporta descrierea ACME. Uneltele realizeaza o multime de sarcini, incluzand verificarea scrierii, grafice automate, animatii ale comportamentului in timpul rularii sistemului, analize asupra performantelor si sigurantei.

Al treilea rol al ACME este reprezentata de uneltele de integrare. Dupa cum am spus si mai devreme un numar mare de ADL-uri au fost create, fiecare cu setul propriu de unelte. ACME poate fi folosit pentru integrarea multor unelte care ofera o reprezentare comuna pentru celalalte descrieri arhitecturale. Aceasta abilitate se rezuma practic la abilitatea celorlalte unelte de a citi si scrie descrierile ACME. Acest lucru este indeplinit de catre codarea ADL- informatia specifica este vazuta ca o proprietate atasata unui schelet structural generic.

2.3 Concluzii

In cele spuse anterior am argumentat ca limbajele de descriere arhitecturala dau baza fundamentala potrivita pentru descrierea modelelor de sisteme bazate pe componente. Dupa aceea am ilustrat natura acetor descrieri prin prezentarea ACME. In final, am descris rolul pe care limbaje cum ar fi ACME incep sa il joace.

Dezvoltarea ACME a fost un efort comun, cu contributii din partea multor cercetatori. ACME ofera doua beneficii cheie comparativ cu sistemele non-ACME. In primul rand ACME simplifica substantial manevrarea aspectelor structurale ale descrierii si translatiei arhitecturale. In al doilea rand ACME are atat un set proriu de unelte cat si un capabilitatea de a utiliza alte seturi de unelte externe ceea ce face mult mai accesibila utilizarea acestuia[4].

3. TEHNOLOGII DE DEZVOLTARE

3.1 Introducere

Aceasta modalitate de inginerie software bazata pe componente reprezinta de fapt o modalite de dezvoltare a produselor software bazata pe refolosirea altor produse software .

Definitia unei componente conform lui Councill si Heinmann: “O componenta software este un element care , conform unui model, poate fi in mod independent lansata si cumpusa fara modificari in ceea ce priveste un standard de compozitie”.

“In zilele noastre produsele software devin din ce in ce mai complexe si mai greu de controlat , ceea ce duce la costuri mari de dezvoltare , productivitate mica , software care poate fi usor pierdut de sub control ssi deci poate fi un risc destul de mare in ceea ce priveste dezvoltarea de noi tehnologii.”(G.Pour, “Software Component Technologies: JavaBeans and ActiveX,” Proceedings of Technology of Object-Oriented Languages and Systems, 1999).

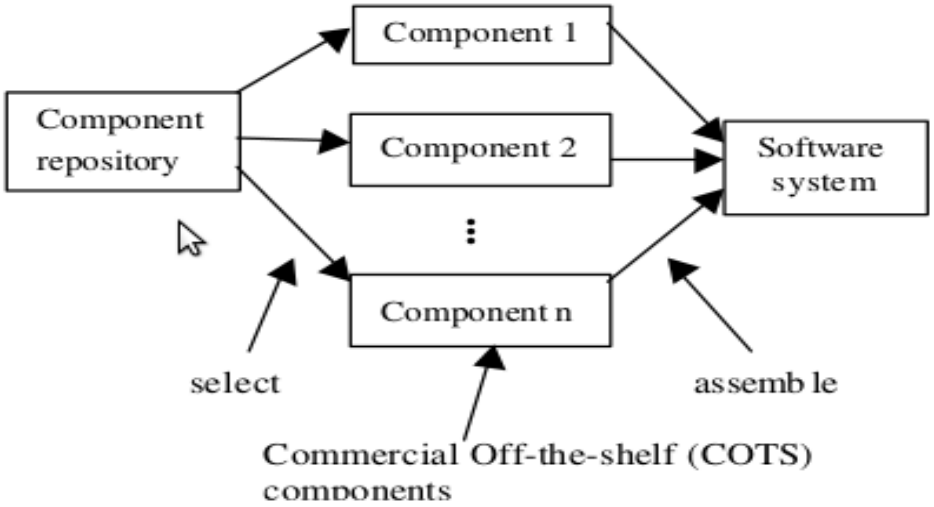
In aceasta maniera , putem afirma faptul ca se cauta noi metode de dezvoltare software eficienta .

Una dintre cele mai promitatoare solutii din zilele noastre este dezvoltarea software bazata pe componente. “Aceasta solutie , de dezvoltare software pe componente , este bazata pe ideea ca sistemele software pot fi dezvoltate prin selectarea unor componenete potrivite urmat apoi de asamblarea acestora respectand o anumita arhitectura software ” (G. Pour, “Component-Based Software Development Approach: New Opportunities and Challenges,” Proceedings Technology of Object-Oriented Languages, 1998).

Acest tip de dezvoltare software se poate observa ca este diferit fata de modalitatea traditionala de dezvoltare in care sistemele era implementate de la zero.

In ceea ce priveste dezvoltarea software pe componente , toate aceste componente pot fi dezvoltate de catre diferiti dezvoltatori , folosind diferite limbaje de programare si diferite platforme.

Urmatoarea schema este sugestiva (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)



Se poate observa in figura de mai sus faptul ca fiecare componenta este preluata dintr-un depozit de componente (Component repository) si apoi asamblata pentru a produce un sistem software.

“Dezvoltarea software pe componente poate reduce in mod semnificativ costul de dezvoltare si poate imbunatati mentenabilitatea fiabilitatea si timpul de buna functionare a sistemelor software” (G.Pour, M. Griss, J. Favaro, “Making the Transition to Component-Based Enterprise Software Development:Overcoming the Obstacles – Patterns for Success,”Proceedings of Technology of Object-Oriented Languagesand systems, 1999).

“Putem afirma ca aceasta modalitate a trezit mult interes atat in comunitatea stiintifica de cercetare cat si in industria software.Ciclul de viata si modelul software al acestei noi modalitati de dezvoltare , dezvoltarea software bazata pe componente , este ceea ce difera fata de vechile modalitati software de dezvoltare”(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

“Pana in acest moment tehnologiile de dezvoltare software pe componente putem spune ca sunt inca in curs de proiectare si dezvoltare , si deci sunt departe de a fi considerate tehnologii mature. Nu exista standarde in aceasta noua zona de dezvoltare si nici nu se poate afirma ca exista o definitie unificatoare , generala a cuvantului cheie 'componenta'.

In general putem spune ca o componenta are 3 mari caracteristici :

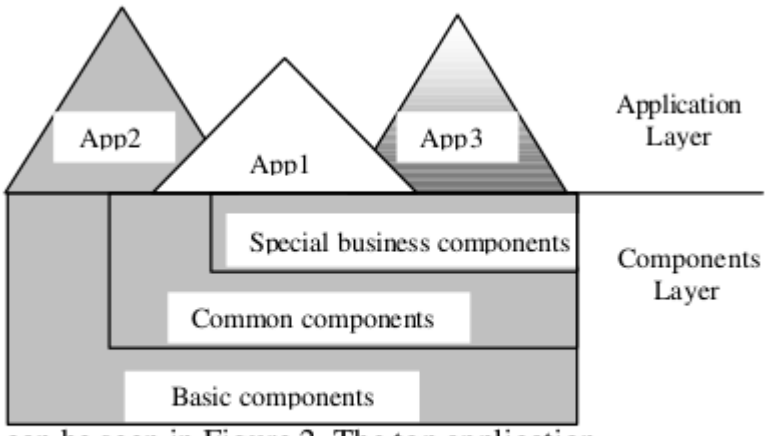
1)O componenta este o parte independenta si care poate fi inlocuita a unui sistem care indeplineste o functie bine stabilita.

2)O componenta poate functiona doar in contextul unei arhitecturi bine definite.

3)O componenta comunica cu celelalte componente prin interfetele sale. “

(A.W.Brown, K.C. Wallnau, “The Current State of CBSE,”IEEE Software , Volume: 15 5, Sept.-Oct. 1998).

“Pentru a asigura faptul ca un sistem bazat de componente poate functiona in mod optim si eficient , trebuie avut mare grija in ceea ce priveste arhitectura sistemului care reprezinta un factor major. Conforma celor sustinute de cei din comunitatea stiintifica si de cei din industria software , arhitectura unui sistem bazat pe componente trebuie sa fie o arhitectura pe niveluri si modulara. Aceasta arhitectura pe niveluri poate fi privita in urmatoarea figura.”



can be seen in Figure 2. The top application

Figure 2. System architecture of component-based software systems

(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong").

"Descrierea arhitecturii este urmatoarea

1)Primul nivel (Application layer) reprezinta nivelul pentru aplicatii care stau la baza unei afaceri.

2)Cel de-al doilea nivel reprezinta nivelul de componente si consta din alte 3 subniveluri.

2a) Acest nivel consta din componente implicate intr-o anumita afacere(componente speciale) sau aplicatie,incluzand componentele care pot fi folosite in mai multe aplicatii.

2b)Acest nivel este un nivel de mijloc si este format din software comun si interfete catre alte entitati (componente).

2c)Acest nivel este nivelul cel mai de jos si include componentele de baza care interfateaza cu sistemul de operare si cu resursele hardware. “

(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

Putem concluziona afirmand faptul ca aceasta modalitate de a dezvolta produse software este una ieftina si rapida si in multe situatii chiar de calitate deoarece se stie ca refolosirea codului este una din beneficiile programarii orientate pe obiect , daca este implementata eficient , dar trebuie avut grija la unele probleme ce pot aparea in anumite situatii.

“Pana in prezent au fost folosite diferite tehnologii cu scopul de a implementa diferite sisteme software bazate pe componente ca de exemplu componentele software distribuite orientate pe obiect si aplicatii de tip Web-based enterprise“.

(G. Pour, “Enterprise JavaBeans, JavaBeans & XML Expanding the Possibilities for Web-Based Enterprise Application Development,” Proceedings Technology of Object-Oriented Languages and Systems).

“Exista de asemenea si niste parti comerciale implicate in acett tip de inginerie softare bazata pe componente care se ocupa de realizare acestor tehnologii necesare pentru realizarea aplicatiilor bazate pe componente , aceste parti sunt : BEA , Microsoft , IBM si SUN.”

(W. Kozaczynski, G. Booch, “Component-Based Software Engineering,” IEEE Software Volume: 155, Sept.-Oct. 1998).

“Un exemplu semnificativ din acest domeniu este proiectul celor de la Ibm numit SanFrancisco care ofera o insfrastructura distribuita orientata pe obiect ce poate fi refolosita si mai ofera un set de componente ce poat fi folosite de catre dezvoltatorii de aplicatii.”

(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

3.2 Exemple de tehnologii folosite pentru dezvoltarea software bazate pe componente

“Anumite standarde si servicii ca de exemplu Visual Basic Controls (VBX) , ActiveX controls , librarii de clase , si JavaBeans pot face posibil ca pentru limbajele lor afiliate cum ar fi VisualBasic , C++ si Java sa poate distribui si realiza diferite parti de aplicatii. Dar , toate aceste standarde si servicii se bazeaza pe anumite cerinte in ceea ce priveste necesitatea de e realiza o comunicare si imbinare a componentelor rezultate intr-o aplicatie.”(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

“Infrastructura acestor componente , uneori numita si model de componente , poate fi asemanata cu o conducta care permite comunicarea intre componente . Printre aceste infrastructuri , putem numi ca tehnologii care au fost deja dezvoltate doar 3 care au fost standardizate si anume : CORBA , COM(Microsoft's Component Object-Model) si DCOM(Distributed COM) , si Java Beans si Enterprise JavaBeans apartinant celor de la Sun.”

(W. Kozaczynski, G. Booch, “Component-Based Software Engineering,” IEEE Software Volume: 155).

O prima tehnologie este CORBA (Common Object Request Broker Architecture) care reprezinta un standard pentru aplicatii si care este definit si intretinut de OMG (Object Management Group) , o organizatie ce realizeaza produse software pentru diferite ramuri .

CORBA se ocupa de partea de comunicare dintre diferitele componente ale unei aplicatii in ciuda diferitilor dezvoltatori si locatii care participa la dezvoltarea produsului software. Una dintre caile prin care componentele unei aplicatii comunica intre ele o reprezinta interfata.

“Cea mai importanta parte a unui sistem CORBA este ORB (Object Request Broker) care reprezinta un middleware ce stabileste relatii de tip client-server intre componente. Folosind un astfel de ORB , clientul poate invoca o metoda asupra unei instante (obiect) a serverului a care locatie este complet transparenta. ORB este responsabil de interceptia apelului metodei si de gasirea obiectului care poate

indeplini aceasta cerinta , paseaza parametrii metodei , apeleaza metoda si returneaza rezultatele. Clientul nu este neaparat sa stie unde este obiectul localizat , limbajul sau de programare , sistemul sau de operare sau alte aspecte ale sistemului care nu au legatura cu interfata. CORBA este des folosit in sisteme orientate pe obiect distribuite , in sistemele software bazate pe componente" (S.S.Yau, B. Xia, "Object-Oriented DistributedComponent Software Development based on CORBA," Proceedings of COMPSAC'98. The Twenty-Second Annual International).

O a doua tehnologie este COM (Component Object Model) si respectiv DCOM (Distributed COM). Tehnologia COM a aparut pentru prima oara in anul 1993 si reprezinta o arhitectura generala pentru produsele software bazate pe componente. Aceasta ofera aplicatii pe componenente dependente de platforme (bazate pe Windows , Windows NT) si independente de limbaj. "COM defineste modul in care componentele si clientii interactioneaza.Aceasta interactiune este definita astfel incat clientul si componenta se pot conecta fara o alta componenta de sistem intermediara.In mod special , COM ofera un standard binar pe care componenetele ci clientii trebuie sa il respecte pentru a asigura interoperabilitate dinamica. Aceasta include on-line software update si refolosirea de limbaj " (Y.M.Wang, O.P.Damani, W.J. Lee, "Reliability and Availability Issues in Distributed Component Ojbect Model (DCOM),").

Ca o extensie a COM-ului apare DCOM care reprezinta un protocol care activeaza componentele software pentru a comunica direct intr-o retea , intr-o maniera sigura si eficienta.

"DCOM a fost realizat pentru a folosit in foarte multe transporturi de retele , incluzand protocoale de internet cum ar fi HTTP. Atunci cand un client si componenta si ruleaza de pe statii diferite , DCOM inlocuieste comunicarea locala interproces cu un simplu protocol de retea.Nici clientul si nici componenta nu pot observa schmbarile fizice ale retelei."(Y.M.Wang, O.P.Damani, W.J. Lee, "Reliability and Availability Issues in Distributed Component Ojbect Model (DCOM),").

O a treia tehnologie este JavaBeans si respectiv Enterprise JavaBeans

“Modelul de tehnologie in ceea ce priveste dezvoltarea de componente a celor de la Sun bazata pe tehnologia Java consta din 2 parti : partea de client JavaBeans si partea de server Enterprise JavaBeans.Componenta arhitecturala bazata pe JavaBeans poate suporta aplicatii de pe platforme diferite si in acelasi timp suporta si aplicatii refolosite , componente de tipul client sau de tipul server. “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

“Platforma Java ofera o solutie eficienta problemelor de securitate de portabilitate prin folosirea unui concept de Java applect Java ofera o integrare universala si o tehnologie ce poate fi usor folosita in dezvoltarea de produse software bazate pe componente. JavaBeans si extensiile de EJB reprezinta niste puncte forte ale Java in ceea ce priveste securitatea , portabilitatea si eficienta sistemelor bazate pe componente . Portabilitatea si securatatea fac din Java o tehnologie importanta pentru dezvoltarea obiectelor de tip server independente de sistemul de operare , servere Web si servere de baze de date.”(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong).

O scurta comparatie intre tehnologii poate fi realizata in urmatorul tabel.

	CORBA	EJB	COM/DCOM
Development environment	Underdeveloped	Emerging	Supported by a wide range of strong development environments
Binary interfacing standard	Not binary standards	Based on COM; Java specific	A binary standard for component interaction is the heart of COM
Compatibility & portability	Particularly strong in standardizing language bindings; but not so portable	Portable by Java language specification; but not very compatible.	Not having any concept of source-level standard of standard language binding.
Modification & maintenance	CORBA IDL for defining component interfaces, need extra modification & maintenance	Not involving IDL files, defining interfaces between component and container. Easier modification & maintenance.	Microsoft IDL for defining component interfaces, need extra modification & maintenance
Services provided	A full set of standardized services; lack of implementations	Neither standardized nor implemented	Recently supplemented by a number of key services
Platform dependency	Platform independent	Platform independent	Platform dependent
Language dependency	Language independent	Language dependent	Language independent
Implementation	Strongest for traditional enterprise computing	Strongest on general Web clients.	Strongest on the traditional desktop applications

Acest tabel a fost luat din articolul ” “Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong “ si am considerat ca nu are nevoie de traducere deoarece este usor de inteles.

3.3 Ciclul de viata al unui sistem software bazat pe componente

Systemele software bazate pe componente sunt dezvoltate prin selectarea diferitelor componente si asamblarea lor spre deosebire de cazul in care se vrea dezvoltarea generala a unui proiect de la inceput , si deci in acest caz ciclul de viata al sistemelor software bazate pe componente este diferit de sistemele software dezvoltate in mod traditional.

“Ciclul de viata al sistemelor bazate pe componente poate fi rezumat astfel :

1) Analiza cerintelor
2) Arhitectura software in ceea ce priveste selectia,constructia,analiza si evaluarea
3) Identificarea componentelor si customizarea lor
4) Integrarea sistemului
5) Testarea sistemului
6) Mentananta produsului.

Arhitectura software defineste un sistem in termeni de componente computationale si interactiuni intre componente. Principala problema o reprezinta alcatuirea si asamblarea componentelor care sunt presupuse a fi dezvoltate separat si chiar independent. .”(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

“Identificarea componentelor , customizarea si integrarea este cruciala in ciclul de viata al sistemelor software bazate pe componente . Aceast proces amplu include 2 parti:

1) Evaluarea fiecarei componente candidat bazate pe functionalitatea si calitatea cerintelor care vor fi folosite pentru a evalua aceasta componenta.
2) Customizarea acelor componente candidat care ar trebui modificate inainte de integrarea componentei candidata in sistemul soft bazat pe componente.

Integrarea este data de luarea unor decizii cheie despre cum se poate oferi comunicare si coordonare in ceea ce priveste diferitele componente ale unui sistem software bazat pe componente. “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.4 Un model de QA (Quality Assurance) pentru sistemele software bazate pe componente

“Deoarece sistemele software bazate pe componente sunt dezvoltate printr-un proces diferit fata de sistemele obisnuite , modelul de QA asociat acestora ar trebui sa faca referire atat la procesul de design al componentelor cat si la procesul de design al sistemului. Acest lucru poate fi observat in figura urmatoare unde avem componenta si respectiv sistemul format din componente.

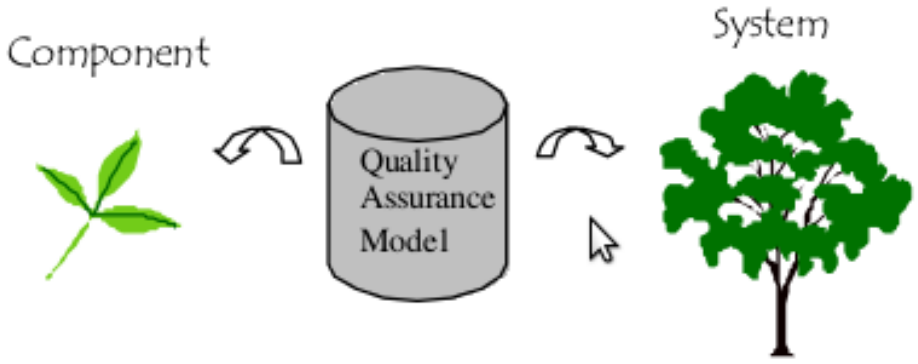


Figure 3. Quality assurance model for both components and systems

In particular , Consiliul din Hong Kong a dezvoltat un astfel de model numit HPSQA . Prevederile din acest model referitoare la sistem si la componente contin urmatoarele faze :

1) Necesitatea analizei componentelor

2) Dezvoltarea componentelor

3) Certificarea componentelor

4) Customizarea componentelor

5) Design-ul arhitecturii sistemului

6) Integrarea sistemului

7) Testarea sistemului

8) Intretinerea sistemului “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

(“Hong Kong Productivity Council, http://www.hkpc.org , April, 2000”)

3.5 Necesitatea analizei componentelor

“Aceasta faza reprezinta procesul de descoperire , intelegere , documentare , validare si manageriere a cerintelor unei componente. Obiectivele analizei cerintelor componentei sunt cu scopul de a produce cerintele relevante si complete pe care o componenta ar trebui sa le realizeze . Aceste cerinte tb indeplinite de asemenea si de catre limbajul de programare , de platforma de dezvoltare si de interfetele asociate componentei. “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.6 Dezvoltarea componentelor

“Dezvoltarea componentelor este procesul de implementare a cerintelor cu scopul a obtine o componente functionala , de calitate si cu multiple interfete. Obiectivul acestei etape il reprezinta componentele finale , interfetele si dezvoltarea documentelor . Dezvoltarea componentelor ar trebui sa conduca la componente finale de o calitate care sa satisfaca necesitatile . “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.7 Certificarea componentelor

“Acesta faza presupune 2 subetape :

1) Aflarea sursei componentei
2) Selectarea componentei
3) Testarea componentei

Obiectivul acestei etape este de a verifica sursa componentei , de a testa si de a selecta componentele candidate si de a verifica daca cestea satisfac cerintele impuse pentru a crea sistemul bazat pe componente dori “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.8 Customizarea componentelor

“Aceasta etapa presupune urmatoarele 3 subetape

1) Modificarea componentei pentru a realiza o anumita cerinta
2) Realizarea anumitor schimbari necesare pentru a rula componenta pe o anumita platforma
3) Upgradarea unei anumite componente pentru a obtine performante mai bune

Obiectivul acestei etape este de a face schimbarile necesare pentru o componenta dezvoltata cu scopul de a se adapta anumitor cerinte si pentru a coopera cu alte componente.

Toate componentele tb sa fie customizate in raport cu cerintele sistemului de operare sau in raport cu cerintele interfetei altor componente cu scopul de a putea fi imbinate. “(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.9 Design-ul arhitecturii sistemului

“Aceasta etapa reprezinta un proces de evaluare , selectare si de creare a arhitecturii software necesare pentru un sistem bazat pe componente.

Obiectivele acestei etape sunt de a colectiona cerintele utilizatorilor , de a identifica specificatiile de sistem , de a selecta arhitectura de sistem care se potriveste si de a realiza detaliile necesare implementarii cum ar fi platforma de dezvoltare , limbajele de programare.”(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.10 Integrarea sistemului

“Aceasta este o etapa de asamblare a componentelor selectate intr-un intreg sistem care corespunde unei anumite arhitecturi alese in etapa anterioara.

Obiectivele acestei etape il reprezinta formarea sistemului prin combinarea componentelor selectate . “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.11 Testarea sistemului

“Aceasta este o etapa de testare si de avaluare a sistemului cu scopul de a

1) Confirma ca sistemul respecta cerintele date
2) Identifica si corecta defectele aparute in implementarea sistemului

Obiectivul acestei etape de testarea a sistemului este de a integra componentele selectate in concordanta cu cerintele sistemului. Testarea sistemului ar trebui sa contina testarea de functii si testarea fiabilitatii. “ (“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

3.12 Intretinerea sistemului

“Aceasta etapa este procesul prin care se ofera suport si mentenanta in ceea ce priveste folosirea corecta a sistemului bazat pe componente ce a fost realizat.

Obiectivul acestei etape este de a oferi sprijin eficient utilizatorilor in ceea ce priveste eventuale erori de folosire , imbunatatirea performantelor software , si adaptarea sistemului la un mediu schimbat. “(“Component-Based Software Engineering”-Xia Cai,Michael R.Lyu , University of Honk Kong)

4. MODELE

4.1. Modelul COM

Component Object Model (COM) este o interfata binara standard de componente software introdusa de Microsoft in anul 1992. Este folosita pentru comunicatii inter-proces si creare dinamica de obiecte intr-o gama larga de limbaje de programare. COM sta la baza mai multor tehnologii si arhitecturi dezvoltate de Microsoft, inclusiv OLE, OLE Automation, ActiveX, COM+ si DCOM.

Esenta tehnologiei COM este reprezentata de faptul ca aceasta e o modalitate independenta de limbaj de a implementa obiecte care pot fi folosite in medii diferite fata de cele in care au fost create, netinand cont de limitele masinii. Pentru componente bine definite, COM permite refolosirea obiectelor fara cunostinte despre implementarea interna a lor, pentru ca ii forteaza pe programatori sa furnizeze interfete bine definite, separate de implementare. Diferitele tipuri de structuri semantice ale limbajelor sunt potrivite prin responsabilizarea obiectelor in legatura cu propria creare si distrugere - acest lucru se realizeaza prin prin numararea referintelor (reference counting). Transportul intre diferitele interfete ale unui obiect este realizat prin functia *QueryInterface()*. Metoda preferata de mostenire in modelul COM este crearea de sub-obiecte catre care sunt delegate apelurile catre metode.

COM este o tehnologie de interfata definita si implementata ca standard doar in Microsoft Windows, in Apple - Core Foundation 1.3 si, mai tarziu, in API. Pe de alta parte, obiectele COM pot fi folosite cu toate limbajele .NET prin .NET COM Interop. DCOM in retea utilizeaza formate binare, pe cand WCF incurajeaza utilizarea mesajelor SOAP bazate pe XML. COM este similar cu alte interfete software bazate pe componente, precum CORBA si Java Beans, fiecare dintre ele avand propriile avantaje si dezavantaje.

Spre deosebire de C++, COM asigura o interfata ABI stabila care nu se schimba intre diverse versiuni ale compilatoarelor. Acest aspect face interfetele COM atractive pentru librariile C++ orientate pe obiecte care sunt folosite de catre clienti si compilate de diverse versiuni de compilatoare.

COM este o platforma de dezvoltare importanta pentru Windows si, astfel, a influentat progresul mai multor alte tehnologii.

4.1.1. COM+

Pentru ca Microsoft sa le ofere programatorilor suport atat pentru tranzactii sau resurse distribuite, aplicatii deconectate, publicarea si inregistrarea evenimentelor, gestiunea eficienta a memoriei si a procesorului, cat si pentru a pozitiona sistemul de operare Windows ca o alternativa la alte sisteme de operare la nivel de afaceri, Microsoft a introdus o tehnologie numita Microsoft Transaction Server (MTS) pe Windows NT4.

O data cu Windows 2000, aceasta extensie semnificativa a COM-ului a fost incorporata in sistemul de operare (fata de modul in care se folosea pana atunci - printr-o serie de unelte externe furnizate de MTS) si a fost redenumita, sugestiv, COM+. In acelasi timp, Microsoft nu a mai considerat modelul DCOM ca o entitate separata. Componentele care foloseau serviciile COM+ erau tratate mai usor de catre stratul adaugat COM+, in particular de catre suportul sistemului de operare legat de interceptari. In prima versiune a MTS, interceptarea era "insailata" o data cu instalarea unei componente MTS, Windows Registry va apela software-ul MTS, nu direct componenta.

Un avantaj al COM+ este ca putea fi folosit in "ferme de componente". Instante ale unei componente, daca erau codate corect, puteau fi cumulate si refolosite de catre noi apeluri pentru rutinele de initializare, fara a fi sterse din memorie. Componentele puteau fi, de asemenea, distribuite (puteau fi apelate de catre o alta masina). COM+ si Microsoft Visual Studio au furnizat echipamente pentru a usura generarea de proxy-uri din partea clientului, astfel incat, desi pentru efectuarea apelului propriu-zis la distanta se folosea DCOM, era usor de folosit de catre dezvoltatori.

COM+ a introdus si un mecanism de evenimente editor/abonat, numit COM+ Events si a furnizat o noua metoda de a gestiona MSMQ (schimb de mesaje asincron intre aplicatii) cu componente numite Queued Components. Eventimentele COM+ extind modelul de programare COM+ pentru a suporta evenimente legate tarziu sau apeluri de metode dintre editor/abonat si sistemul de evenimente.

4.1.2 .NET

.NET este o platforma de dezvoltare software unitara care permite realizarea, distribuirea si rularea aplicatiilor-desktop Windows si a aplicatiilor WEB. Tehnologia .NET pune laolalta mai multe tehnologii (ASP, XML, OOP, SOAP, WDSL, UDDI) si limbaje de programare (VB, C++, C#, J#) asigurand totodata atat portabilitatea codului compilat intre diferite calculatoare cu sistem Windows, cat si reutilizarea codului in programe, indiferent de limbajul de programare utilizat.

.NET Framework este o componenta livrata impreuna cu sistemul de operare Windows. De fapt, .NET 2.0, ce vine cu Windows Server 2003, se poate instala pe versiunile anterioare, pana la Windows 98 inclusiv; .NET 3.0 vine instalat pe Windows Vista si poate fi instalat pe versiunile Windows XP cu SP2 si Windows Server 2003 cu minimum SP1.

Pentru a dezvolta aplicatii pe platforma .NET este bine sa avem 3 componente esentiale:

- un set de limbaje (C#, Visual Basic .NET, J#, Managed C++, Smalltalk, Perl, Fortran, Cobol, Lisp, Pascal etc),
- un set de medii de dezvoltare (Visual Studio .NET, Visio),
- si o biblioteca de clase pentru crearea serviciilor Web, aplicatiilor Web si aplicatiilor desktop Windows.

Atunci cand dezvoltam aplicatii .NET, putem utiliza:

- Servere specializate - un set de servere Enterprise .NET (din familia SQL Server 2000, Exchange 2000 etc), care pun la dispozitie functii de stocare a bazelor de date, email, aplicatii B2B (Bussiness to Bussiness – comert electronic intre partenerii unei afaceri).
- Servicii Web (in special comerciale), utile in aplicatii care necesita identificarea utilizatorilor (de exemplu, .NET Passport - un mod de autentificare folosind un singur nume si o parola pentru toate ste-urile vizitate)
- Servicii incluse pentru dispozitive non-PC (Pocket PC Phone Edition, Smartphone, Tablet PC, Smart Display, XBox, set-top boxes, etc.)

Componenta .NET Framework ce sta la baza tehnologiei .NET, este ultima interfata intre aplicatiile .NET si sistemul de operare si actualmente contine:

- Limbajele C#, VB.NET, C++ si J#. Pentru a fi integrate in platforma .NET toate aceste limbaje respecta niste specificatii OOP numite *Common Type System* (CTS). Ele au ca elemente de baza: clase, interfete, delegari, tipuri valoare si referinta, iar ca mecanisme: mostenire, polimorfism si tratarea exceptiilor.
- Platforma comuna de executare a programelor numita *Common Language Runtime* (CLR), utilizata de toate cele 4 limbaje. CTS face parte din CLR.
- Ansamblul de biblioteci necesare in realizarea aplicatiilor desktop sau Web numit *Framework Class Library* (FCL).

Un program scris intr-unul dintre limbajele .NET conform *Common Language Specification* (CLS) este compilat in *Microsoft Intermediate Language* (MSIL sau IL). Codul astfel obtinut are extensia *exe*, dar nu este direct executabil, ci respecta formatul unic MSIL.

CLR include o masina virtuala asemanatoare cu o masina Java, ce executa instructiunile IL rezultate in urma compilarii. Masina foloseste un compilator special JIT (*Just In Time*).

Compilatorul JIT analizeaza codul IL corespunzator apelului unei metode si produce codul masina adecvat si eficient. El recunoaste secventele de cod pentru care s-a obtinut deja codul masina adecvat permitand reutilizarea acestuia fara recompilare, ceea ce face ca, pe parcursul rularii, aplicatiile .NET sa fie din ce in ce mai rapide.

Faptul ca programul IL produs de diferitele limbaje este foarte asemanator are ca rezultat interoperabilitatea intre aceste limbaje. Astfel, clasele si obiectele create intr-un limbaj specific .NET pot fi utilizate cu succes in altul. In plus, CLR se ocupa de gestionarea automata a memoriei (un mecanism implementat in platforma .NET fiind acela de eliberare automata a zonelor de memorie asociate unor date devenite inutile – *Garbage Collection*).

4.1.3 Windows Runtime

Noua tehnologie dezvoltata de Microsoft, modelul de programare si aplicatia Windows Runtime (sau WinRT) este, in esenta, un API bazat pe arhitectura COM. Datorita proprietatilor mostenite din arhitectura COM, Windows Runtime admite interfatare usoara pentru mai multe limbaje, la fel ca si COM, dar este in acelasi timp un API nativ. Definitiile API sunt stocate in fisiere "*.winmd*", care sunt codate in formatul de date ECMA 335, acelasi format pe care il foloseste si platforma .NET, cu cateva modificari.

Programatorii COM isi construiesc software-ul folosind componente COM. Diferite tipuri de componente sunt identificate de catre ID-urile de clasa (CLSID), care sunt Indentificatori Unici Globali (GUIDs). Fiecare componenta COM isi expune functionalitatea prin una sau mai multe interfete. Diferitele interfete suportate de catre o componenta sunt deosebite una de alta folosind ID-urile interfetei (IID), care sunt si ele, la randul lor, GUID.

Interfetele COM contin legaturi cu diferite limbaje, ca C, C++, Visual Basic, Delphi, precum si alte limbaje ce utilizeaza scripturi implementate de catre platforma Windows. Accesul catre componente este realizat prin metodele interfatelor. In acest fel se aproba tehnici ca programare intre procese sau chiar intre calculatoare (acesta folosind DCOM).

Toate componentele COM trebuie cel putin sa implementeze interfata standard *IUnknown*. Astfel, toate interfetele COM sunt derivate din *IUnknown*. Interfata *IUnknown* contine 3 metode: *AddRef(), Release()* (care implementeaza contorul referintelor si controleaza timpul de viata al interfetelor) si

QueryInterface(), care prin specificarea unui IID permite unui apelant sa retina referinte de la diferitele interfete pe care le implementeaza componenta. Efectul metodei *QueryInterface()* este similar cu *dynamic_cast*<> in C++.

Interfetele unei componente COM trebuia sa aiba proprietati de reflexivitate, simetrie si tranzitivitate. Proprietatile de reflexivitate se refera la abilitatea metodei unei apelari *QueryInterface()* pe o interfata anume cu ID-ul interfetei de a returna aceeasi instanta a interfetei. Proprietatea de simetrie reprezinta faptul ca atunci cand interfata B este regasita din interfata A prin *QueryInterface()*, interfata A poate fi regasita din interfata B deasemenea. Proprietatea de tranzitivitate se refera la faptul ca daca interfata B poate fi obtinuta din interfata A si interfata C poate fi obtinuta din interfata B, atunci interfata C ar trebui sa poata fi obtinuta din interfata A.

O interfata este constituita dintr-un pointer spre un tabel virtual de functii care contine o lista de pointeri la functiile care implementeaza functiile declarate in interfata, in aceeasi ordine in care sunt ele declarate.

COM specifica si alte interfete standard folosite pentru a permite comunicatia intre componente. De exemplu, una din aceste interfete este *IStream*, care este expusa de componente care au aceleasi desfasurari semantice (de exemplu o componenta *FileStream* folosita pentru a citi si a a scrie fisiere). O alta interfata standard folosita este *IOleObject*, care este expusa de componente care asteapta sa fie legate intr-un container.

O clasa este o metoda a COM independenta de limbaj,definita similar modului in care este definita o clasa in sensul obiect-orientat.

O clasa poate fi o grupare de obiecte similare sau o simpla reprezentare a unui anumit tip de obiect. Se poate spune ca o clasa este "amprenta" ce descrie un obiect.

O coclasa furnizeaza implementari concrete a una sau mai multor interfate. In COM, aceste implementari concrete pot fi scrise in orice limbaj de programare care suporta o dezvoltare bazata pe componente COM (de exemplu Delphi, C++, Visual Basic etc)

4.2. Modelul Enterprise JavaBeans

Enterprise JavaBeans este o arhitectura bazata pe componente care simplifica procesul dezvoltarii aplicatiilor bazate pe componente distribuite intr-un mediu de productie. Folosind EJB, se pot scrie aplicatii scalabile, sigure si securizate fara scrierea in prealabil a unui framework de componente distribuite. Principalul scop al arhitecturii EJB este acela de a elibera producatorul (dezvoltatorul) de aplicatii enterprise (pentru corporatii) de aspectele de nivel sistem ale unei aplicatii (securitatea, controlul resurselor si al proceselor, controlul tranzactiilor, etc.). Odata eliberat de aceste probleme, dezvoltatorul (creatorul) de bean-uri se poate preocupa doar de aspectele de business ale aplicatiei (functionalitatea la nivel inalt a aplicatiei). Platforma EJB ofera creatorului de soft un model de aplicatie distribuita si pe mai multe nivele, abilitatea de a refolosii componente, manipularea interna a datelor prin XML, un model de secuitate unificat si control flexibil al tranzactiilor, independenta de platforma.

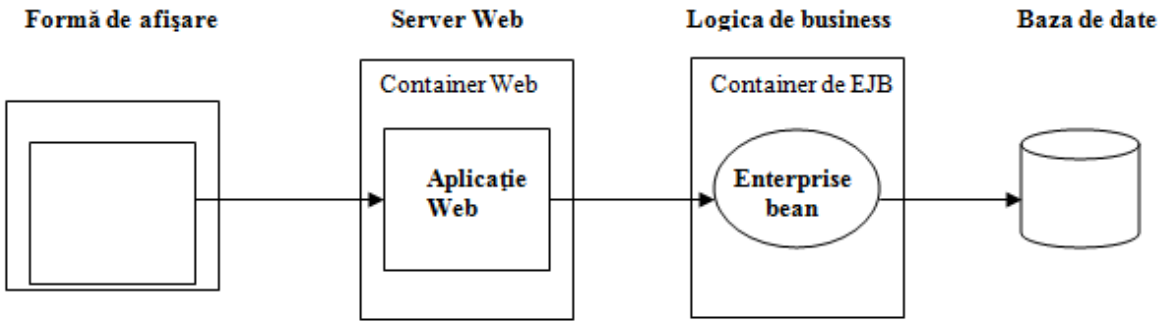
In 1998 la JavaOne a fost lansata arhitectura EJB (Enterprise JavaBeans) si totodata a fost anuntata platforma free Java Platform for the Enterprise (JPE) pusa la dispozitie de SUN pentru dezvoltarea aplicatiilor EJB care mai tarziu avea sa fie redenumita in Java 2 Enterprise Edition (J2EE). Arhitectura EJB si platforma J2EE aveau sa se se constituie intr-un cadru de dezvoltare al aplicatiilor pentru corporatii (enterprise application) pe 4 nivele (4-tier) care cuprindea majoritatea tehnologiilor Java (API - Application Programming Interface) dezvoltate pana la acea vreme:

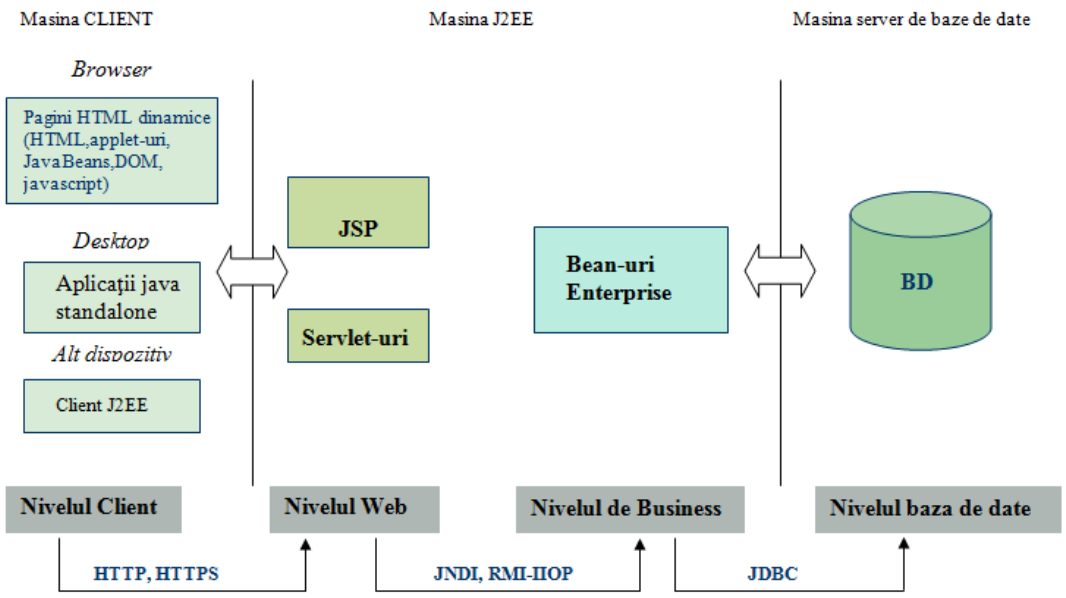
- JDBC (Java DataBase Conectivity): API pentru acesarea bazelor de date relationale
- Java Servlet Tehnology: componente retea care extind functionalitatea unui sever web; sunt folosite pentru generarea pagunilor html dinamice
- JSP (Java Server Pages): asemanatoare cu servleturile, paginile jsp sunt documente html care contin pe langa cod html si cod java
- JMS (Java Message Service): API pentru comunicarea intre doua sisteme prin mesaje asincrone
- JNDI (Java Naming and Directory Interface): API pentru accesarea obiectelor prin servicii de nume si directoare (naming and directory service)
- JTA (Java Transaction API): API pentru integrarea componentelor (bean-urilor) cu suportul de tranzactii
- Java Mail API: servicii ce permit trimiterea mesajelor de email intr-o maniera independenta de platforma si independenta de protocol, din interiorul programelor java.

- JAF (JavaBeans Activation Framework): servicii standard pentru determinarea tipului unei date arbitrare, incapsularea accesului la ea, descoperirea operatiilor posibile asupra ei si crearea unei componente JavaBean care sa indeplineasca aceste operatii
- JAXP (Java API for XML Processing): API pentru parsarea documentelor XML
- J2EE Connector Architecture: API pentru accesarea sistemelor de informatii externe (sisteme mainframe cu suport puternic de tranzactii, sisteme ERP – Enterprise Resource Planning -, etc.)
- JAAS (Java Authentification and Authorization Service): API pentru efectuarea de operatii legate de securitatea sistemelor

Aceasta noua arhitectura de dezvoltare a aplicatiilor avea sa fie o arhitectura dependenta de limbaj (Java) care avea sa respecte din plin principiul Java al portabilitatii: ”Write Once, Run Anywhere.”
Arhitectura EJB cuprinde urmatoarele 4 nivele:

- nivelul client ale carui componente ruleaza pe masina client
- nivelul web ale carui componente ruleaza pe serverul J2EE
- nivelul de business al aplicatiei ale carui componente (bean-uri enterprise) ruleaza pe serverul J2EE
- nivelul bazei de date a aplicatiei care exista pe un server de baze de date





4.2.1 Nivelul client

In cadrul nivelului client se regasesc aplicatii client sau clienti web. Un client web cuprinde pagini web dinamice continand diferite tipuri de limbaje de marcaj (HTML, XML, etc.) generate de componente web care ruleaza la nivelul web (pagini jsp, servleturi), si un browser web care sa interpreteze aceste documente. Acest tip de client web se mai numeste *thin client* deoarece el nu face interogari la baze de date, el nu executa procese de business complexe. Un client web poate fi de asemenea un applet java care este o clasa java care se executa in masina virtuala java (JVM) a browserului.

4.2.2 Nivelul web

Nivelul web cuprinde servlet-uri si pagini jsp. Servlet-urile sunt clase java care proceseaza in mod dinamic cereri si construieste raspunsuri. La fel sunt si paginile jsp, numai ca, spre deosebire de servleturi ele ofera o perspectiva mai apropiata de html-ul static.

4.2.3 Nivelul de business

Nivelul de business cuprinde bean-uri enterprise care rezolva logica de business a aplicatiei sau logica aplicatiei. Un bean enterprise preia date le

prelucreaza si le scrie in baza de date sau invers: citeste date din baza de date, le prelucreaza si le intoarce la client. EJB defineste trei tipuri de bean-uri: bean-uri sesiune, bean-uri entitate si bean-uri orientate pe mesaje. Un bean sesiune reprezinta o conversatie tranzitorie cu clientul; cand clientul isi termina executia bean-ul impreuna cu datele lui nu mai exista. Pe de alta parte, un bean entitate reprezinta date persistente stocate intr-o inregistrare a unui tabel al bazei de date.

4.2.4 Nivelul de date (baza de date)

La acest nivel se afla de regula un server de baze de date care are suport pentru tranzactii si recunoaste limbajul SQL.

Pentru a se elibera de sarcinile ce tin mai mult de sistemul de operare decat de aplicatie (gestiunea tranzactiilor, pooling de resurse si instante, multithreading, protocoale distribuite, gestiunea proceselor, politicile de securitate), componentele J2EE, se sprijina pe serviciile oferite de *containere*. Astfel ca, componentele J2EE se pot preocupa numai de partea de business a aplicatiei (functionalitatea la nivel inalt a ei). Un *container EJB* este un server/framework care se ocupa cu aceste detalii ce tin de sistemul de operare, si se constituie ca o interfata intre componenta si functiile low-level specifice ale platformei care suporta componenta. Containerele platformei J2EE sunt urmatoarele:

- container EJB
- container web
- container de aplicatii client

4.2.5 Bean Enterprise

Un bean enterprise este o componenta server-side care incapsuleaza logica de business a unei aplicatii. In principiu, un bean enterprise incapsuleaza un obiect de business. Bean-urile enterprise traiesc in interiorul containerelor EJB. Acesta le asigura bean-urilor servicii de nivel sistem ca tranzactii, managementul thread-urilor, etc. Aceste servicii care sunt implementate de container fac posibila dezvoltarea rapida a bean-urilor, pentru ca programatorul se va concentra numai pe rezolvarea problemei de business respective. Apoi, datorita acestei delimitari clare intre partea de business implementata in componente EJB si partea de prezentare, in cazul in care specificatia de business se schimba, nu este nevoie decat sa rescriem bean-ul care implementeaza functionalitatea respectiva, partea de prezentare ramanand neschimbata, rezultand astfel *thin clients*, adica aplicatii care ruleaza usor pe masina client, fiindca nu cer multe resurse. Bean-urile enterprise pot rula pe orice server compatibil J2EE.

Interfata de baza pe care toate bean-urile, indiferent de tipurile lor, trebuie sa o implementeze este *javax.ejb.EnterpriseBean* care nu contine nici o metoda si extinde *java.io.Serializable*. Bean-urile sesiune, entitate si orientate pe mesaje au fiecare interfete specifice pe care trebuie sa le implementeze, interfete care extind *javax.ejb.EnterpriseBean*: bean-urile sesiune tebuie sa implementeze interfata *javax.ejb.SessionBean*, bean-urile entitate trebuie sa implementeze *javax.ejb.EntityBean*, iar bean-urile orientate pe mesaje trebuie sa implementeze *javax.ejb.MessageDrivenBean*. De aceea, clasa bean nu implementeaza niciodata direct interfata *javax.ejb.EnterpriseBean*.

4.3 Modelul CORBA

Common Object Request Broker Architecture (CORBA) este un stadard definit de catre Object Management Group (OMG) care permite scrierea componentelor software in diferite limbaje si rularea lor pe mai multe calculatoare care lucreaza impreuna (suporta multiple platforme).

CORBA automatizeaza multe sarcini uzuale ale programarii relelelor, de exemplu inregistrarea obiectelor, localizarea si activarea lor, demultiplexarea cererilor, gestiunea erorilor, triajul parapetrilor si rezolvarea operatiilor. Datorita usurintei cu care CORBA integreaza masini de la diferiti furnizori, cu dimensiuni variind de la mainframes si desktop pc la sisteme integrate, acest standard este de multe ori preferatul intreprinderilor mari. Una din cele mai importante si cele mai des folosite utilizari este in servere care trebuie sa gestioneze un numar mare de clienti la rate de succes mari cu o mare fiabilitate. CORBA lucreaza in backgroundul a celor mai mari website-uri, chiar si unele folosite zi de zi de catre majoritatea persoanelor.

Aplicatiile CORBA sunt compuse din obiecte, unitati individuale de software care combina functionalitate si date si care de multe ori (dar nu intotdeauna) reprezinta ceva in lumea reala. In general, exista mai multe instante ale unui singur tip de obiect - de exemplu, un website de vanzari online poate avea mai multe instante pentru obiectele de tip "cos de cumparaturi", toate identice ca functionare dar diferite in sensul ca fiecare dintre ele este asociat cu un anumite client si fiecare contine date referitoare la cumparaturile facute de acel client in particular. Pentru alte tipuri de obiecte, poate exista o singura instanta.

Pentru fiecare tip de obiect, asa cum este "cosul de cumparaturi" pe care l-am dat drept exemplu, trebuie definita o interfata in OMG IDL. Interfata este sintaxa pe care obiectul-server o ofera clientilor care o invoca. Orice client care vrea sa

invoce o operatie pe acel obiect trebuie sa foloseasca aceasta interfata IDL pentru a specifica ce interfata vrea sa efectueze si pentru a randi argumentele pe care le trimite. Atunci cand invocatia ajunge la obiectul cautat, aceeasi definire de interfata este folosita pentru a de-randui argumentele in asa fel incat obiectul poate sa efectueze operatia solicitata cu ele. Definirea interfetei este apoi utilizata pentru a randui rezultatele in vederea trimiterii lor inapoi, la destinatie.

Interfata IDL nu este dependenta de limbajul de programare, ea mapand toate limbajele de programare populare prin interfetele OMG: OMG are mapari standardizate pentru C, C++, Java, COBOL, Smalltalk, Ada, Lisp, Python si IDLscript.

Separarea interfetei de implementare, realizata de OMG IDL, este esenta standardului CORBA - cum se efectueaza interoperabilitatea, pastrand in acelasi timp transparenta. Interfata fiecarui obiect este definita foarte strict. In contrast, implementarea unui obiect - codul si datele sale - sunt ascunse restului sistemului (incapsulate), in afara unei limite pe care clientul nu o poate depasi. Clientii acceseaza obiectele doar prin interfata respectiva, invocand doar operatiile pe care acel obiect le expune prin interfata IDL, doar cu acei parametri (de intrare si de iesire) care sunt inclusi in apel.

Figura urmatoare arata cum, la nivelul unui singur proces, totul interactioneaza: se compileaza fiecare IDL in schelete de clienti si obiecte (dreapta) si se scrie obiectul (in stanga) si clientul lui. "Trunchiurile" si "scheletii" servesc pe post de proxy-uri pentru clienti si, respectiv, pentru servere. Deoarece IDL defineste interfetele atat de strict, trunchiul de pe partea clientului nu are nicio problema sa se lege perfect de scheletul de pe partea serverului, chiar daca cele doua sunt compilate pe limbaje de programare diferite, sau chiar ruleaza pe ORB-uri diferite de la producatori diferiti.

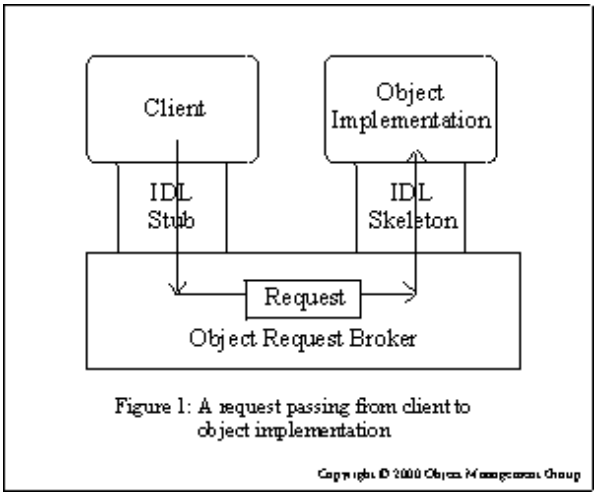


Figure 1: A request passing from client to object implementation

In CORBA, fiecare instanta a unui obiect are propria referinta unica la obiect - un "jeton" electronic folosit ca identificator. Clientii folosesc referintele la obiect pentru a directiona apelurile, identificand la ORB instanta exacta pe care vor sa o apeleze (asigurandu-se, de exemplu, ca ceea ce clientul vrea sa cumpere ajunge in propriul cos de cumparaturi, nu in cosul vecinului). Clientul "crede" ca apeleaza o operatie pe instanta obiectului, dar de fapt el apeleaza pe trunchiul IDL care apoi actioneaza ca un proxy. Trecand prin trunchi pe partea clientului, apelul continua prin ORB (Object Request Broker) si prin scheletul de pe partea de implementare, ajungand in final la obiect unde este executat.

In figura urmatoare este reprezentat un apel la distanta. Pentru a apela o instanta indepartata a obiectului, clientul trebuie mai intai sa obtina referinta obiectului sau. Pentru a efectua apelul, clientul foloseste acelasi cod ca in cazul apelului local descris mai sus, substituind obiectul referinta cu instanta indepartata. Atunci cand ORB-ul examineaza obiectul referinta si descopera ca obiectul cautat este indepartat, el ruteaza apelul prin retea spre ORB-ul obiectului indepartat.

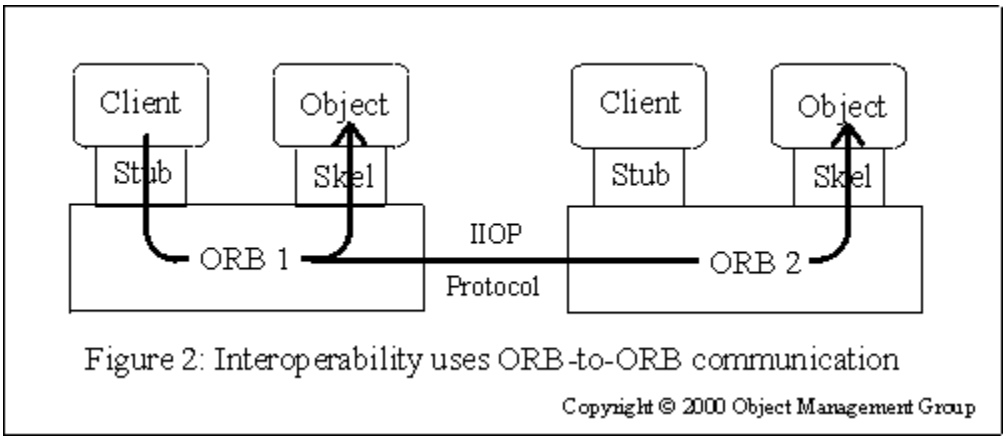


Figure 2: Interoperability uses ORB-to-ORB communication

OMG are acest proces standardizat la doua nivele cheie: mai intai, clientul cunoaste tipul de obiect pe care il apeleaza (obiectul - cos de cumparaturi, de exemplu) si trunchiul client si scheletul obiectului sunt generate din acelasi IDL. Acest lucru semnifica ca clientul cunoaste exact care operatii pot fi invocate, care sunt parametrii de intrare si unde trebuie sa ajunga acestia in apel. Atunci cand

apelul isi atinge destinatia, totul este la locul potrivit. Am vazut inainte cum reuseste OMG IDL sa realizeze acest lucru. In al doilea rand, ORB-ul clientului si cel al obiectului trebuie sa potriveasca din punct de vedere al protocolului comun - reprezentarea ce specifica obiectul destinatie, operatia, parametrii (intrare, iesire) fiecarui tip folosit si cum este reprezentat totul pe fire. OMG are aceste concepte definite deja - protocolul standard IIOP (OMG poate folosi si alte protocoale in afara de IIOP - dar in general IIOP se foloseste datorita avantajelor interoperabilitatii eficiente si pentru ca este pretins de OMG pentru conformitate). Desi ORB poate stabili din obiectul referinta daca obiectul destinatie este indepartat, clientul nu poate. In jetonul obiectului de referinta pe care clientul il detine in timpul apelului nu exista nimic ce poate fi folosit la identificarea locatiei obiectului destinatie. In acest mod se asigura transparenta localizarii - principiul CORBA care simplifica design-ul aplicatiilor ce folosesc obiecte distribuite.

5. TESTAREA

5.1 Metode de testare generale

In general, testarea componentelor de software se refera la a concepe activitati care duc la descoperirea erorilor de software si la validarea calitatii componentelor software,la nivel de element. Obiectivul principal este de a confirma calitatea componentei software, verificand functionalitatea,comportamentul si performatele acesteia.

Metodele de testare pot fi clasificate în doua mari categorii:

- Testarea black box. Aceasta abordare se concentreaza asupra intrarilor, iesirilor si functionalitatii modulelor software. Se mai numeste testare functională. Punctul de plecare este fie o specificatie, fie codul. În cazul codului, datele de test trebuie să permita verificarea functionalitătii programului. Testerul nu este interesat de modul în care este implementat programul respectiv, ci de modul în care acesta furnizeaza raspunsul dorit.

- Testarea white box. Aceasta abordare presupune inspectarea structurii codului modulelor software. Este cunoscuta si sub numele de testare structurala, presupune analizarea implementarii programului. Se verifica executarea corecta a tuturor cailor si ramurilor codului programului testat.

Obiectivele testarii pe element a unui modul sunt:

- Dezvoltarea unui test calitativ pentru a descoperii cat mai multe erori
- Aplicarea unor teste cu cost redus pentru a demonstra ca functionalitatea si comportamentul elementului corespund cerintelor.

Sunt patru sarcini majore in testarea componentei a unui modul:

1. Prima saracia este de a dezvolta si aplica testarea white-box pentru a valida logica interna, datele si structura programului bazat pe un model si un cod sursa dat.

2. De a construi si a aplica testarea black-box pentru a valida accesibilitatea functiilor externe, comportamentul lor si interfetele, bazat pe cerintele date.
3. Validarea si masurarea performantelor (ca de exemplu viteza de procesare, exactitatea etc).
4. Raportarea rezultatelor in urma testarii.

In ingineria software bazata pe componente atat producatorii, cat si utilizatorii trebuie sa efectueze testarea componentelor software. Astfel testarea componentelor va include doua tipuri de testari:

1. Testarea componentelor orientata catre producator(Vendor-oriented component testing) care apare ca un pas in procesul de dezvoltare al componentei. Se refera la activitatile de testare pe care le aplica producatorul pentru a verifica specificatiile componentei.
2. Testarea componentelor orientata catre utilizator (user-oriented component testing). Apare ca o parte a procesului de dezvoltare a softului bazat pe componente pentru un proiect specific. Testul se face intr-un context specific, pentru a vedea daca toate componentele software implicate furnizeaza functiile specificate si ating performatele dorite.

5.2 Testarea componentelor orientata catre producator

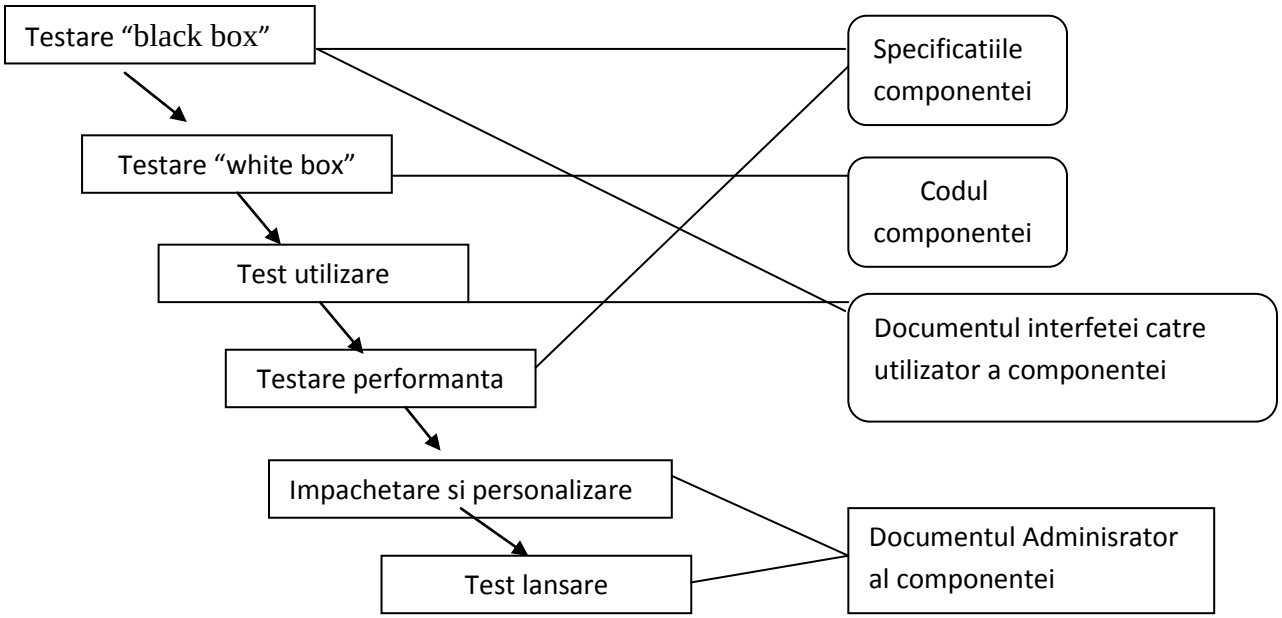
Inginerii de test ai unui producator de software implementeaza un proces de testare bazat pe niste modele de test bine definite, metode, strategii si criteria pentru a valida componentele softului dezvoltat.

Testarea componentelor de catre producator are trei mari obiective:

- Descoperirea a cat mai multe erori posibile
- Validarea interfetei componentelor, functiile, comportamentul si performanta pentru a se asigura ca sunt indeplinite specificatiile componentelor.
- Verificarea reutilizarii componentelor, impachetarea si desfasurarea pe platformele specifice si a mediului de operare.

Asa cum se vede si in figura, procesul de testare a unei componente pentru un producator consta in urmatorii 6 pasi:

1. Testul componentei “black box”: In primul pas, dezvoltatorii de componente si inginerii de test folosesc metode de test “black box” pentru a verifica functii incomplete sau eronate si comportamentul componentelor.



2. Testarea componentei “white box”: Acesta este al doilea pas al testarii componentei in care dezvoltatorii componentei folosesc metode de test “white box” pentru a descoperi erori interne in logica si structura programului, obiectelor de date si structurii de date.

3. Testarea de utilizare a componentei: Inginerii de test exerseaza diferite modele de utilizare prin interfetele componentei pentru a confirma ca functiile corecte si comportamentul asteptat sunt livrate.

4. Testarea de performanta: Inginerii de test si de asigurare a calitatii valideaza si evalueaza performanta componentelor.

5. Impachetare si customizare: Acest pas este folositor pentru componente care furnizeaza calitati de customizare. Obiectivul testarii sunt calitatiile de customizare integrare si implementarea unei abordari de impachetare.

6. Test de lansare: La ultimul pas al procesului de testare al componentei, se valideaza mecanismul de lansare a componentei pentru a se asigura ca este proiectat corect si implementat in concordanta cu un model de componenta.

5.3 Testarea componentelor orientata spre utilizator

Testarea componentelor orientata spre utilizator se refera la procesul de validare al unei componente si activitatile de testare care confirma calitatea componentelor software-ului implicat pentru un sistem specific. Inginerii implicati in testarea orientate spre utlizator efectueaza testarea de component pentru a indeplini urmatoarele obiective:

- Validarea functiilor si performantelor unei component refolosite pentru a se asigura ca sunt indeplinite cerinte specifice unui proiect si sistem.

- Confirmarea utilizarii adecvate si lansarea unei componente reutilizabile intr-o platforma specifica si un mediu de operare.

- Verificarea calitatii componentelor personalizate dezvoltate folosind componente reutilizate.

- Testarea calitatii de componente noi create pentru un proiect specific.

Validarea de componente noi proiectate este similara cu testarea componentelor orientata spre producator. Procesul de testare descries anterior poate fi folosit si aici.

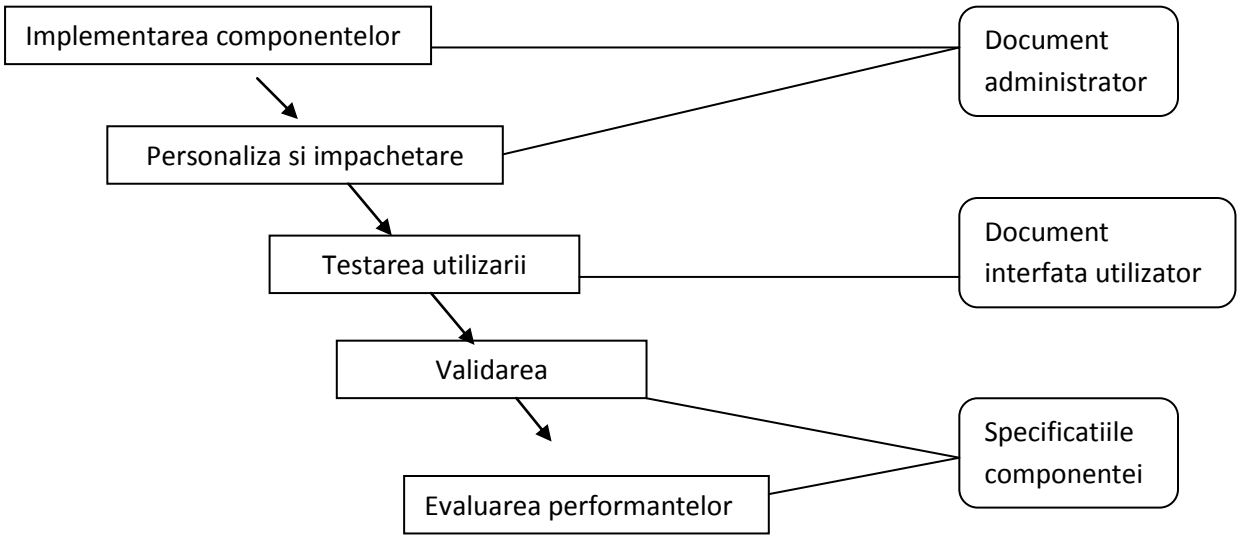
Cu toate astea, validarea componentelor refolosite si actualizate are diferite puncte de interes si limitari. De exemplu, un utilizator nu are de obicei acces la codul sursa al unei componente. Utilizatorul trebuie sa verifice

componentele refolosite folosind testarea “black box” fara a avea cunostinte despre ce se afla in interiorul ei. Acest lucru se intampla ca parte a unui proces de evaluare a componentelor intr-un stadiu incipient al procesului de dezvoltare de componente.

Figura alatura ilustreaza un proces de validare pentru componente complet reutilizate. Procesul include urmatoarele 5 stagii:

1) Implementarea componentelor: Acesta este primul pas al testarii orientate spre utilizator. Aici se verifica componenta pentru a vedea daca poate fi implementata cu succes in noul sau context si mediu de operare pentru un proiect.

2) Personalizarea si impachetarea compoentenlor: Acest pas verifica daca o componenta poate fi personalizata si impachetata cu succes folosind abilitatile integrate.



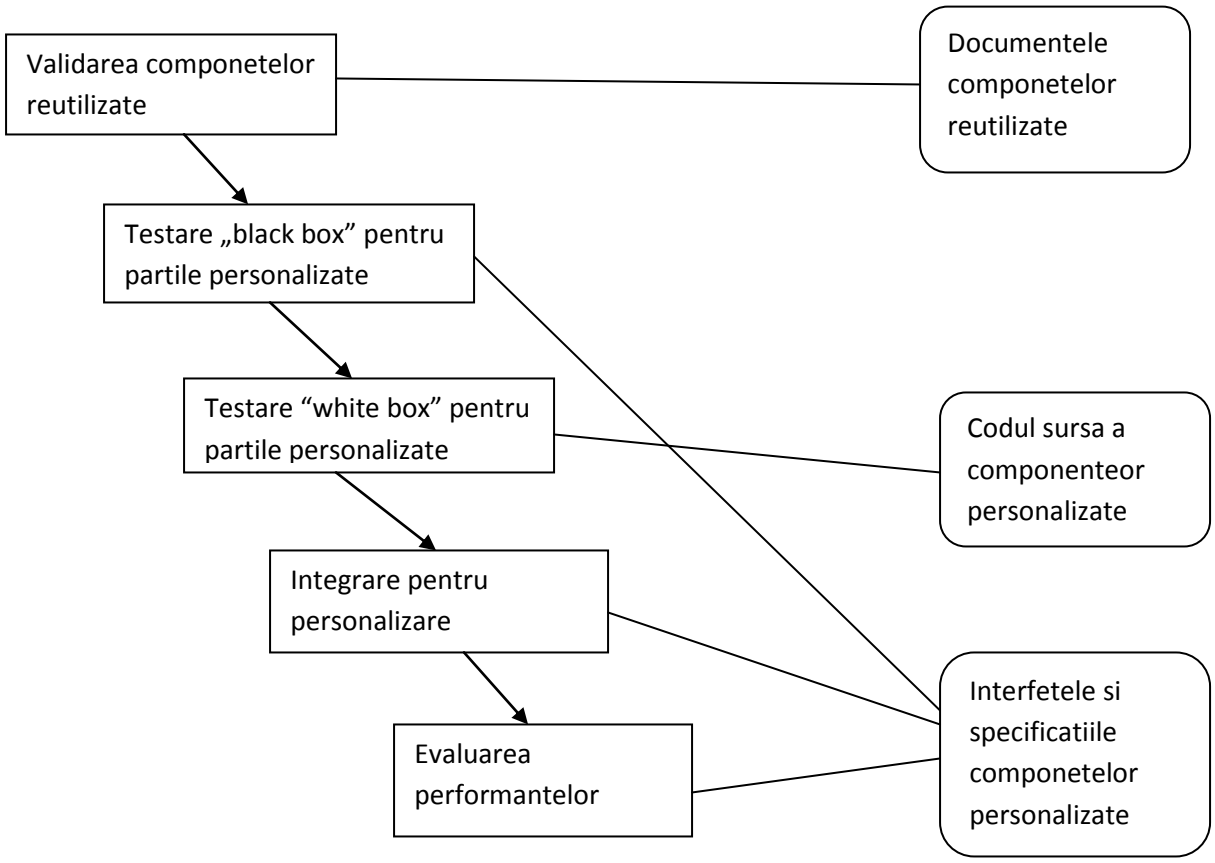
3) Testarea componentelor la utilizare: O component este testata pe baza unor modele de utilizare prin intermediul interfetelor. Scopul principal este de a acoperi modelul de folosire in noul context si mediu.

4) Validarea componentelor: Sarcina majora a acestui pas este de a efectua teste ”black-box” pe componente pentru a valida functiile si comportamentul componentelor in noul context si mediu refolosit.

5) Evaluarea performantelor: La acest pas se evalueaza si se masoara performanta unei componente intr-un context si mediu nou pentru a se asigura ca se satisfice cerintele de performanta.

5.4 Validarea componetelor personalizate

Validarea componentelor adaptate si actualizate este un alt pas major pentru componenta utilizatorilor. Aceste componente sunt cunoscute sub numele de “componente personalizate”. Ele sunt dezvoltate alterand si personificand componentele reutilizabile. Avand in vedere ca acestea contin componente reutilizabile si parti noi adaugate, pentru proiecte specifice sis item, procesul lor de validare difera de cel al comonentelor refolosite. In figura se arata procesul de validare al componentelor personalizate:



1. Validarea comonentelor reutilizate: Se valideaza componentele complet refolosite.
2. Testare “black box” pentru componentele personalizate: obiectivul la aceasta etapa este de a descoperii erorile de functionare si erori in comportamentul componentelor care au fost personalizate recent.
3. Testarea “white box” pentru componentele personalizate: Testarea “white box” se face pe un cod sursa dat, pentru a descoperii erori de logica si de structura a partilor personalizate.
4. Integrarea pentru personalizare: Compontele refolosite sunt integrate cu partile personalizate pentru a forma o component personalizata specifica, intr-un mediu nou.
5. Evaluarea performantelor: Trebuie evaluate performanta noilor component personalizate in noul context, pe baza specificatiilor.

Bibliografie

[1] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/
[2]Component-Based Software Engineering: Putting the Pieces Together – George T. Heineman , William T. Councill
[3]http://67.207.137.15/paper/filename/32/SOA_vs_Component_Based_Architecture.pdf
[4]http://repository.cmu.edu/cgi/viewcontent.cgi?article=1694&context=compsci&sei-redir=1&referer=http%3A%2F%2Fscholar.google.ro%2Fscholar_url%3Fhl%3Dro%26q%3Dhttp%3A%2F%2Frepository.cmu.edu%2Fcgi%2Fviewcontent.cgi%253Farticle%253D1694%2526context%253Dcompsci%26sa%3DX%26scisig%3DAAGBfm3DRaWCX9Yjk756Ay6aNZQFizC25A%26oi%3Dscholarr%26ei%3DZ03YULXPGYbYtAbdkIGYBg%26sqi%3D2%26ved%3D0CC0QgAMoATAA#search=%22http%3A%2F%2Frepository.cmu.edu%2Fcgi%2Fviewcontent.cgi%3Farticle%3D1694%26context%3Dcompsci%22
[5]http://www.cs.cmu.edu/afs/cs/project/able/ftp/acme-cascon97/acme-cascon97.pdf
Andrew Tanenbaum - Software Engineering

Component Testability and Component Testing Challenges:
http://www.diku.dk/OLD/undervisning/2005f/336/gao00.pdf

Component-Based White-Box Testing
http://www.it.iitb.ac.in/~prathabk/pages/tech_archives/cbtest/component_based_white_box_testing_presentation.pdf

Ian Sommerville – Software Engineering 8-th edition

http://www.omg.org/spec/

http://www.microsoft.com/com/default.mspx

http://www.polberger.se/components/read/com.html

http://www.javaworld.com/jw-10-1998/jw-10-beans.html?page=1

http://docs.oracle.com/javaee/5/tutorial/doc/javaeetutorial5.pdf - part IV - pagina 627