Dezvoltarea de software bazata pe componente

Prep. Iulian ÎNTORSUREANU,
Catedra de Informatica Economica, A.S.E. Bucuresti

Dezvoltarea bazata pe componente este o paradigma de dezvoltare a aplicatiilor software care-si propune atingerea unei productivitati sporite a procesului de dezvoltare printr-o reutilizabilitate de nivel superior. În material se face o definire a acestei abordari, urmata de descrierea elementelor de infrastructura necesare operabilitatii sale. Sunt analizate implicatiile adoptarii dezvoltarii bazate pe componente în procesul de dezvoltare a software-ului.

Cuvinte cheie: *software, proiectare, dezvoltare, componente.*

Introducere

Evolutia paradigmelor de concepere si realizare a aplicatiilor software a fost, într-o masura semnificativa, dirijata de necesitatea reutilizarii în proportie cât mai mare a codului deja realizat. Avantajele unei astfel de abordari sunt importante. Evitând "reinventarea rotii" se diminueaza, pe de o parte, efortul de realizare si testare a software-ului, pe de alta parte obtinând-se un plus de fiabilitate a produsului finit. În conditiile unei piete cu exces de cerere (cum este piata produselor software), astfel de avantaje nu sunt deloc neglijabile pentru dezvoltatori.

Abordarile precedente (programarea structurata, programarea orientata-obiect), desi aduc îmbunatatiri semnificative din acest punct de vedere (biblioteci de functii, respectiv biblioteci de clase si mecanismul de mostenire), nu rezolva în mod satisfacator problema reutilizabiltatii. O posibila explicatie este faptul ca aceste paradigme ofera o reutilizabilitate de nivel relativ scazut, respectiv elemente reutilizabile legate de activitatea de programare (interfata, structuri de date etc.) si nu de logica economica a aplicatiei (denumita si business logic).

O alta provocare pentru dezvoltatorii de software o reprezinta satisfacerea cerintelor de flexibilitate si scalabilitate a sistemelor software, în conditiile unor arhitecturi de calcul distribuite care tind sa devina norma, mergând pîna la utilizarea Internetului ca infrastructura de comunicatie.

Din perspectiva întreprinderilor, pentru care sistemele informatice bine concepute se concretizeaza în avantaje concurentiale, exista cerinte precum: implementarea rapida si eficienta a noilor functionalitati cerute de schimbari în mediul economic; extinderea sistemelor informatice spre Internet ca mediu de afaceri; integrarea pachetelor software comerciale cu aplicatiile proprii si mostenite.

Un raspuns la aceste probleme este *dezvoltarea de software pe baza de componente* (Component Based Development - CBD). Fundamentele acestei paradigme, derivata din POO, au aparut începînd cu anii 1996-1997. Interesul unor firme de prestigiu în acest domeniu (Computer Associates, Rational, Microsoft, Sterling etc.) dovedeste potentialul acestei noi abordari a proiectarii si dezvoltarii de software.

Ce este CBD ?

Dezvoltarea bazata pe componente (sau CBSE – Component Based Software Engineering) se poate defini ca o paradigma de realizare a produselor software orientata spre asamblarea aplicatiilor din componente software prefabricate. Componentele interoperabile pot fi realizate de catre dezvoltatorii de aplicatii, dar în principal CBD pleaca de la premisa ca aceste componente vor fi puse la dispozitia dezvoltatorilor prin intermediul unor cataloage specializate. Se apreciaza ca adoptarea pe scara larga a acestei paradigme va

duce la industrializarea dezvoltarii software-ului, aceasta tranzitie fiind echivalenta cu revolutia industriala din celelalte domenii economice, accelerând procesul de fabricatie si diminuând costurile [1].

CBD promite o serie de avantaje, pe de o parte intrinseci acestei paradigme, iar pe de alta parte derivate din reutilizabilitatea oferita de componente.

Avantaje independente de reutilizabilitate:

- Gestionarea mai buna a complexitatii: proiectele sunt mai bine focalizate pe aspectele esentiale, riscurile de nerealizare fiind reduse; aspectele de detaliu sunt încapsulate în componente;
- Reducerea efortului de dezvoltare, deoarece componentele sunt deja realizate; se poate reduce si efortul de mentenanta, care cade în responsabilitatea realizatorilor componentelor. Eforturile se vor concentra spre realizarea conexiunilor între componentele folosite;
- Flexibilitate – în conditiile existentei unor specificatii clare privind componentele, exista posibilitatea substituirii componentelor vechi cu unele noi, care ofera functionalitate sporita. O trasatura esentiala a CBD este separarea clara între specificatia unei componente si implementarea ei;
- Componentele care ruleaza în medii distribuite elimina necesitatea duplicarii codului.

Avantaje derivate din reutilizabilitate:

- Reducerea timpului de livrare, prin utilizarea unor componente deja realizate;
- Diminuarea necesitatilor de testare, componentele fiind deja testate si, eventual, certificate;
- Posibilitatea de a selecta cea mai avantajoasa solutie ca functionalitate si cost, datorita concurentei între furnizorii de componente;

Pentru a întelege modul în care aceasta abordare afecteaza procesul de realizare a software-ului este esentiala clarificarea notiunii de componenta software, precum si descrierea tehnologiilor care faciliteaza implementarea efectiva a aplicatiilor bazate pe componente.

În sens larg, o componenta software este o entitate software oferind o functionalitate bine stabilita, care poate fi livrata independent si asamblata cu alte entitati software pentru a forma aplicatii sau fragmente de aplicatii. O componenta software este determinata de trei aspecte [1]:

- *Specificatia* descrie latura semantica a componentei, respectiv serviciile pe care le ofera si modul de utilizare a acestora de catre un potential client. Cunoasterea specificatiei de catre un client îi este suficienta acestuia pentru a-i utiliza serviciile, fara a tine cont de modul cum acestea sunt efectiv realizate.
- *Proiectul de implementare* descrie modul în care este proiectata si construita componenta pentru a respecta specificatia. Implementarea tine de structurile de date utilizate si codul respectiv, si poate fi diferita ca limbaj de programare si platforma hardware de client.
- *Realizarea fizica a componentei*, respectiv executabilul corespunzator.

Astfel, încapsularea componentelor, adica separarea neta a specificatiei de implementare, permite o flexibilitate sporita în realizarea aplicatiilor prin posibilitatea de a substitui componentele care sunt conforme cu o anumita specificatie, dar difera din alte puncte de vedere (cost, functionalitate suplimentara etc.).

Un model conceptual mai detaliat al componentelor rezulta din abordarea lor din trei perspective, fiecare perspectiva fiind extinsa de urmatoarea [2]:

- *Perspectiva împachetarii* pune accentul pe reutilizare si apartine UML. O componenta este, din aceasta perspectiva, o entitate reutilizabila care împacheteaza fizic elemente de modelare (de exemplu, fisiere executabile, documente, biblioteci de subprograme, DLL, tabele s.a.).
- *Perspectiva serviciilor* evidentiaza calitatea componentelor de furnizori de servicii. Beneficiarii componentelor au acces la servicii prin intermediul interfetelor, care sunt comparabile cu contracte care

trebuie respectate de ambii parteneri (de exemplu, servicii ale sistemului de operare, biblioteci de functii, functii din cadrul API).

- *Perspectiva integritatii* permite definirea frontierei unei componente, conditie necesara pentru substituirea componentelor. Componentele sunt entitati software care mentin integritatea datelor cu care lucreaza, fiind independente de implementarea altor componente (de exemplu, sistemele de operare, cadre de aplicatii - frameworks, API-uri în ansamblu).

Specificatia unei componente este exprimata prin interfete, care reprezinta seturi de servicii grupate într-o maniera logica. Interfetele concretizeaza încapsularea, oferind clientilor modul concret de interactiune cu o componenta, clientul fiind izolat de aspectele de implementare. Se spune ca o componenta implementeaza una sau mai multe interfete.

Termenul de interfata este o extensie a abordarii obiectuale clasice, o interfata fiind asimilabila cu o clasa abstracta (clasele abstracte nu pot fi instantiate, operatiile fiind definite numai prin semnatura, ca functii virtuale pure, deci cu obligativitatea implementarii în clasele derivate). Limbajul Java implementeaza în mod explicit conceptul de interfata.

În consecinta, interfetele, ca si clasele, pot fi derivate. Semnificatia este însa usor diferita, în cazul interfetelor fiind vorba de mostenirea si reutilizarea specificatiilor, nu a codului, ca la derivarea claselor.

Pâna în prezent, utilizarea componentelor a luat amploare mai mult pe partea client a aplicatiilor. Mediile de programare actuale (de tip vizual) ofera facilitati în acest sens, începând cu componente necesare definirii interfetelor grafice ale aplicatiilor si mergând pâna la componente care mijlocesc accesul la servicii Internet, baze de date etc.

Pe de alta parte, definirea unor componente la nivel de server este mai dificila, trebuind sa tina seama de varietatea arhitecturilor de calcul pe care ele pot rula într-un caz concret. De asemenea, specificatiile unor astfel de componente sunt mai greu de specificat decât în cazul componentelor client.

Componentele care înglobeaza concepte sau procese de un nivel logic superior, din contextul economic al aplicatiei, sunt denumite si "business components". Ele ofera cu adevarat premisele unui nou salt în reutilizarea software.

Infrastructura necesara CBD [2], [8], [9], [10]

Functionarea unei aplicatii CBD necesita diverse forme de coordonare a activitatilor componentelor, care adesea ruleaza pe masini si platforme software diferite. Infrastructura (denumita si middleware) reprezinta setul de servicii pus la dispozitia componentelor pentru rezolvarea necesitatilor de coordonare ale acestora în contextul amintit. Componentele trebuie construite tinând seama de restrictiile impuse de infrastructura, fiind în schimb degrevate de implementarea serviciilor respective.

În mod esential, infrastructura trebuie sa ofere urmatoarele tipuri de servicii:

- *Împachetare*: componentele trebuie sa poata comunica infrastructurii elemente descriptive, precum serviciile oferite si semnatura operatiilor care ofera aceste servicii.
- *Distributie*: servicii care sunt responsabile de activarea/dezactivarea componentelor apelate, împachetarea parametrilor de apel etc. în contextul unei arhitecturi de calcul distribuite. Infrastructura trebuie sa izoleze o componenta de detaliile privind localizarea fizica a altor componente.
- *Securitate*: servicii care permit autentificarea sursei cererilor/raspunsurilor dintre componente si confidentialitatea conexiunilor prin care se efectueaza comunicatia.
- *Gestiunea tranzactiilor*: servicii care asigura evitarea aparitiei de inconsistente ale datelor în contextul unei interactiuni complexe între componente.
- *Comunicarea asincrona între componente,* prin mecanisme bazate pe cozi de mesaje.

Implementari ale infrastructurii

În prezent, "batalia" pentru impunerea unei infrastructuri se da între trei competitori: CORBA, proupusa de consortiul OMG (Object Management Group), COM+ propusa de Microsoft si tehnologia distribuita Java a firmei Sun.

CORBA (Common Object Request Broker Arhitecture) face parte dintr-o specificatie mai larga – OMA (Object Management Arhitecture) privind comunicatia inter-componente în mediu distribuit. Aspectele majore ale CORBA sunt:

– IDL (Interface Definition Language), care descrie modul de împachetare a informatiei în vederea accesarii prin interfete;

– Modelul de componente CORBA, care descrie modul în care componentele pot apela servicii ale altor componente;

– IIOP (Internet Interoperability Protocol), un protocol de comunicatie între diferitele implementari CORBA.

Dintre serviciile prevazute de CORBA sunt implementate: servicii privitoare la ciclul de viata al componentelor (creare/ distrugere), servicii de nume (permit identificarea si partajarea instantelor, servicii de securitate, servicii tranzactionale.

Infrastructura COM+ se bazeaza pe tehnologia Microsoft COM (Component Object Model). Aceasta a fost extinsa pentru medii distribuite sub forma DCOM (Distributed COM), care este compusa din:

– MIDL (Microsoft Interface Definition Language) care descrie modul de împachetare a informatiei în vederea accesarii prin interfete;

– COM care descrie modul în care componentele pot apela servicii ale altor componente;

– Extensii DCOM care permit accesul transparent la componente în medii distribuite

Pe baza acestor specificatii s-au construit aplicatiile:

– MTS (Microsoft Transaction Server) care ofera servicii de securitate si gestiunea tranzactiilor;

– MSMQ (Microsoft Message Queue) care ofera servicii de comunicatie asincrona între instante ale componentelor prin cozi de mesaje.

DCOM împreuna cu MTS si MSMQ sunt cunoscute sub denumirea de COM+ si reprezinta solutia Microsoft pentru aplicatii CBD, având dezavantajul dependentei de platformele Windows.

Limbajul Java porneste în competitie cu o serie de avantaje evidente: a fost proiectat pentru scrierea de aplicatii distribuite, include suport multi-threading, simplifica gestiunea memoriei si include în mod nativ conceptul de interfata. Java poate fi asimilat unei infrastructuri CBD daca luam în considerare tehnologiile conexe care suporta aplicatii CBD distribuite:

- *JavaBeans* reprezinta un model de componente pentru partea client, permitând utilizarea acestor componente în medii vizuale de dezvoltare
- *Remote Method Invocation* (RMI), care ofera modul de accesare de catre o clasa a operatiilor altor clase în mediu distribuit.
- *Java Naming and Directory Interface* (JNDI), care gestioneaza identificarea claselor în mediu distribuit

Celelalte servicii de infrastructura sunt prevazute prin specificatia Enterprise Java Beans (EJB) pentru componente pe partea de server.

Influente asupra procesului de realizare a software-ului

Paradigma dezvoltarii de software pe baza componentelor induce modificari inerente fata de abordarile traditionale în modul de proiectare a solutiilor informatice. Accentul se muta de la constructia aplicatiilor individuale catre realizarea unui portofoliu de componente care pot servi la asamblarea unei multitudini de aplicatii. Dintre aspectele care trebuie clarificate mentionam:

- dimensiunea si domeniul de functionalitate acoperit de o componenta;
- descrierea dependentelor dintre componente în cadrul unei aplicatii;
- publicarea caracteristicilor componentelor si moduri de evaluare a utilitatii lor într-un context dat;

- modul de efectuare a mentenantei unei aplicatii bazate pe componente si gestiunea evolutiei acestei aplicatii.

Rezulta ca o metodologie de analiza si proiectare orientata spre CBD trebuie sa permita relevarea proiectarii interfetelor si sa sprijine selectia, evaluarea si asamblarea componentelor pentru a realiza o aplicatie informatica. Astfel de metodologii pot beneficia de existenta unor instrumente de tip CASE, care permit gestiunea în maniera unitara a elementelor de modelare.

În general se apreciaza [2] ca orice abordare metodologica orientata spre CBD necesita, pentru transformarea cerintelor în solutii informatice, trei activitati importante: *analiza contextului*; *definirea arhitecturii*; *formularea solutiei* (figura 1).

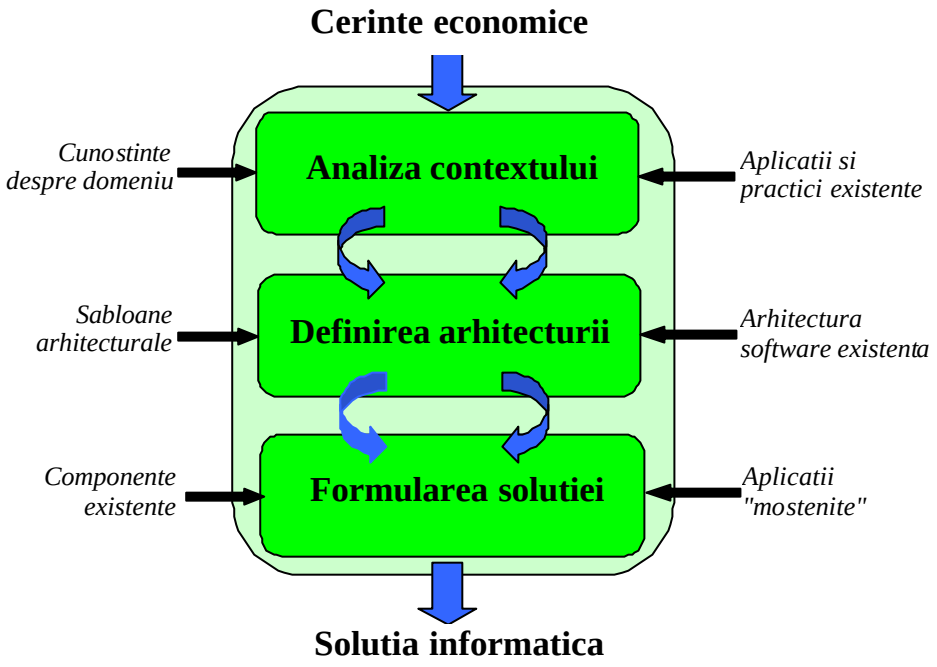

Fig. 1. Activitati CBD

Analiza contextului. Domeniul aplicatiei trebuie studiat prin prisma aspectelor statice si dinamice (de comportament) care pot fi modelate cu ajutorul tehnicilor de modelare a claselor, a cazurilor de utilizare (*use-cases*) si a diagramelor de secventa/ colaborare. Pentru tipurile abstracte identificate se poate continua cu specificarea atributelor si operatiilor, precum si formularea conditiilor anterioare si ulterioare operatiilor (tranzitiilor de stare), preferabil folosind un limbaj formalizat implementat de un instrument de modelare.

Definirea arhitecturii. În cadrul acestei activitati trebuie sa fie luate deciziile privind modul de alocare a comportamentului dorit al sistemului, identificat prin analiza, pe componente care pot fi apoi partajate de mai multe aplicatii si distribuite fizic. Pentru fiecare componenta se specifica setul de interfete care va trebui implementat. Este importanta modelarea dependentelor dintre componente (semnificând serviciile oferite de o componenta altor componente). Specificatia arhitecturala a aplicatiei este compusa din colectia de componente si dependentele dintre ele.

Formularea solutiei. Aplicatia urmeaza a fi realizata prin proiectarea detaliata si implementarea fiecarei componente, urmata de asamblarea componentelor într-un tot unitar, conform specificatiei arhitecturale. Aici se ia decizia tehnologiei (si, implicit, a infrastructurii) utilizate, putând fi influentata de factori externi proiectului (decizii la nivel de firma sau departament).

Deoarece interfetele sunt clar exprimate, dezvoltarea componentelor poate avea un caracter paralel. Adesea, în cursul implementarii au loc reconsiderari privind nevo-

ile utilizatorilor, arhitectura dorita a aplicatiei, fiind necesare schimbari ale specificatiilor componentelor si dependentelor dintre componente. Aceste schimbari trebuie documentate riguros, pentru mentine coerenta procesului de dezvoltare, necesitând o buna disciplina a membrilor echipei.

Limbajul Unificat de Modelare si CBD

Limbajul Unificat de Modelare (UML), rezultatul efortului de standardizare a unui larg consortiu de firme de software, ofera o notatie unanim acceptata pentru modelarea sistemelor si aplicatiilor orientate obiect. UML cuprinde, de asemenea, concepte si notatii adecvate dezvoltarii bazate pe componente.

În cadrul modelarii structurale, existenta interfetelor poate fi aratata în diagramele de clase în doua moduri: *notatia simpla* (“acadea”) si *notatia extinsa*, (folosind simbolul de clasa cu stereotipul *<<Interface>>*). Aceasta din urma permite afisarea operatiilor interfetei (figura 2).

Relatiile în care este implicata o interfata sunt:

- relatia de realizare (*realization*), care semnifica implementarea unei interfete de catre o clasa sau componenta;
- relatia de dependenta (*dependency*), care ilustreaza faptul ca o clasa utilizeaza operatii ale interfetei.

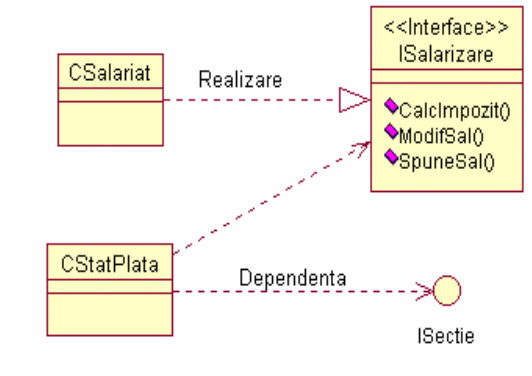

Fig. 2. Interfete în UML

În cazul modelarii arhitecturale, mai apropiate de implementare, componentele (în acceptiunea UML), întelese ca realizari fizice ale uneia sau mai multor clase, sunt desemnate printr-un simbol special. Interfetele si relatiile lor cu componentele sunt similare cu cele de la modelarea structurala. Diagramele de componente sunt utilizate pentru a vizualiza structura statica a elementelor fizice ale aplicatiei si relatiile dintre ele (figura 3). Diagramele de distributie (*deployment diagrams*) pot arata nodurile (procesoarele) pe care rezida componentele (figura 4).

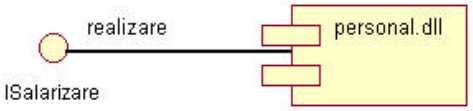

Fig. 3. Diagrama de componente

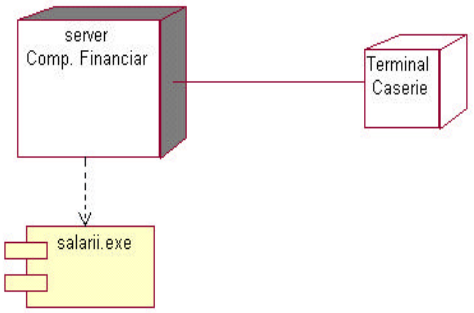

Fig. 4. Diagrama de distributie

În completarea unificarii notatiei prin UML exista preocupari pentru standardizarea unui format de reprezentare a modelelor, pentru a permite interoperabilitatea între diversele instrumente CASE si schimbul mai liber de informatie. Eforturile în aceasta directie s-au concretizat în standardul OMG XMI (XML Metadata Interchange) care ofera o modalitate de stocare si interschimb a modelelor, bazata pe XML. XMI este în prezent parte a standardului UML 1.3.

Metode orientate spre CBD

Pe lânga unificarea notatiei, a fost nevoie de instrumente mai puternice, care sa ofere specificatii metodologice pentru a defini procesul de dezvoltare în maniera consistenta, formalizata si repetabila. Astfel, au aparut mai multe metode (metodologii) de analiza si proiectare orientate spre CBD. Cele mai semnificative sunt Catalysis si Rational Unified Process.

Catalysis, creata de Desmond D’ Souza si Alan Cameron Wills, doreste sa ofere un

proces consistent pentru dezvoltarea, documentarea si integrarea componentelor, în vederea îndeplinirii cerintelor utilizatorilor sub forma unei solutii bazate pe componente. Accentul este pus pe precizia exprimarii elementelor solutiei, prin stabilirea unui vocabular de notiuni si notatii (pe baza UML), care poate fi extins prin mecanisme precis definite. Metoda urmareste surprinderea aspectelor conceptuale care apar la specificarea si proiectarea componentelor.

Baza conceptuala a metodei Catalysis este formata din patru concepte principale: **tipul**, utilizat în modelarea comportamentului unui set de obiecte din perspectiva externa aplicatiei; rafinarea si detalierea descrierii aplicatiei are loc prin mentionarea **conformantei** între tipuri; interactiunile dintre tipuri sunt modelate sub forma **colaborarilor**, care surprind actiunile unui set de obiecte apartinând diverselor tipuri, obiecte cu anumite roluri unul fata de celalalt; **cadrele** (*frameworks*), semnificând sabloane mai generale de structura si comportament, care pot fi concretizate (utilizate, instantiate) pentru cazuri reale.

Rational Unified Process

Propus de firma Rational ca metoda recomandata pentru lucrul cu UML, Rational Unified Process acopera complet ciclul de realizare a software-ului, în maniera orientata-obiect sau orientata pe componente. Obiectivul sau este obtinerea unui software de calitate, care îndeplineste nevoile utilizatorilor finali, mentinând sub control termenele si bugetul alocat proiectului. Metoda cuprinde si aspecte de conducere a proiectelor, oferind o abordare consistenta în privinta alocarii sarcinilor si responsabilitatilor în cadrul echipei/organizatiei [3]. Dintre caracteristicile metodei amintim: procesul se desfasoara în maniera iterativa si incrementala; dezvoltarea urmeaza cazurile de utilizare ca linie directoare, de la analiza cerintelor pâna la testarea produsului; procesul este configurabil, adaptabil nevoilor particulare ale fiecarei organizatii dezvoltatoare de software.

Paradigma dezvoltarii de software bazata pe componente se afirma din ce în ce mai mult în industria software ca abordare eficienta a problemei realizarii de aplicatii economice complexe, în mediu de rulare distribuit, care sa satisfaca exigente de flexibilitate si extensibilitate cerute de un mediu economic dinamic. Eficientizarea se face în principal pe baza unei reutilizari superioare, la nivelul logicii aplicatiei, tinzând spre dezideratul asamblarii aplicatiilor din componente preexistente. Cristalizarea unor modalitati de reprezentare unanim acceptate si a unor metode de analiza si proiectare corespunzatoare creeaza premisele unei adoptari pe scara larga a CBD.

Bibliografie

[1] Keith Short – *"Component Based Development and Object Modeling"* – Sterling Software, 1997

[2] Alan W. Brown – *"Building Systems from Pieces with Component-Based Software Engineering"* - Chapter 6 of *"Constructing Superior Software"*, Macmillan Technical Publishing, 1999

[3] Grady Booch, James Rumbaugh, Ivar Jacobson – *"The Unified Modeling Language User Guide"*, Addison-Wesley, 1998

[4] John Daniels – *"Objects and Components: the concepts"*, CBDEdge – www.sterling.com/CBDEdge

[5] Alan W. Brown, Kurt C. Wallnau – *"An examination of the current state of CBSE: A Report on the ICSE workshop on component-based software engineering (CBSE)"*, CBDi Forum

[6] Alan W. Brown, Balbir Barn – *"Enterprise-Scale CBD: Building Complex Computer Systems From Components"*, Sterling Software

[7] Alan W. Brown – *"Moving from Components to CBD"*, Sterling Software, 1999

[8] Valentin Cristea – *"CORBA si viitorul sistemelor distribuite"*, PC Report, 3/1999

[9] Pito Peter – *"Enterprise Java Beans"*, PC Report, Nov. 1999

[10] Alan W. Brown – *"Making Sense of the Middleware Muddle"*, CBDEdge