DCOM, CORBA, JAVA: STUDIU COMPARATIV ŞI ORIENTĂRI DE PERSPECTIVĂ

ing. Daniela Saru

Facultatea de Automatică şi Calculatoare

Universitatea Politehnica Bucureşti

Rezumat: Articolul conţine o prezentare succintă a tehnologiilor DCOM (cu tehnologiile corelate pe care le include, OLE, COM, protocol DCOM), CORBA şi a limbajului Java în contextul rolului pe care acestea îl au în elaborarea şi utilizarea aplicaţiilor software distribuite, actuale şi de perspectivă. De asemenea, sunt evidenţiate, prin comparaţie, caracteristicile de structură şi de funcţionalitate şi avantajele oferite, în funcţie de domeniul de aplicare, de către tehnologiile DCOM şi CORBA şi de către combinaţiile Java/CORBA şi ActiveX/DCOM.

Cuvinte cheie: sisteme distribuite eterogene, client, server, modularizare, orientare obiect, interfaţă, limbaj de definire a interfeţelor (IDL), RPC, OLE, COM, OCX, DCOM, ActiveX, CORBA, OMA, ORB, IIOP, *stub*, *skeleton*, *proxy*, WWW, Java, RMI, JNI, pico Java I.

1. Introducere

Tehnologia *OLE (Object Linking and Embedding)* a fost lansată de firma Microsoft în anul 1990 ca un instrument pentru sistemul de operare Windows, destinat realizării unor operaţii de tip *cut-and-paste* (selectare, copiere, mutare a unor entităţi). Ulterior, a fost extinsă *(OLE2)* cu facilităţi care să permită comunicarea între aplicaţiile Windows, devenind astfel posibilă crearea unor documente compuse (de exemplu, includerea unei foi de calcul Excel în cadrul unui document de tip Word), mutarea componentelor document dintr-o fereastră în alta *(drag-and-drop)* şi automatizarea (posibilitatea de coordonare a aplicaţiilor care realizează sarcini complexe şi de descriere a succesiunii operaţiilor ce urmează a fi executate - *scripting*).

Modelul de realizare a comunicaţiei între aplicaţii în cadrul tehnologiei OLE2 a primit numele de *COM (Component Object Model)*, folosit uneori şi cu semnificaţia de *Common Object Model*[1]. COM este proiectat pentru a lucra pe o singură maşină "gazdă" şi numai cu obiecte COM (numite şi obiecte OLE sau obiecte Windows). Pentru definirea interfeţelor asociate obiectelor, se foloseşte un limbaj elaborat de către firma Microsoft *(MIDL - Microsoft's Interface Definition Language)* care are la bază limbajul DCE IDL (OSF). O interfaţă grupează mai multe funcţii corelate, iar un obiect reprezintă o instanţiere a unei clase, care implementează una sau mai multe interfeţe şi care este caracterizată de un identificator unic. Deoarece nu pot fi create obiecte cu identificator unic, un client care doreşte să comunice cu un anumit obiect realizează acest lucru prin intermediul unui pointer care îi permite accesul la funcţiile implementate în cadrul interfeţei asociate. Din acest motiv, la momente diferite de timp, un client nu se poate reconecta la o aceeaşi instanţă obiect (cu aceeaşi informaţie de stare), ci numai la un pointer către o interfaţă a aceleiaşi clase. COM cuprinde şi un sistem de gestiune a numelor de fişiere *(Monikers)* care permite clienţilor să încarce/să salveze starea aplicaţiilor din/în memoria nevolatilă (fişiere disc).

Automatizarea OLE este o extensie a COM [2]. Interfeţele compatibile cu acestă extensie se pot descrie cu ajutorul unei extensii a MIDL care reprezintă limbajul de definire a obiectelor *(ODL - Microsoft's Object Definition Language)*. Aceste interfeţe pot fi înregistrate (opţional) în cadrul unei biblioteci de tipuri *(Type Library)* din care se pot obţine informaţii la momentul execuţiei în mod similar funcţionării "depozitului" de interfeţe (IR) din arhitectura CORBA [3]. Prin folosirea automatizării OLE, se pot crea aplicaţii care să ofere obiecte automatizate *(server-e automatizate)* unor instrumente de tip *script* şi macrolimbajelor. Instrumentele şi programele client, care accesează astfel de server-e, se numesc *controller-e automatizate*. Interfeţele OLE automatizate pot fi invocate dinamic de către un client care nu le cunoştea în momentul compilării. Pentru aceasta este necesar ca server-ul automatizat să ofere suport pentru o interfaţă specială (**IDispatch**) şi să se folosească biblioteca de tipuri.

Aşa-numitele *controale OLE (OLE Controls)*, denumite abreviat şi *OCXs* (după extensia de fişier MS-DOS pe care o utilizează) reprezintă tot o extensie a infrastructurii COM [2]. Un control OLE este un server (implementat într-un limbaj de programare ca C++, Visual Basic etc.) care utilizează pentru comunicarea cu aplicaţiile apelante proprietăţi, metode şi evenimente. Un eveniment reprezintă o notificare emisă de un control OLE ca răspuns la o acţiune a utilizatorului (de exemplu, acţionarea *mouse*-ului) care poate determina modificarea acţiunii controlului. Controlul OLE nu poate furniza implementarea unui eveniment, ci doar comunică apariţia lui şi apelează implementarea furnizată de către o altă aplicaţie. Setul de evenimente propriu unui control OLE se defineşte ca set de interfeţe ODL. În esenţă, un control OLE este un obiect OLE cu facilităţi suplimentare [4]: persistenţă, includere în

alte componente, automatizare, interfeţe de ieşire etc.

Către sfârşitul anului 1995, Microsoft a lansat o extensie a OCXs, numită *ActiveX*, care cuprinde un set de tehnologii (COM/DCOM/OLE/OCX), create pe baza controalelor OLE şi destinate facilitării dezvoltării de aplicaţii în Internet [1][2]. O aplicaţie care este un control ActiveX poate fi inclusă în pagini Web sau într-un container ActiveX de tip VisualBasic, Visual C++ sau Microsoft Access.

DCOM (Distributed COM) este denumirea dată ansamblului modelului COM, îmbogăţit cu un protocol care să permită crearea de aplicaţii distribuite eterogene [1]. Protocolul DCOM este un protocol la nivel aplicaţie, destinat realizării apelurilor de proceduri la distanţă, care acţionează asupra obiectelor software. Acesta are la bază specificaţia DCE RPC, fiind un exemplu de instrument folosit pentru construirea unui mediu software, orientat obiect "deasupra" facilităţilor oferite de către standardul DCE [5]. În prezent, DCOM face parte din sistemele de operare Windows 95 şi Windows NT.

(programare), arhitectura apelurilor la distanţă şi arhitectura protocolului DCOM.

Tehnologia ***CORBA (Common Object Request Broker Architecture)*** este o specificaţie standardizată, elaborată într-o primă variantă în anul 1991 de către OMG *(Object Management Group)*, pe baza OMA *(Object Management Arhitecture)* [3] şi urmată, până în prezent, de alte câteva versiuni îmbogăţite substanţial din punctul de vedere al facilităţilor oferite. Scopul creării OMA este ca funcţionalitatea de bază a aplicaţiilor să fie furnizată prin intermediul unei interfeţe standard, ceea ce face posibilă atât existenţa mai multor implementări ale aceleiaşi funcţionalităţi (care pot să difere din punctul de vedere al performanţei, preţului sau adaptării la diferite platforme specializate), cât şi crearea unor componente specializate, care să folosească această funcţionalitate prin intermediul interfeţei standard.

În CORBA, interfaţa asociată unui obiect se defineşte cu ajutorul unui limbaj specializat, *IDL (Interface Definition Language)*. Interfaţa cuprinde specificarea tuturor operaţiilor ce vor putea fi realizate de către obiect, împreună cu parametrii de intrare şi de ieşire de care au nevoie, şi cu

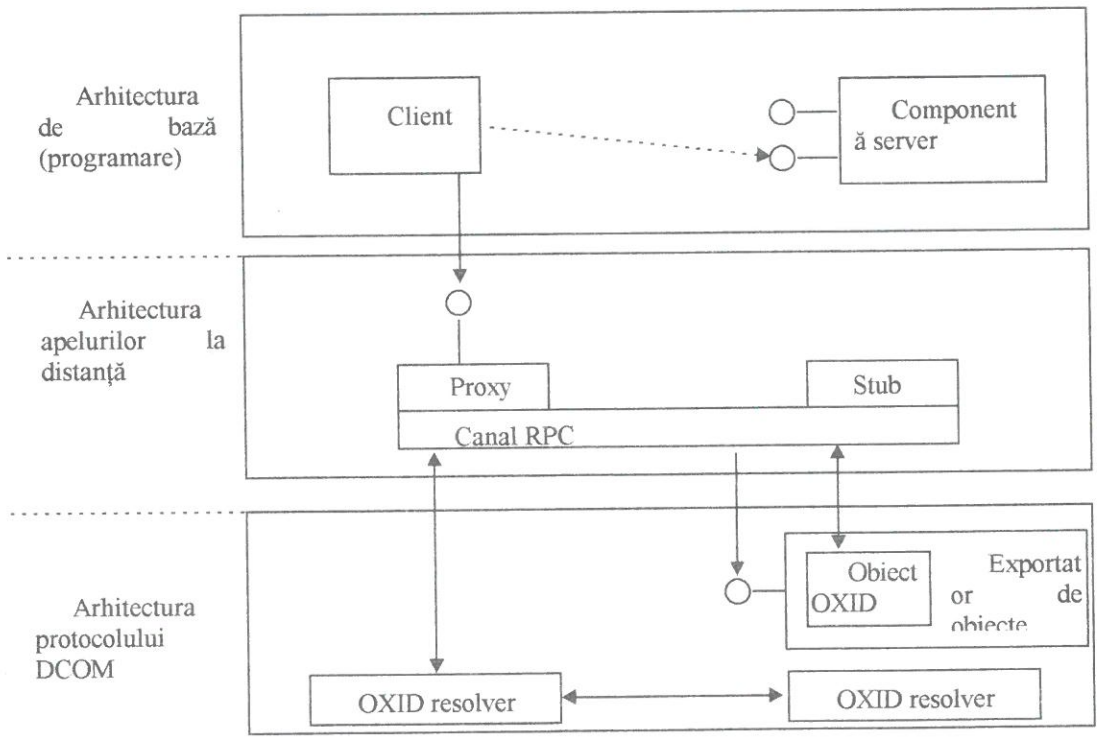

Figura 1. Structura DCOM

În ansamblu, DCOM include toate extensiile COM prezentate anterior, fiind o arhitectură care permite crearea de aplicaţii distribuite pe baza unor componente software binare (existente sau nou create). Structura DCOM cuprinde în esenţă trei niveluri [6] (figura 1): arhitectura de bază

excepţiile care pot fi generate. Arhitectura CORBA separă interfaţa scrisă în OMG IDL, de implementarea care poate fi realizată cu ajutorul unui limbaj de programare oarecare. Separarea cele două implementări (client şi obiect folosit) asigură un grad ridicat de flexibilitate şi se

realizează, practic, prin intermediul a cel puţin trei componente: un *stub IDL* asociat clientului, un *ORB* (sau mai multe dacă este necesară interacţiunea cu alte sisteme) şi un *skeleton IDL* asociat implementării obiectului folosit (figura 2).

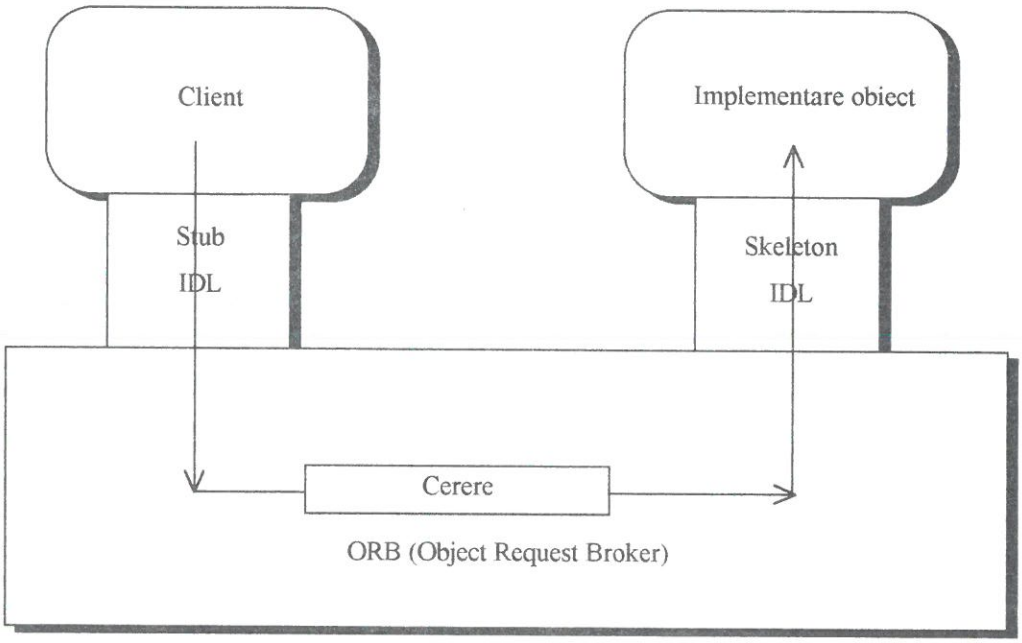

Figura 2. Transmiterea unei cereri de la client c`tre implementarea obiectului

Atât *stub*-ul, cât şi *skeleton*-ul se conectează, pe de o parte, la ORB şi, pe de altă parte, la implementarea asociată (scrisă cu ajutorul unui anumit limbaj de programare) şi sunt generate automat în urma compilării interfeţei IDL. Mecanismul de transmitere a unei cereri de la un obiect client către un obiect server se caracterizează prin următoarele aspecte [3]:

- atât implementarea client, cât şi implementarea server sunt izolate prin intermediul interfeţelor IDL faţă de ORB, clienţii neavând acces la detaliile de implementare ale server-elor pe care le folosesc; aceasta permite folosirea unor implementări diferite pentru o aceeaşi interfaţă şi utilizarea unor limbaje de programare diferite pentru implementarea diverselor componente ale unei aplicaţii;
- toate cererile sunt gestionate de către ORB, forma de invocare fiind aceeaşi pentru apeluri locale sau la distanţă *(remote)*, detaliile legate de distribuţia componentelor neconstituind procuparea programatorului de aplicaţie.

ORB (Object Request Broker) reprezintă componenta de bază a arhitecturii (CORBA), având ca sarcină principală asigurarea infrastructurii de comunicaţie, necesară accesului transparent la obiecte. ORB permite utilizatorului să invoce operaţii asociate obiectelor fără ca acesta să se preocupe de localizarea obiectelor în reţea, de tipul hardware-ului pe care se execută, de sistemul de operare, de limbajele cu ajutorul cărora sunt implementate, de deosebirile legate de modul de reprezentare a datelor sau protocolurile de reţea utilizate. Standardul CORBA specifică atât funcţiile ce trebuie îndeplinite de ORB, cât şi un set de interfeţe standard pentru aceste funcţii.

Celelate două mari categorii de componente ale arhitecturii OMA (CORBA) sunt *serviciile CORBA* [3][9] şi *facilităţile CORBA* (*orizontale* şi *verticale*), pentru ultima dintre acestea neexistând încă specificaţii standard finalizate [7]. Deoarece într-o arhitectură OMA fiecare client şi fiecare obiect pot comunica numai prin intermediul unuia sau a mai multor ORB-uri, fiecare serviciu CORBA este accesibil fiecărui obiect din sistem, OMA nefiind o arhitectură stratificată, caracteristică ilustrată sugestiv în figura 3.

Interoperabilitatea tuturor obiectelor CORBA este garantată prin folosirea *referinţei la obiect* (rezolvă problema localizării obiectului) şi a limbajului IDL (rezolvă problema "traducerii" invocărilor între clienţi şi obiectele invocate). *Referinţa la obiect (object reference)* se asociază fiecărui obiect CORBA (de către ORB) în momentul creării şi are semnificaţie atâta timp cât obiectul nu este desfiinţat în mod explicit. Clientul poate obţine şi poate folosi în invocarea obiectului referinţa corespunzătoare, această asociere permiţând ORB să direcţioneze cererea către obiectul ţintă. Referinţa la obiect nu este o simplă

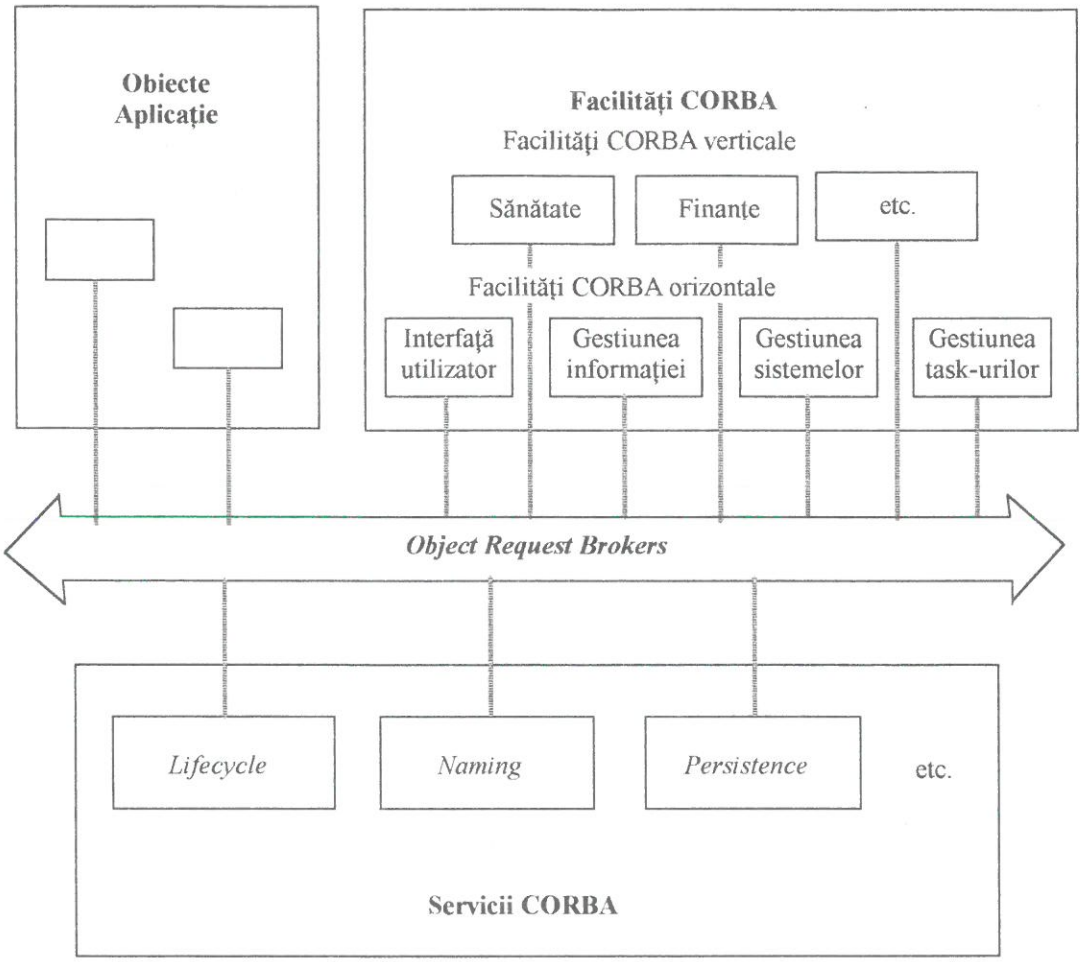

Figura 3. Structura OMA (Object Management Architecture)

adresă de memorie sau de reţea prin care se specifică un obiect; OMA conţine anumite cerinţe referitoare la validitatea unei astfel de referinţe. De exemplu, în cazul în care clientul memorează referinţa unui anumit obiect în cadrul unui fişier sau bază de date şi, ulterior, încearcă să o folosească, invocarea obiectului trebuie să se încheie cu succes chiar dacă între timp obiectul ţintă a fost mutat (dar nu a fost desfiinţat). Referinţa la obiect are un rol cheie în interacţiunea dintre utilizator şi resurse: este "înţeleasă" de către orice tip de ORB şi poate fi transmisă oriunde în cadrul sistemului distribuit prin intermediul bazelor de date, a serviciilor specializate (*Naming*, *Trading*) etc.

Comunicarea între ORB-uri se poate realiza fie *direct* (dacă toate ORB-urile implicate utilizează acelaşi protocol), fie cu ajutorul unor *punţi (bridges)* cu rol de "traducător" între diferite protocoluri (în caz contrar) [8]. Prima variantă (comunicarea directă) reprezintă o soluţie simplă, uşor de administrat şi eficientă, fiecare ORB folosind protocolul standard IIOP *(Internet Inter-ORB Protocol)* [3][8], DCE CIOP *(DCE Common Inter-ORB Protocol)* [3] sau un protocol proprietate. Cea de-a doua variantă (*bridging* - folosirea punţilor de comunicare) este mai complexă, dar asigură interoperatibilitate transparentă între domenii diferite. De obicei, la graniţa dintre aceste domenii există deja punţi hardware, care oferă o locaţie firească pentru punţile software CORBA 2.0.

Limbajul Java constituie, în prezent, cel mai larg acceptat şi mai promiţător reprezentant al tehnologiei destinate programelor mobile. Creat în cadrul firmei *Sun Microsystems*, Java a fost destinat iniţial dispozitivelor inteligente ca mecanism comun prin care să facă schimb de informaţii despre şi, respectiv, să-şi poată influenţa reciproc comportarea. De exemplu [10], un dispozitiv inteligent de telecomandă, echipat cu cunoştinţe despre un anumit televizor şi despre comenzile pe care i le poate transmite poate fi folosit şi pentru controlul altor echipamente inteligente (combină muzicală, cuptor cu microunde etc.), dacă acestea "descarcă" *applet*-uri Java corespunzătoare în dispozitivul de telecomandă.

Creşterea aproape explozivă din ultimii câţiva ani a numărului de utilizatori conectaţi prin

Internet a avut drept catalizator *WWW - World Wide Web*. Această tehnologie, deosebit de utilă pentru căutarea şi folosirea informaţiilor aflate "la distanţă" în raport cu clientul, nu asigură însă şi facilităţi de calcul distribuit. Extinderea *browser*-elor Web cu ajutorul *tehnologiei destinate programelor mobile (agenţilor mobili)* permite descărcarea dinamică a acestui tip de programe din orice server în cadrul *browser*-ului, care devine astfel o unealtă de tip interfaţă utilizator de uz general pentru clientul (program aplicaţie) descărcat. Ţinând cont de faptul că tehnologia Web permite existenţa unor clienţi Web care se execută pe o mare diversitate de platforme (sub sisteme de operare de tip Windows, OS/2, MacOS, UNIX, în sisteme de calcul tradiţionale sau chiar incluse în televizoare, telefoane mobile etc.), pentru a fi viabilă, tehnologia destinată programelor mobile trebuie să poată fi folosită în toate aceste arhitecturi.

Decizia firmei Sun de a aplica tehnologia Java în cadrul *browser*- elor şi server-elor Web constituie un exemplu de apariţie a celui mai bun instrument la timpul potrivit. Deoarece în ultimul timp au apărut numeroase lucrări editate în limba română dedicate limbajului Java, în cadrul articolului se vor considera cunoscute caracteristicile limbajului menţionat, evidenţiindu-se numai aspectele cele mai importante din punctul de vedere al studiului comparativ propus.

2. Studiu comparativ: DCOM, CORBA

CORBA şi DCOM sunt două arhitecturi software care permit crearea şi integrarea aplicaţiilor în mediile distribuite heterogene. Ambele modele sunt orientate obiect, lucrează cu limbaje de definire a interfeţelor care folosesc ca bază limbajul C++ (IDL OMG, respectiv MIDL creat ca extensie orientată obiect a IDL DCE (OSF)) şi preiau responsabilitatea realizării comunicaţiei între aplicaţii din sarcina programatorilor sau a utilizatorilor, astfel încât aceştia nu mai sunt preocupaţi de probleme ca situarea în reţea, platforma hardware, sistemul de operare, limbajul folosit pentru implementare (de exemplu, formatul datelor sau convenţiile de apelare), protocoluri în reţea etc.

CORBA are asociată o specificaţie standardizată, elaborată în cadrul OMG (într-o primă variantă în anul 1991) prin contribuţia celor peste 750 de membri, acceptată deja ca bază pentru implementarea produselor de un număr mare de firme software, ceea ce permite utilizatorilor să creeze aplicaţii care să înglobeze componente interoperabile, achiziţionate de la producători diferiţi, şi care funcţionează sub sisteme de operare dintre cele mai diverse (MS-DOS, Windows 3.x, Windows 95, Windows NT, aproape toate versiunile UNIX, OS/2, OS/400, MacOS, VME, MVS, VMS şi multe sisteme de operare în timp real [1]). DCOM este o arhitectură elaborată de firma Microsoft, pentru care implemetarea a precedat elaborarea specificaţiei standard (invers decât în cazul CORBA) şi a cărei acceptare de către alte firme este încă destul de limitată (specificaţia care include şi protocolul DCOM a fost publicată în cursul anului 1996). DCOM are însă avantajul includerii în sistemele de operare Windows 95 şi Windows NT care "populează" numărul impresionant de calculatoare personale, utilizate în prezent. De altfel, în lista sistemelor de operare care permit utilizarea arhitecturii DCOM figurează doar cele două menţionate anterior, umând a fi incluse UNIX şi MVS [1][6].

Din punctul de vedere al limbajelor de programare cu ajutorul cărora pot fi implementate componentele aplicaţiilor, în timp ce CORBA dovedeşte o mare flexibilitate, incluzând specificaţii standard pentru C, C++, Ada, Smalltalk şi COBOL şi având în curs de elaborare standardul pentru Java (pentru care există deja implementări), DCOM rămâne încă strâns legat de modelul limbajului C++ (acces prin pointeri), având deocamdată anumite probleme în folosirea unor limbaje ca Java sau COBOL în care acest mecanism nu poate fi utilizat [1].

CORBA oferă utilizatorilor o paletă largă de servicii modularizate (*Naming, Event, LifeCycle, Transactions, Security, Properties, Persistence* etc.) în timp ce Microsoft poate implementa un asemenea nivel în cadrul platformelor non Windows doar prin translatarea întregului produs DCOM (ActiveX). În plus, datorită componentelor "moştenite" prin implementarea tehnologiei destinate gestionării documentelor compuse (OLE, OLE2), DCOM conţine o interfaţă de programare a aplicaţiilor (API) mult mai complexă decât cea necesară majorităţii aplicaţiilor distribuite care nu includ astfel de prelucrări [11].

Din punctul de vedere al facilităţilor de creare a documentelor compuse în cadrul aplicaţiilor distribuite în sisteme heterogene, ambele arhitecturi oferă alternative valoroase, cu tradiţie în cazul DCOM (OLE) sau mai noi (*OpenDoc* [2], adoptat ca standard pentru facilitatea CORBA orizontală corespondentă - *Distributed Document Component Facility, Compound Presentation & Interchange* [12]). Realizarea unei comparaţii pertinente între cele două arhitecturi cu referire la acest aspect implică prezentarea unui volum mare de informaţii şi ar putea constitui, eventual, materialul de bază al unei alt articol decât cel de faţă.

Între cele două arhitecturi care fac obiectul studiului elaborat în cadrul acestei secţiuni există şi alte asemănări şi deosebiri care vor fi prezentate în

continuare, grupate în cadrul a trei categorii distincte, asociate cu:

- *arhitectura de bază (programare)* - partea "vizibilă" pentru programatorii obiectelor (aplicaţiilor) client şi server;
- *arhitectura apelurilor la distanţă* - realizează în mod transparent pentru utilizatori comunicarea între procese în cadrul aceleiaşi maşini gazdă (lucrează cu pointeri la interfeţe sau referinţe la obiecte);
- *arhitectura protocolului de legătură* - extinde arhitectura apelurilor la distanţă astfel încât acestea să se poată realiza între procese ce se execută pe maşini distincte.

La nivelul *arhitecturii de bază (programare)*, comparaţia între cele două arhitecturi evidenţiază următoarele asemănări, respectiv deosebiri:

- ambele arhitecturi încapsulează funcţionalitatea server-ului în obiecte a căror interfaţă este descrisă cu ajutorul unui limbaj de tip IDL orientat obiect; server-ul poate fi folosit de către client numai prin intermediul metodelor specificate în interfaţă, implementarea server-ului fiind "ascunsă" clientului;
- la nivelul IDL se pot folosi mecanisme specifice limbajelor orientate obiect: încapsularea datelor, polimorfism, moştenire simplă; în CORBA este permisă utilizarea moştenirii multiple şi se pot specifica excepţii; DCOM nu utilizează moştenirea multiplă, dar permite definirea de obiecte cu mai multe interfeţe;
- codul IDL care conţine descrierea interfeţei (interfeţelor) se compilează, generându-se astfel atât codul *proxy/stub* (DCOM) sau *stub/skeleton* (CORBA), cât şi fişierul *header* asociat interfeţei, acestea urmând a fi folosite de către server şi de către client; în biblioteca DCOM există obiecte *proxy* şi *stub* deja create pentru toate interfeţele DCOM şi OLE [13], motiv pentru care atât cunoaşterea şi folosirea limbajului de definire a interfeţelor (MIDL) cât şi parcurgerea etapei de compilare sunt obligatorii numai pentru programatorii care doresc să creeze ei înşişi anumite interfeţe noi;
- în cadrul etapei de compilare, DCOM asociază fiecărei interfeţe şi fiecărei clase obiect câte un identificator unic [13], prin execuţia unei funcţii apelate explicit de către utilizator, în timp ce CORBA permite utilizatorilor să identifice interfeţele prin nume al căror domeniu de valabilitate este bine stabilit [3]; în plus, CORBA permite folosirea câte unui identificator unic pentru fiecare din interfeţele cuprinse în cadrul "depozitului" de interfeţe (IR); acesta este generat, de obicei, pe baza numelui menţionat anterior şi a unui prefix de localizare, dar poate fi specificat şi de către utilizator (sunt permise inclusiv formate UUID DCE); din acest punct de vedere, CORBA prezintă avantajul de a nu obliga utilizatorii să folosească, în mod explicit, o funcţie pentru a asigura identificarea în mod unic a interfeţelor [4];
- la fel ca şi în CORBA, operaţiile ce pot fi definite în cadrul interfeţelor DCOM pot avea argumente de tip **in**, **out** sau **inout**, dar rezultatul acetora poate fi exprimat doar într-o formă *(HRESULT)* bine determinată şi parţial "proprietară" firmei Microsoft [4][2][5]: 32 de biţi, dintre care un bit succes/insucces, un câmp de 15 biţi ce poate fi definit numai de către firma Microsoft şi restul de 16 biţi care pot codifica alte informaţii; în CORBA valoarea rezultatului operaţiilor poate fi de orice tip permis şi se pot specifica excepţii pentru tratarea situaţiilor deosebite; de asemenea, operaţiile pot fi declarate ca unidirecţionale (**one-way**) şi pot folosi un argument opţional pentru contextul de realizare. În cadrul specificaţiei CORBA este inclus şi un set cuprinzător de excepţii sistem tipice;
- din punctul de vedere al metodologiei de creare a obiectelor, în DCOM există precizate anumite reguli [13] care asigură portabilitatea codului server, cerinţă care în CORBA va fi îndeplinită (cu referire la un acelaşi server executat cu produse software ce aparţin unor producători diferiţi) abia după definitivarea standardizării adaptorului de obiecte portabile *(POA Portable Object Adapter)* [14];
- implementarea server-ului permite în ambele arhitecturi auto-gestionarea duratei de viaţă; server-ul DCOM creează un eveniment pe care aşteaptă până în momentul în care sunt "şterse" contoarele de referinţe asociate tuturor obiectelor pe care le conţine, după care se dezactivează; server-ul CORBA creează o instanţă a clasei pe care o reprezintă şi se blochează pe **impl_is_ready**()[3] în aşteptarea invocărilor ce pot sosi din partea clienţilor. Dacă pe o durată precizată de timp acestea nu apar, server-

ul se dezactivează. Durata de timp poate fi stabilită de către programator;

- atât în CORBA cât şi în DCOM, înainte de lansarea în execuţie a aplicaţiilor astfel create este necesară înregistrarea procesului server (stabilirea corespondenţei între numele interfeţei şi numele fişierului executabil care o conţine); în plus, DCOM impune înregistrarea similară a server-ului corespunzător codului *proxy/stub* (care sunt chiar componente DCOM) în formă de bibliotecă DLL *(Dynamic Link Lybrary)*;
- în ambele arhitecturi există facilităţi de invocare dinamică care permit clienţilor să folosească server-e despre care nu existau informaţii la momentul compilării;
- ambele arhitecturi asigură server-ului posibilitatea de a obţine informaţii de la clientul său în timpul executării unei operaţii sau de a-i trimite notificări sau cereri fără dezavantajele oferite de soluţia implementării clientului însuşi ca server: în DCOM se foloseşte tehnica *obiectelor conectabile* [13] care impune implementarea unor interfeţe specifice în cadrul server-ului, în timp ce în CORBA acest aspect constituie sarcina unui serviciu specializat *(Event)*, fără implicaţii asupra interfeţelor definite de către utilizator.

Nivelul următor *(arhitectura apelurilor la distanţă)*, evidenţiază ca principale deosebiri între DCOM şi CORBA modul de înregistrare a server-elor şi momentul în care se creează componentele *proxy/stub*, respectiv *stub/skeleton*. Deoarece o parte dintre aceste caracteristici au fost prezentate comparat în cadrul paragrafului anterior, ele vor fi doar enumerate, cu precizarea elementelor corespondente din cele două arhitecturi:

- locul în care se păstrează numele implementării obiectului: registru - *Registry* (DCOM), "depozit" de implementări - *Implementation Repository* (CORBA);
- locul în care în care se păstrează informaţii despre tipul şi caracteristicile interfeţelor: bibliotecă de tipuri - *Type Library* (DCOM), "depozit" de interfeţe - *Interface Repository* (CORBA);
- componenta arhitecturii care realizează localizarea implementării obiectelor: *SCM - Service Control Manager* [13] (DCOM), ORB - *Object Request Broker* (CORBA);
- componenta arhitecturii care realizează activarea implementării: SCM (DCOM), OA - *Object Adapter* (CORBA);
- pentru realizarea apelurilor la distanţă clientul include o componentă: *proxy* (DCOM), *stub* sau *proxy*, după caz (CORBA);
- pentru realizarea apelurilor la distanţă server-ul include o componentă: *stub* (DCOM), *skeleton* (CORBA);

Alte două deosebiri importante care apar la acest nivel sunt:

- DCOM permite folosirea obiectelor cu interfeţe multiple şi, ca o consecinţă a acestui fapt, asigură posibilitatea ca un obiect *proxy/stub* să determine încărcarea mai multor interfeţe *proxy/stub*; în CORBA nu există un mecanism similar;
- în CORBA, toate interfeţele moştenesc clasa **CORBA::Object** al cărei constructor realizează în mod implicit operaţiile de înregistrare a obiectelor, generare a referinţelor la obiect, crearea instanţelor *skeleton* etc. în DCOM aceste sarcini revin programelor server sau sunt gestionate dinamic de către executivul DCOM.

Nivelul *arhitecturii protocolului de legătură* extinde arhitectura apelurilor la distanţă astfel încât acestea să se poată realiza între procese ce se execută pe maşini distincte. La acest nivel DCOM şi CORBA evidenţiază multe asemănări dar şi câteva deosebiri semnificative, care se referă, în principal la:

- modul de reprezentare a pointerilor la interfaţă, respectiv a referinţelor la obiect, astfel încât să poată asigura livrarea informaţiei cerute de către client server-ului aflat la distanţă;
- formatul standard în care se translatează datele pentru a fi vehiculate în cadrul sistemelor eterogene.

Elementele corespondente din cele două arhitecturi implicate la acest nivel sunt [5][13]:

- prelucrarea informaţiilor de conectare între client şi obiect: *OXID Resolver* (DCOM), ORB (CORBA);
- conectarea la server se realizează de către: exportatorul de obiecte (DCOM), OA - *Object Adapter* (CORBA);
- referinţa la obiect este reprezentată prin: OBJREF [13] (DCOM), IOR - *(Interoperable Object Reference)* sau referinţă la obiect, după caz (CORBA);

- generarea referinţei la obiect este realizată de: exportatorul de obiecte (DCOM), OA - *Object Adapter* (CORBA);
- formatul standard în care se translatează datele pentru a fi vehiculate în cadrul sistemelor eterogene *(marshalling format)*: *NDR - Network Data Representation* DCE [5] (DCOM), CDR - *Common Data Representation* (CORBA);

Luând în considerare valoarea celor două arhitecturi, exprimată atât prin avantaje / dezavantaje, cât şi prin actualii şi potenţialii utilizatori ai acestora, OMG a iniţiat procesul de standardizare a interconectării celor două tehnologii, specificaţia *DCOM/CORBA Interworking* fiind în curs de finalizare [7]. De altfel, firma IONA Technologies, a anunţat deja pentru luna februarie a anului 1998 lansarea în variantă beta a primului produs din cadrul unei linii de produse software, care implementează această interconectare, oferind utilizatorilor DCOM ai platformelor Windows posibilitatea de a-şi integra aplicaţiile în alte platforme, colaborând cu utilizatori CORBA şi beneficiind de toate avantajele oferite de către cele două arhitecturi. Linia de produse poartă numele *OrbixCOMet*.

3. Studiu comparativ: Java/CORBA, ActiveX/DCOM

În lumea sistemelor distribuite, Java şi CORBA constituie o combinaţie ce pare ideală. Având multe asemănări din punctul de vedere al modelului orientat obiect (distincţie netă între interfaţă şi implementare sau clasă, corespondenţă între tipurile de date CORBA IDL şi tipurile de date Java, mecanisme de moştenire aproape identice, corespondenţă între spaţiile de nume (module) CORBA şi pachetele Java etc.), Java şi CORBA îndeplinesc roluri complementare din punct de vedere arhitectural: Java permite crearea unor obiecte portabile şi distribuirea acestora în reţea, în timp ce CORBA asigură interconectarea acestora şi integrarea lor cu restul componentelor din mediul de calcul ales (baze de date, sisteme native *(legacy systems)*, obiecte sau aplicaţii scrise în alte limbaje etc.).

Din momentul lansării versiunii JDK 1.1, Java dispune de propriul ORB generic (inclus), numit *RMI (Remote Method Invocation)*, care permite realizarea unor invocări de metode asupra obiectelor aflate la "distanţă", dar nu este un ORB compliant CORBA, ci doar o extensie a nucleului limbajului. Java (şi RMI) reprezintă o *tehnologie de programare* destinată, în principal, scrierii şi organizării codului executabil al programelor. Pentru colaborarea cu programe scrise în alte limbaje, Java permite folosirea unei API numite *JNI (Java Native Interface)* care este destul de greu de manipulat şi care se adresează, mai ales, limbajelor C şi C++. RMI oferă anumite avantaje programatorilor (un obiect Java poate invoca metode ale altui client Java aflat la distanţă), dar nu permite colaborarea unei aplicaţii Java cu obiecte sau cu aplicaţii scrise într-un alt limbaj. Prin comparaţie, CORBA reprezintă o *tehnologie de integrare* [15], proiectată special pentru a constitui liantul între aplicaţiile scrise în limbaje diferite.

De ce sunt necesare aplicaţii scrise în alte limbaje decât Java dacă acesta pare să fie limbajul viitorului ? Răspunsul la această întrebare (şi, implicit, evidenţierea rolului CORBA în comparaţie cu RMI) se referă la mai multe aspecte, dintre care [15]:

- portabilitatea programelor Java implică anumite penalizări din punctul de vedere al performanţei (programele Java sunt executate de către o maşină virtuală, nu direct de către hardware); pentru această problemă s-a găsit însă o rezolvare prin implementarea maşinii virtuale Java într-o pastilă de siliciu (*chip* - ul *picoJava I*), cu o arhitectură asemănătoare procesoarelor *RISC (Reduced Instruction Set Computer)*, optimizată pentru performanţe de execuţie maxime; s-au obţinut, în acest mod, performanţe mai bune de 15-20 de ori, faţă de varianta procesor 486 cu interpretor, şi de 5 ori, faţă de varianta procesor Pentium cu compilator *JIT (Just In Time)* [16];
- Java elimină multe dintre problemele de programare, legate de gestionarea memoriei, asigurând un mecanism de colectare a blocurilor de memorie, neutilizate în rezerva de blocuri libere *(automatic garbage collection)*; această facilitate implică, însă, penalizări de performanţă inacceptabile în cazul unor aplicaţii mai speciale (sisteme de timp real cu cerinţe foarte stricte, sisteme cu încărcare foarte mare etc.) deoarece, pe durata realizării colectării de blocuri neutilizate, maşina virtuală forţează suspendarea aplicaţiei;
- limbajul Java poate deveni la rândul său un limbaj "învechit", ale cărui numeroase şi valoroase aplicaţii deja create *(legacy systems)* să se dorească a fi înglobate sau doar folosite în/de către aplicaţiile viitorului;
- RMI este, în esenţă, o arhitectură specifică implementării Java în timp ce ORB-urile compliante CORBA pot fi

create astfel încât să îndeplinească cerinţe de mediu de execuţie sau de sistem specifice (de exemplu, există ORB-uri specializate pentru sisteme dedicate în care ORB-urile proiectate pentru staţiile unui sistem de tip client / server ar fi complet inadecvate) şi, în acelaşi timp, să poată interacţiona şi să asigure portabilitatea aplicaţiilor;

- implementările ORB pot fi modificate pentru a oferi caracteristici superioare de performanţă, siguranţă în funcţionare şi scalabilitate fără a impune modificarea aplicaţiilor deja create (CORBA este o specificaţie de interfeţe, nu un cod bază pentru implementare);
- în prezent, există o gamă largă şi flexibilă de produse ORB ce acoperă domenii dintre cele mai diverse, de la *"ORBlets"* scrise în Java şi destinate clienţilor Web din sisteme distribuite (pot fi "descărcate" în *browser*-e sau chiar *browser*-ul include suport CORBA), până la ORB-uri ce lucrează în sisteme tranzacţionale de timp real; deşi reprezintă un instrument valoros, RMI poate fi considerat, prin comparaţie, un mecanism simplificat al cărui domeniu de lucru este mult mai restrâns;
- CORBA include un număr mare de servicii esenţiale pentru scrierea şi folosirea aplicaţiilor în cadrul sistemelor distribuite; RMI permite folosirea unui număr mai mic de astfel de servicii (printre care *Naming, Directory, Security*), existând preocuparea de a se adopta specificaţia CORBA pentru serviciile care urmează a fi implementate în continuare.

Ca răspuns al "provocării" Java în domeniul WWW, firma Microsoft a lansat o extensie a OCXs, numită *ActiveX*, care cuprinde un set de tehnologii (COM/DCOM/OLE/OCX), create pe baza controalelor OLE, şi destinate facilitării dezvoltării de aplicaţii în Internet [1][2]. O aplicaţie care este un control ActiveX poate fi inclusă în pagini Web sau într-un container ActiveX de tip VisualBasic, Visual C++, Visual J++ sau Microsoft Access. Spre deosebire de Java care reprezintă un limbaj de programare şi specificaţia unei maşini virtuale, ActiveX reprezintă o arhitectură care permite unui program (controlul ActiveX) să interacţioneze cu alte programe existente în reţea (de exemplu, în Internet). *Controalele ActiveX* pot fi create şi folosite ca părţi ale unui program sau componente de sine stătătoare. Odată create, pot fi reutilizate chiar şi în cadrul unor programe scrise cu ajutorul unui limbaj de programare diferit. Această caracteristică provine din faptul că obiectele ActiveX sunt de fapt obiecte OLE cu funcţionalitate adaptată cerinţelor WWW. Un avantaj important, oferit de această moştenire, este posibilitatea folosirii obiectelor OLE scrise până în prezent în rol de controale ActiveX, ceea ce permite crearea programelor cu minimum de timp şi de efort (există biblioteci voluminoase de obiecte OLE; se pot reutiliza şi controalele elaborate prin efort propriu). Ceea ce lipseşte încă produsului ActiveX, dar există în cazul limbajului Java, este proprietatea de a "scrie o dată" şi a de a "executa oriunde" *("write once, run everywhere")* şi anumite facilităţi de asigurare a securităţii.

În timp ce un *applet* Java se execută pe o maşină virtuală, care lucrează "deasupra" sistemului de operare (*applet*-ul nu are deci nimic în comun cu sistemul de operare), un control ActiveX funcţionează numai dacă este compilat pe platforma (hardware + sistem de operare) în care urmează să fie folosit. Aceasta înseamnă că, în timp ce o pagină Web - Java poate fi folosită de orice *browser* care implementează JVM, o pagină Web - ActiveX poate fi utilizată de *browser* doar în cazul în care autorul paginii a avut grijă să compileze controlul pentru unitatea centrală de prelucrare, specifică platformei pe care se lucrează (chiar şi în cazul controalelor ActiveX scrise în limbajul Java). Prin urmare, creatorii de aplicaţii care folosesc ActiveX sunt nevoiţi să păstreze mai multe copii ale fiecărui control ActiveX, adică cel puţin câte o copie pentru fiecare platformă client cu care ar putea lucra [11]. În plus, până în prezent, singurele platforme software pe care se pot executa controale ActiveX sunt Windows 95, Windows NT şi Windows 3.x (urmând să apară facilităţi UNIX şi MacIntosh).

O altă deosebire importantă între Java şi ActiveX o reprezintă asigurarea securităţii. Datorită modului de execuţie (maşina virtuală care se interpune între programul Java şi sistemul de operare), Java asigură un mecanism de izolare a *applet*-ului *("sandbox-ing")* faţă de sistemul gazdă, astfel încât să nu poată realiza operaţii neautorizate de către acesta (codul executabil Java este obligat să rămână într-o zonă bine delimitată) [17]. Controalele ActiveX funcţionează asemănător fişierelor executabile (.EXE) Windows deoarece sunt trecute prin faza de compilare. Prin urmare, nu se pot impune restricţii asupra acţiunilor pe care le realizează ca urmare a instrucţiunilor pe care le includ (se cunoaşte exemplul controlului ActiveX "Exploder", scris de către Fred McLain (http://www.halcyon.com/mclain/ActiveX/) care, în momentul în care era descărcat în Internet Explorer 3.0, iniţia o aşteptare de 10 secunde, după care dezactiva Windows 95 şi "închidea" maşina gazdă [1]). Ceea ce se poate realiza în cazul utilizării controalelor ActiveX este verificarea

identităţii acestora şi acceptarea sau refuzarea descărcării lor în *browser* pe baza acestei identităţi. Verificarea nu împiedică un control ActiveX să realizeze o acţiune nepermisă (atunci când s-a acceptat descărcarea), ci furnizează doar posibilitatea identificării autorului acestei acţiuni. Identitatea unui control ActiveX este creată de *Authenticode*, produs Microsoft care implementează "semnarea" codului cu ajutorul *certificatului numeric* (informaţii despre creatorul controlului respectiv: nume, adresă e-mail etc.) şi a *semnăturii numerice* (un identificator unic specific, criptat, similar amprentelor folosite în criminalistică). Identificatorul Authenticode astfel rezultat este inclus în controlul ActiveX. Pentru a putea utiliza Authenticode şi a crea un certificat numeric, creatorul controlului are nevoie de o *cheie publică* şi de o *cheie privată*, înregistrate de către o *autoritate de atestare (Certifying Authority)* [1], procedura de obţinere a acestora fiind destul de complicată. *Browser*-ul Internet Explorer verifică identitatea controalelor ActiveX printr-un model numit *"bouncer"* ("paznic"), care permite utilizatorului să folosească facilităţile de securitate pe trei niveluri distincte:

- nu se permite descărcarea controalelor ActiveX care nu sunt semnate (nu au identificator numeric);
- semnalizează încercarea de descărcare a unui control ActiveX fără semnătură şi lasă utilizatorul să hotărască dacă se realizează sau nu această operaţie;
- se descarcă automat orice control ActiveX (cu sau fără semnătură).

Un model similar este folosit şi în produsele firmei NetScape. În plus, în ambele tipuri de *browser*-e se poate crea o listă cu autorii de controale în care *browser*- ul să aibă implicit "încredere" (să descarce controalele automat, oricare ar fi nivelul de securitate selectat).

În sfârşit, o ultimă deosebire de esenţă între ActiveX/DCOM şi Java/CORBA se referă la dimensiunea aplicaţiilor şi la echipamentul hardware implicat: în timp ce aplicaţiile DCOM sunt prea voluminoase pentru a fi executate pe alte echipamente decât sisteme de calcul *desktop* (de birou), implementările Java ale arhitecturii CORBA se pot utiliza chiar şi pentru sisteme incluse în telefoane mobile, televizoare, agende şi planificatoare eletronice etc. Aceasta nu înseamnă însă că ActiveX/DCOM nu reprezintă un instrument valoros, ci doar că fiecare dintre cele două alternative (Java/CORBA, ActiveX/DCOM) prezintă anumite caracteristici care se constituie în avantaje sau dezavantaje în raport cu tipul de aplicaţie pentru care este folosită.

4. Concluzii

Folosind un material bibliografic de ultimă oră, articolul analizează comparativ cele mai importante caracteristici de structură şi de funcţionalitate ale unor instrumente software valoroase, dedicate elaborării şi utilizării aplicaţiilor distribuite (tehnologiile DCOM şi CORBA şi limbajul Java). În articol se subliniază rolul şi avantajele oferite de către instrumentele software, studiate în funcţie de tipul aplicaţiei şi al categoriei de utilizatori cărora li se adresează, fiind relevată importanţa existenţei unor mecanisme care să permită colaborarea şi interconectarea aplicaţiilor scrise conform DCOM, CORBA sau Java. În esenţă, concluziile rezultate în urma studiului comparativ realizat sunt:

- datorită interoperatibilităţii ORB-urilor (asigurată de standardul CORBA 2.0), a gamei largi de servicii şi a facilităţilor de programare orientată obiect oferite, este de aşteptat ca tehnologia CORBA să devină un standard unanim acceptat, care să permită crearea unor aplicaţii deosebit de performante în domeniul sistemelor distribuite eterogene. În perspectivă, se urmăreşte ca această tehnologie obiectuală, standardizată de către OMG, să fie încorporată în sistemele de operare, ce vor fi livrate o dată cu staţiile de lucru ale celor mai importante firme specializate, într-un mod asemănător DCOM/Windows;
- valoarea arhitecturilor DCOM şi CORBA, exprimată atât prin avantaje/dezavantaje, cât şi prin actualii şi potenţialii utilizatori ai acestora, a determinat OMG să iniţieze procesul de standardizare a interconectării celor două tehnologii, specificaţia *DCOM/CORBA Interworking* fiind în curs de finalizare;
- combinaţia Java/CORBA constituie un instrument software deosebit de puternic şi de mare perspectivă. Având multe asemănări din punctul de vedere al modelului orientat obiect (distincţie netă între interfaţă şi implementare sau clasă, corespondenţă între tipurile de date CORBA IDL şi tipurile de date Java, mecanisme de moştenire aproape identice, corespondenţă între spaţiile de nume (module) CORBA şi pachetele Java etc.), Java şi CORBA îndeplinesc roluri complementare din punct de vedere arhitectural: Java permite crearea unor obiecte portabile şi distribuirea acestora în reţea, în timp ce CORBA asigură interconectarea

obiectelor şi integrarea lor cu restul componentelor din mediul de calcul ales (baze de date, sisteme native *(legacy systems)*, obiecte sau aplicaţii scrise în alte limbaje etc.);

- deosebirile de esenţă dintre ActiveX/DCOM şi Java/CORBA se referă:
  1. la proprietatea de a "scrie o dată" şi de a "executa oriunde" *("write once, run everywhere")*,
  2. la anumite facilităţi de asigurare a securităţii,
  3. la dimensiunea aplicaţiilor şi a echipamentului hardware implicat.

Bibliografie

1. * * *: Comparing ActiveX and CORBA/IIOP - *http://www.omg.org/news/activex.htm*
2. **LINTHICUM, D., S.:** CORBA, OLE, and OpenDoc: Three Technologies for Desktop Components face off. În: BYTE, January 1996/State Of The Art/Integration, Not Perspiration- *http://www.byte.com/art/9601/sec8/art2.html*
3. **SIEGEL, J.:** CORBA Fundamentals and Programming, Wiley Computer Publishing Group, USA,1996.
4. **KATIYAR, D.:** Notes on OLE and CORBA - *ftp://theory.stanford.edu/pub/katiyar/misc/ole_vs_corba.html*
5. **BROWN, N., KINDEL, CH.:** Distributed Component Model Protocol - DCOM/1.0.Internet Draft- *http://www.microsoft.com/oledev/olecom/draft-brown-dcom-v1-spec01.txt*
6. **WANG, YI-MIN:** Introduction to COM/DCOM- *http://akpublic.research.att.com/~ymwang/slides/DCOMHTML/ppframe.htm*
7. * * *: OMG Technology Committee Work in Progress. OMG Technology Adoptions - *http://www.omg.org/library/schedule.htm*
8. * * *: CORBA 2.0 - *http://www.omg.org/corba/corbaiiop.htm* CORBA Services available electronically - *http://www.omg.org/corba/sectrans.htm#sec*
9. **RESNICK, R., I.:** Bringing Distributed Objects to the World Wide Web - *http://www.interlog.com/~resnick/javacorb.html#orfali*
10. **GARG, R.:** CORBA: The Future of Web Computing - Sun Developer News - *http://www.sun.com/developers/devnews/summer97/corba.html*
11. * * *: OMG Announces Distributed Document Component Facility - *http://www.omg.org/index.html*
12. * * *: The Component Object Model: Technical Overview - *http://www.microsoft.com/oledev/olecom/Com_modl.htm*
13. * * *: ORB Portability Joint Submission (Final) Part1 of 2, ORBOS/97-05-15. ORB Portability Enhancement (POA - Portable Object Adapter) - *ftp://ftp.omg.org/pub/docs/orbos/97-05-15.pdf*
14. **CURTIS, D.:** Java, RMI and CORBA. A white paper - *http://www.omg.org/news/wpjava.htm*
15. **LASZLO, BUDAI:** RISC-ul Java. În: BYTE România, vol. 4, nr.1, ianuarie, 1998.
16. **WUKTA, M. et al.:** Hacking Java. The Java Professional's Resource Kit, QUE Corporation, SUA, 1997.

PPCsm^{2}e

Production Planning and Control System for Small and Medium Manufacturing Enterprises

Target Users

PPCsm^{2}e addresses one of the highest priority domains of today economic development: small and medium manufacturing enterprises. Both order-based and order- and forecast-based production type enterprises are considered. ***PPCsm^{2}e*** may be appreciated as a significant achievement of this kind on the Romanian market and belongs to a limited offer at the European level. Its commercial success depends on the emphasis put by target enterprises on using IT oriented solutions to strengthen their business competitive potential.

Short Description

PPCsm^{2}e software is meant to assist production planning decisions on various planning horizons.

PPCsm^{2}e has been developed in the framework of the EC funded COPERNICUS'94 Project **RaPOrt** (#1191) – *"Rapid Prototyping of Object-Oriented Production Planning and Control Systems for Industrial SME's"*

The provided decision support is based on the production resources availability checking and consists in identifying conditions under which the current production plan may be successfully completed.
PPCsm^{2}e implements the following PPC functions:

- master production scheduling;
- material and capacity requirements planning;
- previsional scheduling of technological operations;
- order progress reporting.

Besides this basic functionality, the ***PPCsm^{2}e*** modular architecture includes system administration, as well as technical and commercial data administration functions.
The installation procedure provides the capability to adapt the functional configuration of the product to specific user requirements.
PPCsm^{2}e is a non-sophisticated, easy-to understand and operate solution. Context sensitive help features are provided to assist inexperienced users.

Technical Requirements

PPCsm^{2}e requires low cost investments, being oriented to Windows'95/ Windows NT platforms on usual PC configurations.

Delivery Conditions

- technical assistance for product implementation
- training
- documentation on floppy disk or paper
- after sale services and up-grades

References

The system prototype has been implemented in two pilot enterprises:

- UTAL S.A. Bucharest – enterprise for technological installations in food industry
- Pifati S.A.Bucharest – enterprise for heating devices

The major benefits of ***PPCsm^{2}e*** implementation, as confirmed by the pilot implementations, are the following:

- higher accuracy and efficiency in product and technological data administration
- faster and more consistent analysis of market demands
- greater efficiency and flexibility in production program development at various planning levels
- more accurate utilisation of production resources based on up-to-date production control information