Aspecte ale proiectarii unui *Order Request Broker* (ORB)
(Partea I)

Claudiu VINTE, Goldman Sachs Ltd., Tokyo, Japonia

*Voi prezenta în acest articol detaliile arhitecturale si maniera de implementare a unei platforme de comunicatie în retea care asigura trasferul mesajelor între aplicatiile unui sistem, vazut la nivel logic, într-o maniera transparenta din punctul de vedere al programatorului de aplicatii si a aplicatiilor luate ca module independente. Aceasta platforma de comunicatie se comporta ca un **O**rder **R**equest **B**roker (**ORB**) pentru aplicatiile care se conecteaza la ea, oferind serviciile la care respectivii clienti au subscris. Am denumit acesta platforma de comunicatie **XQuark** (e**X**change **Q**uark – ca o componenta light si cu performate ridicate în realizarea schimbului de date) si vom face referire la ea, de acum înainte, sub aceasta denumire sau, mai simplu, **XQ**.*

Cuvinte cheie: *arhitectura, comunicatie, aplicatie, sistem, mesaje.*

Descriere a arhitecturii si functionalitate

Proiectarea unei astfel de componente de comunicatie trebuie sa ia în considerare faptul ca modulele sistemului pot fi distribuite pe platforme cu sisteme de operare diferite si, în consecinta, acest *ORB* trebuie sa fie implementat de asa natura încât sa poata fi portat pe o gama larga de platforme. Atunci când am pornit la proiectarea XQuark-ului am avut în vedere, în principal posibilitatea de folosire a acestuia sub doua sisteme de operare larg raspândite: idiomuri de UNIX si Microsoft Windows (98 si NT). În mod current am rulat XQuark sub Windows98, NT, UNIX (SunOS, Solaris), LINUX.

Pentru a realiza aceasta transparenta a comunicarii din perspectiva programelor de aplicatie se impune un nivel de indirectare. Si anume, fiecare aplicatie (vom denumi în acest mod generic orice componenta a sistemului, fie ca joaca rolul de server fie cel de client, din perspectiva functionala în cadrul sistemului) care vrea sa comunice cu celelalte aplicatii din sistem trebuie sa se conecteze la XQuark, care va trebui sa ruleze pe masina respectiva si care va prelua toata sarcina transmiterii si receptiei mesajelor în retea. XQuark va prelua astfel mesajele transmise de aplicatia (sau aplicatiile) locale si le va transmite în mod corespunzator aplicatiei careia îi sunt destinate prin intermediul unui alt XQuark care trebuie sa ruleze pe masina pe care ruleaza programul desti-natar. Mecanismul de ansamblu este ilus-trat în ***figura 1***, care prezinta configuratia formata dintr-un client (101, 1, 0) rulând pe o masina si, respectiv, un server (3, 1, 0) si un alt client (101, 2, 0) rulând pe o alta masina. Din puntul de vedere al Xquark toate aplicatiile sunt clienti ai acestuia, percepându-l ca si *ORB*.

Pentru a realiza o uniformizare a accesului, interfata pusa la dispozitia aplicatiilor client de catre XQuark este limitata la o singura clasa *XQuarkLink*. Partea publica a acestei interfete este usor de replicat dându-se o alta biblioteca de retea.

Arhitectura de ansamblu contine trei bibioteci care sunt puse la dispozitia aplicatiilor, si care concura la realizarea transparentei în schimbul de mesaje:

- **XQmw** – biblioteca *XQuarkMiddle Ware*, care pune la dispozitie mecanismele necesare conexiunii la nivel de aplicatie;
- **XQtw** – biblioteca *XQuarkThread Ware*, în care sunt implementate toate mecanismele de creare si gestionare a firelor de executie multiple;

❑ **XQnw** – biblioteca XQuarkNetWare, în care sunt gestionate conexiunile socket (TCP si UDP) necesare comunicarii si care furnizeaza pentru aplicatii clasa *XQuark Link* ca interfata de comunicare între XQuark si aplicatiile care subscriu la serviciile acestuia.

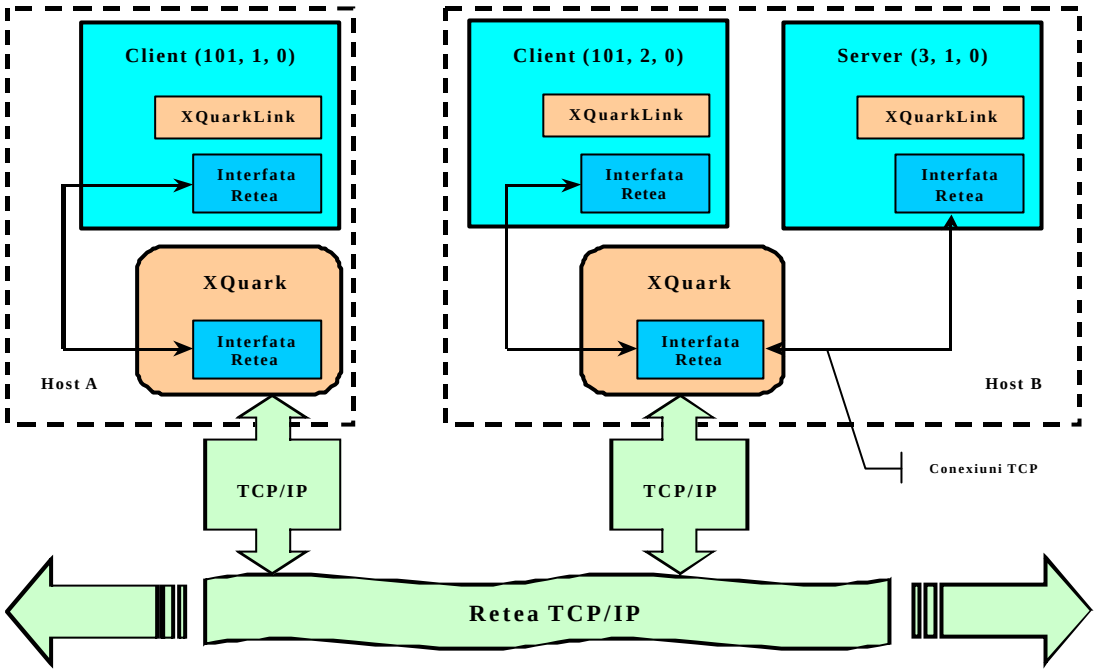

Figura 1

Biblioteca *XQuarkNetWare* pune la dispozitia aplicatiilor client urmatoarele facilitati:

❑ Posibilitatea de utilizare pe multiple platforme - asa cum am mentionat XQuark poate sustine comunicatia între aplicatii care se gasesc pe diferite platforme într-o aceeasi retea la nivel logic a sistemului (Windows98, NT, UNIX – SunOS & Solaris, LINUX); alte platforme UNIX pot fi de asemenea utilizate daca implementeaza fire de executie multiple POSIX; biblioteca ofera posibilitatea conversiei automate a ordinii octetilor atunci când mesajele sunt transmise în retea (procesoarele Intel fac alinerea la otetul cel mai mic al cuvântului – aliniere la stânga – pe când procesoarele Sparc fac alinierea la octetul cel mai mare al cuvântului – aliniere la dreapta);

❑ independenta localizarii spatiale; aplicatiile client nu trebuie sa stie unde sunt localizate celelalte aplicatii cu care vor sa comunice, pentru a trimite mesaje catre acestea si pentru a receptiona la rândul lor mesajele altor aplicatii (chintesenta unei arhitecturii bazate pe un *Order Request Broker*); întreaga activitate de manipulare a mesajelor este realizata, în mod transparent pentru aplicatii, de catre XQuark-urile care trebuie sa ruleze pe fiecare masina din reteaua sistem;

❑ mesaje de stare; XQuark urmareste starea fiecarui client (daca este pregatit sa primeasca mesaje sau este deconectat etc.); daca un mesaj trebuie sa fie trimis catre un client care nu este pregatit sa primeasca mesaje atunci un mesaj având continutul "nu exista o astfel de destinatie" este imediat trimis aplicatiei care a initiat comunicarea;

❑ paradigme multiple de mesagerie; sunt suportate ambele mecanisme de transmisie: *broadcasting* (UDP) si transmisie *point-to-point* (TCP);

❑ instante multiple; pot fi conectate mai multe instante ale unei aplicatii client rulând pe o aceeasi masina la XQuark-ul care va satisface comunicatia tuturor aplicatiilor care ruleaza pe masina respectiva si care au subscris la serviciile sale.

Asa cum este înfatisat în ***figura 1***, pentru a interschimba mesaje fiecare aplicatie trebuie sa se conecteze la XQuark. Aceasta este realizata prin instantierea unui obiect al clasei *XQuarkLink* care va fi descris în continuare. Protocolul de stabilire a conexiunii asigneaza totodata un unic identificator pentru fiecare aplicatie client (adresa acesteia) din reteaua logica a sistemului care este un obiect al clasei *NetworkID*.

Toate mesajele care se vehiculeaza în retea prin mijlocirea platformei create de XQuark trebuie sa fie obiecte derivate din clasa de baza *NetMessage*. Aceasta pentru a se asigura faptul ca fiecare obiect al mesajului este serializabil. Atunci când un mesaj este trimis de catre o aplicatie client acesta este transferat de catre obiectul *XQuarkLink* catre procesul XQuark. În mod similar, când un mesaj pentru aplicatia client ajunge la XQuark-ul care ruleaza pe masina pe care este instalata aplicatia, mesajul este pasat clientului prin intermediul aceluiasi obiect *XQuarkLink*. Comunicatia dintre aplicatia client si XQuark, pe masina locala, se realizeaza pe baza de conexiune *socket*. Pe UNIX, acesta este un UNIX *domain socket* creat in directorul */tmp*. Pe Windows, acesta este un socket normal de retea. Cu toate ca cele mai multe mesaje pe care o aplicatie client le primeste vor fi mesaje trimise acesteia de catre o alta aplicatie client (aflata pe masina locala sau în retea – conexiune point-to-point), pot fi primite, de asemenea, mesaje pe care aplicatia client nu le-a solicitat. Acestea fac parte din categoria mesajelor de *broadcast* (care pot contine date tranzactionale sau pot fi mesaje de control). Când un mesaj soseste pentru un client, acesta este plasat într-o coada de mesaje de catre obiectul *XQuark Link*. Mesajul este apoi procesat de aplica-tia client atunci când aceasta are nevoie de el. Sunt folosite diferite strategii de notificare a receptiei de mesaje, functie de platforma: sub Windows sunt folosite cozi de mesaje Windows, sub UNIX sunt utilizate variabile de conditie. Sub-rutarea în interiorul aplicatiei este de asemenea posibila, prin utilizarea celui de al treilea câmp al structurii din clasa *NetworkID*, realizându-se astfel directionari de mesaje catre componente specifice ale aplicatiei client (de exemplu catre managerul unei anumite ferestre a aplicatiei).

Protocolul pentru stabilirea conexiunii si realizarea schimbului de mesaje se reali-zeaza în maniera urmatoare:

- Când un XQuark este startat el trimite un mesaj de *broadcast* "AM_ALIVE" în retea si asteapta mesaje de confirmare de la celelalte noduri ale retelei în care ruleaza alte procese XQuark. Totusi, daca instanta XQuark se lanseaza cu parametrul "-h" în linia de comanda urmat de adresa IP a unui host tinta, atunci acesta nu trimite mesajul de *broadcast* ci se conecteaza în mod direct la masina specificata.
- Când un XQuark primeste un mesaj "AM_ALIVE", el se conecteaza imediat la masina de la care a primit mesajul.
- Atunci când conexiunea este stabilita (conexiune TCP/IP) fiecare XQuark trimite informatiile legate de aplicatiile locale care s-au conectat la el, si pe care le pastreaza într-o harta a întregului sistem, catre celelalte XQuark-uri din retea. În acest fel fiecare XQuark are "cunostinta" de toate aplicatiile care ruleaza în reteaua logica a sistemului (si sunt conectate la un proces XQuark); identificatorilor de aplicatii afla-te la distanta asignându-li-se la acest mo-ment identificatori unici pe masina locala.
- Unei aplicatii care se conecteaza la un XQuark i se asociaza un *NetworkID* unic pe masina locala.
- Când o aplicatie client întrerupe conexiunea cu XQuark-ul la care este conectata acesta din urma trimite un mesaj de *broadcast* "PROGRAM_DIED". Orice mesaj trimis catre aceasta aplicatie client de acum înainte va genera din partea procesului XQuark aflat pe masina locala un raspuns: "nu

exista o astfel de destinatie". Daca un XQuark este întrerupt, toate celelalte instante XQuark aflate în retea vor genera mesaje "PROGRAM_ DIED" pentru toate aplicatiile client care erau conectate la respectivul XQuark care si-a întrerupt activitatea, si vor trimite aceste mesaje, fiecare în parte, clietilor locali de pe masina pe care ruleaza fiecare.

Interfata pusa la dispozitia aplicatiilor client

Toate aplicatiile client trebuie în mod ne-cesar sa creeze o instanta a clasei *XQuark Link*, pentru a face posibila conexiunea catre XQuark-ul de pe masina locala. Din moment ce mecanismul de distribuire a mesajelor difera functie de platforma, constructorul este diferit pentru fiecare platforma (Win32 sau UNIX).

Doua servicii importante sunt oferite de obiectul *XQuarkLink*:

- SendMessage(const NetNessage &, bool = true) – trimite un mesaj catre o alta entitate, prin intermediul unei conexiuni TCP/IP daca este specificata destinatia, altfel se realizeaza un *broadcast* de pe socketul UDP/IP;
- BroadcastIsOpen() – la startare, fiecare aplicatie client trebuie sa apeleze aceasta functie o singura data. Acesta realizeaza informarea fiecarei entitati din reteaua logica a sistemului asupra faptului ca aplicatia client în cauza este gata sa primeasca si sa triminta mesaje. Daca acest lucru nu este realizat atunci procesul XQuark de pe masina pe care ruleaza aplicatia client nu va directiona mesaje catre aceasta.

Mesajele primite de catre obiectul XQuarkLink sunt trimise catre aplicatia client în mod diferit, depinzând de platforma:

- Pe UNIX, mesajul este pus într-o coada de mesaje si o variabila de conditie este semnalizata pentru informarea aplicatiei client despre faptul ca a fost primit un nou mesaj. În mod normal aplicatia client va astepta ca aceasta variabila de conditie sa fie setata si va începe prelucrarea tuturor mesajelor din coada de mesaje imediat ce aceast lucru s-a realizat.
- Pe Windows, mesajul este trimis catre fereastra principala a aplicatiei client prin intermediul unui apel *PostThreadMessage*. Mesajul apare apoi în coada de mesaje a respectivei ferestre si este procesat ca oricare alt mesaj trimis unei ferestre sub Windows. Daca al treilea câmp al structurii din obiectul *NetworkID* are o valoare diferita de zero (câmpul *AppSubID*), atunci acesta este interpretat de managerul feres-trei principale a aplicatiei si mesajul este redirectat catre fereastra pentru care a fost destinat printr-un apel *PostMessage*.

Managerul de ferestra - sub Win32 - (în mod curent fereastra principala a aplicatiei

client) si coada de mesaje, respectiv variabila de conditie - sub UNIX – trebuie sa fie specificate la momentul crearii obiectului *XQuarkLink* si nu pot fi schim-bate ulterior, pe durata de viata a sesiunii aplicatiei client.

Asa cum am mentionat anterior, toate me-sajele vehiculate sunt instante ale unor clase care trebuie sa fie derivate din clasa *NetMessage*. Aceasta asigura faptul ca obiectul din mesaj este serializabil si are acelasi format al antetului pentru toate mesajele (destinatar, expeditor, tipul mesa-jului). Toate mesajele definite de aplicatiile client trebuie, în consecinta sa fie derivate din clasa *ApplicationMessage*, care este la rândul ei derivata din clasa *NetMessage*.

Când un mesaj este trimis în retea, el este în prealabil serializat si scris într-un *stream* de catre functia *ToBytes*, care apeleaza la rândul ei functia *FlushToStream*. Functia *BinaryOutputStream* este folosita în acest proces pentru a realiza conversia de la ordinea octetilor folosita de platforma locala (*host byte order*) la ordinea folosita în retea (*network byte order*), pentru a se asigura uniformizarea formatului mesajelor în retea si compatibilitatea între platforme (la receptie se apeleaza functia corespon-denta - *BinaryInputStream*). Dupa transfer, se reface obiectul *NetMessage* prin inter-mediul unui apel *CreateFromStream*.

Adresa emitentului (*NetworkID*) este în mod automat inserata de catre construc-torul obiectului *NetMessage*. Totusi, apli-catia care a trimis mesajul poate seta câmpul *AppSubID* pentru a se asigura faptul ca mesajul ajunge în mod corect la destinatia (fereastra) dorita.

Mesajele predefinite sunt reprezentate în clasele *ControlMessage* si *StatusMessage*. În ambele clase sunt definite mesaje care în mod normal sunt generate fie de XQuark fie de operatorul care utilizeaza aplicatia consola de monitorizare a activitatii tuturor nodurilor sistemului si aplicatiilor client conectate la instata procesului XQuark care ruleaza în fiecare din aceste noduri. O aplicatie client este libera sa ignore unele dintre aceste mesaje sau pe toate.

Identificatorii de aplicatii utilizati pen-tru traficul în retea

Fiecarei entitati din sistem îi este asignat de catre XQuark un identificator unic (*NetworkID*). Un *NetworkID* consta din trei numere care au urmatoarele sem-nificatii:

- ***App***. Acesta este un *byte* (1 – 254) care identifica clasa de aplicatii client. De exemplu un *server* poate avea *App ID* 3 si un client al acestuia poate avea *App ID* 101 (cum am vazut în ***figura 1.***). Acest numar este *hard coded* în fiecare aplicatie client conectata la platforma XQuark).
- ***AppSerial***. Un întreg care identifica instata concreta a unei aplicatii. Este asignat de XQuark atunci când aplicatia client initiaza o conexiune catre acesta.
- ***AppSubID***. Acesta este mod uzual zero. Totusi, asa cum am mentionat, o aplicatie client poate sa particularizeze un mesaj înainte de al trimite în retea. Când destinatarul raspunde la acest mesaj, acest raspuns va fi directinat de catre XQuark catre componenta aplicatiei care doreste acest mesaj (*inter-application routing*).

Numarul *AppSerial* este atribuit de catre XQuark aplicatiei client în secventa de initializare a obiectului *XQuarkLink*. Acest numar este unic în interiorul unei clase de aplicatii client. Este important de precizat faptul ca o aceeasi aplicatie client poate avea *AppSerial ID* diferit pe masini diferite. De exemplu, în ***figura 1***, clientul (101, 1, 0) are aceasta adresa pe masina pe care ruleaza, pentru ca este unica instata a acestei aplicatii care ruleaza pe acesta masina. Cu toate acestea, pe masina pe care ruleaza *server*-ul (3, 1, 0) si clientul (101, 2, 0), clientul în cauza poate fi "cunoscut" sub adresa (101, 3, 0) pentru ca pe aceasta masina mai ruleaza o instanta a aceleiasi aplicatii si instantele aflate la distanta trebuie identificate unic pe masina locala, fapt ce se realizeaza în maniera

incrementala. Atunci când *server*-ul pri-meste un mesaj, *AppSerial ID* al clientului expeditor este de fapt un *ID* local pentru server. În mod similar, când *server*-ul raspunde clientului aflat la distanta (101, 1, 0) el trimite mesajul de raspuns la adresa locala a cestuia (101, 3, 0). Aceasta adresa este apoi transformata de catre XQuark-ul care ruleaza pe masina locala, în adresa corespunzatore aplicatiei aflate la distanta (101, 1, 0) si mesajul este trimis la masina corespunzatoare pe care ruleaza aceasta aplicatie.

Un mesaj trimis la adresa (*App*, 0, 0) va fi trimis la prima instanta a aplicatiei *App* care se afla înregistrata în *system map*. De aceea, în acest context, clientii nu trebuie sa fie foarte precisi cu adresa *server*-ul la care sunt conectati (cu exceptia cazului în care ruleaza în sistem mai multe *server*-e de acelasi tip, caz în care clientii trebuie sa stie precis la care dintre ele se conecteaza, altfel vor "vorbi" întotdeauna cu prima instanta a acestui *server* aflata în *system map)*.

Sunt definiti, de asemenea, doi identi-ficatori particulari: *NullID* si *BroadcastID*. Ultimul dintre acestia specifica platformei XQuark ca este vorba despre un mesaj de *broadcast*, care va fi trimis tuturor enti-tatilor din sistem. *NullID* este o adresa care nu este valida si este utilizata numai cu scopul detectarii de erori.

Monitorizarea starii sistemului (*System Map*)

Fiecare instanta XQuark întretine o struc-tura de date de tipul *SystemMap*. Acest obiect pastreaza câte o înregistrare pentru fiecare componenta cunoscuta din sistem (celelalte procese XQuark si aplicatiile client conectate la acestea) într-un tablou de tipul *SystemMapEntry*.

De asemenea, XQuark pastreaza câte o înregistrare pentru fiecare conexiune TCP într-un tablou de tipul *LinkInfo*. Ori de câte ori un mesaj este trimis în retea sau primit din retea, adresa expeditorului este translatata dintr-un *NetworkID* local într-un *NetworkID* de la distanta si viceversa, iar aceasta se realizeaza prin consultarea înregistrarilor stocate în *sytem map*. Daca o adresa nu este regasita în *sytem map* un mesaj de tipul "nu exista o astfel de destinatie" este returnat catre expeditorul mesajului. Datorita faptului ca structura *system map* este modificata de fiecare data când o noua aplicatie client se conecteaza (deconecteaza) la (de la) un XQuark, acesta din urma trimite mesaje de informare catre celelalte procese XQuark aflate în nodurile retelei sistemului, care la rândul lor îsi vor actualiza propriul *system map*, ca raspuns la aceste evenimente. În consecinta, fiecare XQuark "cunoaste" în orice moment starea fiecarei entitati din sistem.

Clasa *LinkInfo* contine obiectul *SSocket* utilizat pentru comunicarea prin canale de transmisie realizate cu ajutorul conexiuni-lor socket. In acesta clasa este implemen-tat, de asemenea, mecanismul de trimitere a mesajelor de *broadcast* (utilizând socket-uri UDP), formate din mai multe pachete, si retransmitere a pachetelor care sunt pierdute, si deci lipsesc din secventa corecta de pachete a mesajului. Daca un pachet dintr-un mesaj de broadcast nu este primit în ordinea corecta (*out-of-sequence packet*) atunci mesajul nu este trecut imediat mai departe la nivelul aplicatiei ci se trimite un mesaj *host*-ului care a transmis secventa originara de pachete prin care se cere acestuia retransmiterea pache-tului lipsa, actiune care se realizeaza prin intermediul unei conexiuni TCP (*point-to-point*, de aceasta data). In acest mod, dupa ce pachetul lipsa este primit, întreg mesajul este livrat nivelului superior (aplicatiei) din coada în care a fost temporar pastrat (*out-of-sequence queue*).

Trimiterea si receptionare mesajelor în reteaua sistem

Vom prezenta în continuare cum se reali-zeaza transferul de mesaje între aplicatii,

folosind *Order Request Broker*-rul XQuark ca platforma de intercomunicare în retea.

Pentru transmiterea unui mesaj, înainte de toate, trebuie sa construim un obiect corespunzator, derivat din clasa *NetMessage,* si care sa contina adresa destinatarului si tipul mesajului (subiectul actiunii care se doreste a fi realizata prin transmiterea acestui mesaj). Acest obiect este apoi tranferat metodei *SendMessage*, a instantei corespunzatoare clasei *XQuarkLink*. Daca adresa destinatarului este o adresa de *broadcast* (*NetworkID::BroadcastID()*), a-tunci mesajul este trimis în retea printr-un socket UDP; altfel, daca este specificata adresa concreta a destinatarului, mesajul este transmis printr-un socket TCP (*point-to-point*). În ambele cazuri este invocata metoda *WriteToStream* a clasei care încap-suleaza mesajul, si care are ca furnizeaza mesajul împachetat în tipul de data *BinaryOutputStream*. Asa cum am vazut anterior, acest obiect contine mesajul convertit într-un *stream* de oteti (facându-se, de asemenea, conversia de aliniere a octetilor de la cea a masinii locale – indiferent de platforma – la aliniarea folosita pentru transmisia în retea). Mesajul este transmis apoi printr-o con-ducta catre procesul XQuark local care îl distribuie catre aplicatia client pentru care a fost destinat.

Pentru receptionarea mesajelor sunt prevazute doua mecanisme. Pe UNIX, aplicatia client trebuie sa-si construiasca o coada de mesaje si sa transfere un pointer catre aceasta coada procesului XQuark la care se conecteaza, în momentul în care initiaza conexiunea. Când obiectul *XQuarkLink* primeste un mesaj de la XQuark, mesajul este pus într-o coada de mesaje si un eveniment corespunzator este semnalizat cu ajutorul unei variabile de conditie pe care a facut-o "cunoscuta ", de asemenea, procesului XQuark la momentul initierii conexiunii de catre aplicatia client. Este de datoria aplicatiei client sa monitorizeze acest eveniment (în mod normal prin ape-luri ale functiei *Wait*) si sa extraga mesajul din coada de mesaje. Mesajul primit este furnizat ca si *pointer* la un obiect de tipul *NetMessage*. Acesta trebuie sa fie identifi-cat în mod corect (*casting*) pe baza informatiei din *Subject*.

Sub Windows, poate fi folosit, de aseme-nea, mecanismul descris anterior, dar este mai sigura si ofera performante mai bune utilizarea cozilor de mesaje Windows, asa cum am aratat într-un paragraf anterior.

Continuarea în numarul viitor