6. Abordări şi soluţii arhitecturale privind OMS

Aşa cum precizam în capitolele precedente, componenta de bază a oricărui sistem informatic de asistare a tranzacţiilor bursiere este Sistemul de Gestiune a Ordinelor (**OMS**). În literatura de specialitate şi în practica tranzacţiilor bursiere, OMS (sau *Front Office*) este uneori asimilat cu întregul *Trading System* al firmei de brokeraj.
Realitatea domeniului arată că proiectarea unui astfel de sistem informatic complex nu se realizează niciodată printr-o abordare idealistic unitară. Întotdeauna activitatea financiară însăşi va fi cea care, în ultimă instanţă, va dicta nevoile. Un sistem informatic în acest domeniu este continuu supus modificărilor şi extinderilor, rare fiind situaţiile în care modulele esenţiale se înlocuiesc cu totul. De aceea, probabil cea mai importantă şi delicată, în acelaşi timp, exigenţă pe care trebuie să o satisfacă orice nouă abordare este aceea a flexibilităţii şi disponibilităţii pentru dezvoltări şi extinderi ulterioare.
De asemenea, datorită dimensiunii şi complexităţii unui astfel de sistem destinat facilitării tranzacţiilor bursiere, datorită componentei istorice (în ceea ce priveşte etapele tehnologice parcurse pentru dezvoltarea şi extinderea acestuia), a multitudinii de platforme *hardware* care coexistă în cele mai multe cazuri, a diversităţii de sisteme de operare folosite pe platforme diferite, aspectele intercomunicării între modulele sistemului devin extrem de stringente prin prisma acestei eterogenităţi.

6.1 Abordări arhitecturale

Indiferent de abordarea arhitecturală, ca o consecinţă directă a aspectelor prezentate anterior, componentele generice ale unui astfel de sistem informatic se identifică prin:

- un motor sistem, care joacă rolul de manager pentru toate ordinele introduse în sistem cu intenţia de a fi trimise către o anumită piaţă şi care, din punctul de vedere al intercomunicării dintre aplicaţii şi procese, va avea sarcina unui server de tranzacţii;
- un modul destinat asigurării înregistrării tuturor tranzacţiilor pe un suport persistent şi care, în coroborare cu serverul general de tranzacţii, va trebui de furnizeze mecanisme de recuperare a datelor şi de refacere a stării sistemului în cazul unor incidente hardware;
- o gamă diversă de aplicaţii client, destinate a răspunde variatelor nevoi legate de activitatea de *trading* desfăşurată de utilizatorii finali şi care trebuie să utilizeze o interfaţă (API – *Application Programming Interface*) de comunicare cu motorul sistem sau cu celelalte aplicaţii (funcţie de modelul arhitectural) comună pentru întreg sistemul: aplicaţii de plasare a ordinelor la bursă, ordin cu ordin (*single order trading*), aplicaţii pentru suportul *tradingului* în bloc/tranşe (*basket trading*), aplicaţii de evaluare a tendinţelor pieţei, fundamentate pe preţurile instrumentelor financiare furnizate de componente ca Reuters, TIBCO Rendenvous ş.a.
- multiple linii externe de comunicaţie, capabile să asigure conexiunea întregului sistem cu diversele pieţe pe care se intenţionează realizarea de tranzacţii.

Diversele abordări se diferenţiază prin felul în care se realizează comunicaţia între aceste module de bază, totul traducându-se în termeni de:

- viteză de răspuns a sistemului;
- parametri de securitate;
- toleranță la incidente neprevăzute;
- flexibilitate în întreținere și extensii ulterioare.

Un prim model, cel mai simplu conceptual, care realizează simpla conectare a modulelor componente ale sistemului într-o manieră liniară este ilustrat în ***Figura 6.1***.
Caracteristic acestui model este faptul că fiecare client se conectează la serverul dedicat gestionării ordinelor (*Order Management Server* - OMS) printr-o conexiune TCP/IP (*point-to-point*).

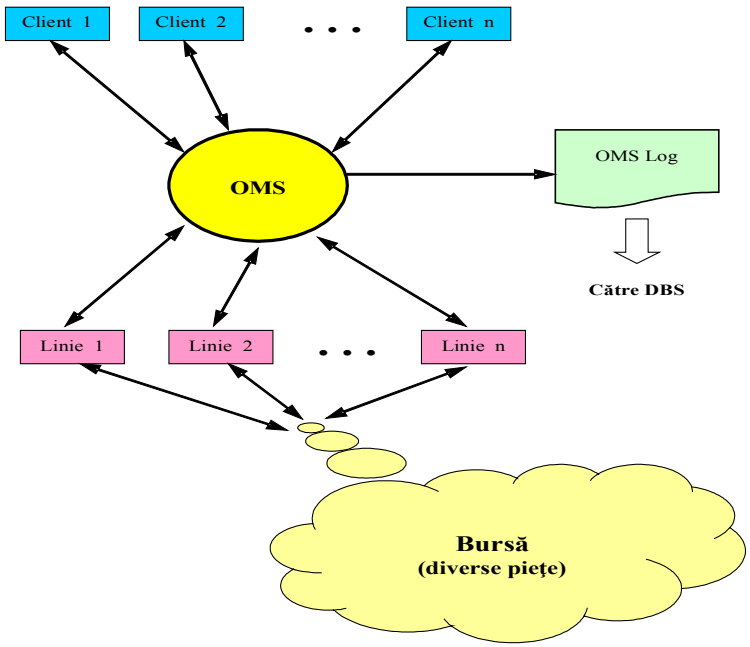

Figura 6.1 – Arhitectura simplificată a unui sistem de asistare a tranzacțiilor bursiere

În mod concret serverul creează o conexiune socket pentru fiecare client în parte - folosind o politică de parcurgere circulară a unei liste de conexiuni socket, prin care primește cereri de la fiecare client și trimite mesaje de răspuns tuturor clienților, pe rând, aceștia având sarcina de a-și filtra, în mod individual, mesajele primite de la server pentru a le considera doar pe cele care îi privesc.
Interacțiunea cu baza de date se poate realiza în mod direct, respectiv serverul (OMS) poate avea o conexiune directă cu un server de baze de date dedicat (IBM DB2, Sybase, Oracle, MySQL etc.), transmițându-i acestuia fiecare cerere pe care o primește de la clienți și toate răspunsurile trimise acestora.
În practică însă, datorită faptului că serverul de gestiune a ordinelor (OMS) și serverul de baze de date trebuie să ruleze pe diferite mașini fizice, și pentru a nu încărca fluxul principal de comunicație (scopul este întotdeauna trimiterea cât mai rapidă a ordinelor clienților către piață) cu datele trimise prin rețea către serverul de baze de date aflat la distanță, se adoptă un sistem simplu și robust, folosind două fișiere text aflate pe aceeași mașină fizică cu OMS, așa cum detaliază ***Figura 6.2***.

Serverul de gestiune a ordinelor (OMS) va scrie, înainte de orice prelucrare, cererea primită de la client în fișierul text `OMSLog`, aflat pe mașina locală. Se asigură în acest fel protecția datelor în cazul unei căderi a serverului și, totodată, se reduce substanțial timpul de înregistrare a datelor pe un suport persistent, nefiind necesară trimiterea lor la distanță și așteptarea de mesaje de confirmare a înregistrării. Din acest moment, din punctul de vedere al OMS-ului problema salvării datelor este rezolvată, el putându-se concentra asupra rezolvării concrete a cererii clientului și, eventual, trimiterii ordinului de vânzare/cumpărare către liniile de comunicație cu instituția bursieră.

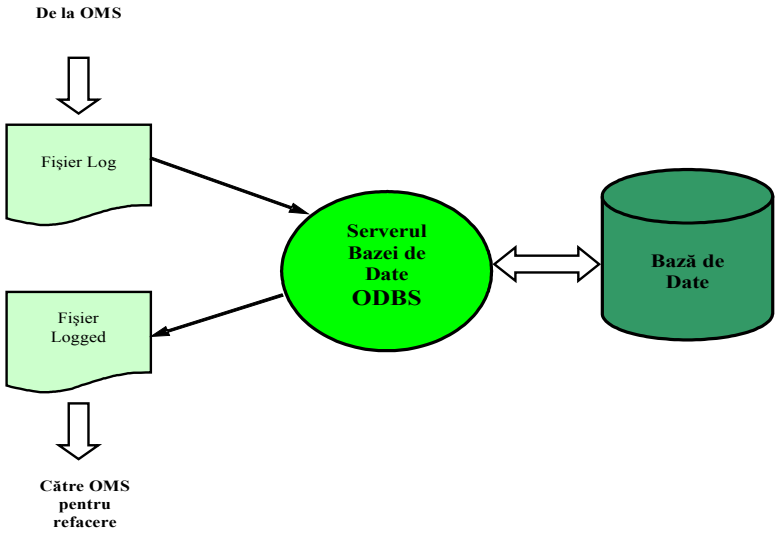

Figura 6.2 – Persistarea tranzacțiilor procesate de OMS

La acest stadiu al satisfacerii unei tranzacții intervine rolul unui al doilea server, care gestionează preluarea datelor din fișierul text `OMSLog` și transmiterea lor către o bază de date aflată la distanță. Prin acest server (ODBS – *Order DataBase Server*) se asigură o anumită paralelizare a fluxului de prelucrare asociat serverului principal OMS, care este în acest mod eliberat de sarcina de a aștepta mesaje de confirmare de la baza de date. În schimb, această sarcină îi este atribuită unui server specializat (ODBS) care, după primirea confirmării faptului că înregistrarea datelor s-a realizat cu succes în baza de date, duplică datele inițiale prin scrierea acestora într-un alt fișier text (`OMSLogged`) și verifică, pentru sincronizare, dacă acestea coincid cu cele scrise de OMS în fișierul `OMSLog`. Serverul bazei de date (ODBS) rulează pe aceeași mașină cu OMS, pe care se găsesc și cele două fișiere text sincronizate.
În această manieră, în cazul unei căderi a serverului principal (OMS), la inițializare, acesta trebuie să aibă un mecanism de consultare și verificare a datelor din cele două fișiere text detectând ce înregistrări nu au ajuns să fie scrise în baza de date (pentru că ODBS scrie în fișierul text corespondent `OMSLogged`, numai după ce a primit confirmarea înregistrării cu succes a articolelor în baza de date), fiind în măsură să refacă starea sistemului înainte de momentul căderii.

Comunicația dintre serverul principal și liniile dedicate piețelor se realizează în această arhitectură după același principiu. OMS deschide conexiune socket cu fiecare linie în parte

și le ascultă pe rând după aceeași politică *round-robin*. Serverul interpretează cererile clienților și, în cazul în care se solicită trimiterea unui ordin către o anumită piață, selectează linia de comunicație specifică (eventual poate alege una din mai multe linii specializate pentru o aceeași piață, pe baza unui algoritm de încărcare optimală a acestora) și trimite ordinul corespunzător către aceasta. Liniile de comunicație specifice diferitelor piețe (adesea la aceeași instituție bursieră) au rolul de a converti pachetele de date din formatul propriu al sistemului de *trading*, în formatul cerut pentru piața respectivă de către instituția bursieră, și reciproc.

Principala trăsătură a acestui model este faptul că fiecare componentă a sistemului comunică unu-la-unu cu o altă componentă cu care se află în conexiune directă și interfața prin care se realizează această comunicare (API) este de regulă unică în sistem. Este o arhitectură greoaie și inflexibilă pentru că fiecare modul trebuie să poarte cu sine această componentă de comunicație iar modificările în cadrul protocolului și a structurilor de date folosite au impact asupra tuturor componentelor sistemului, de cele mai multe ori acestea fiind implementate în diferite limbaje de programare, rulând pe diferite platforme.

6.2 Modele de API

Așa cum am menționat anterior, interfața care se pune la dispoziția aplicațiilor pentru realizarea comunicării poartă termenul generic de API (*Application Programming Interface*). Acest API poate avea diferite niveluri de complexitate și comportamente foarte variate, funcție de scopurile urmărite **[STE1]**. Putem face o distincție generică între API-uri utilizate pentru transmisii în timp real și API-uri folosite pentru prelucrările în loturi (*batch*). Ceea ce este interesant în această încercare de clasificare este faptul că în ambele situații pot fi manipulate aceleași structuri de date, aspect care face posibilă proiectarea și reutilizarea unor componente de API, indiferent de maniera de abordare a comunicării. Prezint, ca exemplu, un fișier de date în format ASCII care are structura de mai jos.

200009251520000008
^~SYSTEM_1~~001~1.00.0505.202~~003~1.00.0505.202~~004~SELL~~005~20000.00~~006~*VOWR
D0O~~007~10.0000000~~009~020026001~~011~890700901~~017~20000505~~018~20000510~~029~20
0000.00~~038~GBP~~039~1.57~~099~1301~~010~gseso~~014~1234~~015~5678~~016~GSJL~~019~0
~~020~OpenTrade~~028~NY~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.1825~~0
33~PUT~~006~9ED623378~~040~sepOption~~036~1~~041~7~~400~1~^
^~SYSTEM_1~~001~1.00.0505.210~~003~1.00.0505.210~~004~BUY~~005~60000.00~~006~*VOWR
D0O~~007~10.0000000~~009~020026001~~011~890700901~~017~20000505~~018~20000510~~029~60
0000.00~~038~GBP~~039~1.57~~099~1301~~010~gseso~~014~1234~~015~5678~~016~GSJL~~019~0
~~020~CloseTrade~~028~NY~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.1825~~0
33~PUT~~006~9ED623378~~040~sepOption~~036~1~~041~8~~400~1~^
^~SYSTEM_1~~001~1.00.0505.214~~003~1.00.0505.214~~004~BUY~~005~12000.00~~006~*VOWR
D0O~~007~10.0000000~~009~020026001~~011~890700901~~017~20000505~~018~20000510~~029~12
0000.00~~038~GBP~~039~1.57~~099~1301~~010~gseso~~014~1234~~015~5678~~016~GSJL~~019~0
~~020~CloseTrade~~028~NY~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.1825~~0
33~PUT~~006~9ED623378~~040~sepOption~~036~1~~041~9~~400~1~^
^~SYSTEM_1~~001~1.00.0505.229~~003~1.00.0505.229~~004~SELL~~005~120000.00~~006~*VOW
FD0O~~007~10~~009~020026001~~011~890700901~~017~20000504~~018~20000509~~029~1200000.
00~~038~GBP~~039~1.57~~099~0001~~010~cobbn~~014~1234~~015~5678~~016~GSJL~~019~0~~02
0~OpenTrade~~028~LD~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.7185~~033~C
ALL~~006~9ED622693~~040~sepOption~~036~1~~041~10~~400~1~^

**^~SYSTEM_1~~001~1.00.0505.228~~003~1.00.0505.228~~004~SELL~~005~120000.00~~006~*VOW
FD0O~~007~10~~009~020026001~~011~890700901~~017~20000504~~018~20000509~~029~1200000.
00~~038~GBP~~039~1.57~~099~0001~~010~cobbn~~014~1234~~015~5678~~016~GSJL~~019~0~~02
0~OpenTrade~~028~LD~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.7185~~033~C
ALL~~006~9ED622693~~040~sepOption~~036~1~~041~11~~400~1~^
999999999999000005**

Acest fișier conține informațiile privitoare la 5 tranzacții bursiere, deci conține, în ultimă instanță, 5 articole ale unei tabele dintr-o bază de date, sau 5 cinci instanțe ale unui tip de structură de date care pot fi păstrate în memoria calculatorului. Fiecare articol conține câmpuri care sunt descrise de perechea:

~~<etichetă câmp>~~<valoare câmp>~~

Transmiterea acestui fișier de către o aplicație către o alta prin intermediul serviciului FTP presupune faptul că aplicație destinatar trebuie să implementeze un mecanism prin care să fie capabilă sa detecteze faptul că i-a fost trimis un nou fișier spre prelucrare, să prelucreze acest fișier primit sub forma unui lot de tranzacții și să genereze un răspuns pentru aplicația expeditor care va fi tot un fișier ce va fi trimis expeditorului originar tot prin intermediul serviciului FTP. Fișierul de răspuns poate avea formatul de mai jos.

200009251626000252
^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.202 ~~400~1~^
^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.210 ~~400~1~^
^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.214 ~~400~1~^
^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.229 ~~400~1~^
^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.228 ~~400~1~^
999999999999000005

Chiar dacă modelul de API nu realizează o transmisie în timp real, dar structurile de date manipulate sunt în mod esențial similare cu cele care s-ar utiliza în situația în care transferul ar fi realizat în timp real.
Dacă aplicația care transferă aceste structuri de date către o altă aplicație realizează o conexiune *socket* TCP (*point-to-point*) cu aplicația destinatar, atunci fiecare articol (structură de date), care în situația precedentă era scris în fișier, poate fi transmis prin intermediul conexiunii *socket* direct către destinatar (sub forma unei cereri de prelucrare a respectivei tranzacții – *call forward*, realizat de obicei de către o aplicație client către un server) **[STE1][STE2]**.

**^~SYSTEM_1~~001~1.00.0505.202~~003~1.00.0505.202~~004~SELL~~005~20000.00~~006~*VOWR
D0O~~007~10.0000000~~009~020026001~~011~890700901~~017~20000505~~018~20000510~~029~20
0000.00~~038~GBP~~039~1.57~~099~1301~~010~gseso~~014~1234~~015~5678~~016~GSJL~~019~0
~~020~OpenTrade~~028~NY~~030~0621200012:00:00:000AM~~031~EUROPEAN~~032~3.1825~~0
33~PUT~~006~9ED623378~~040~sepOption~~036~1~~041~7~~400~1~^**

Aplicația care primește mesajul (cu rol de *server* în acest context) îl va prelucra și va trimite prin intermediul conexiunii *socket* un mesaj de răspuns aplicației care a solicitat prelucrarea tranzacției, sub forma unui *call back*.

^~SYSTEM_2~05~~001~A~~002~LFI-IDF~~003~1.00.0505.202 ~~400~1~^

În această a doua situație, prelucrarea tranzacțiilor de către server se realizează pe măsură ce acestea se produc, iar interfața de comunicare este sensibil diferită. Cu toate acestea, datele esențiale se regăsesc, în ultimă instanță, în același format. Acest lucru face posibilă stocarea și manipularea acestor date într-o manieră uniformă, standardizată pentru un anumit sistem dat, fapt ce-i mărește acestuia din urmă flexibilitatea.
Mesajele transmise între aplicații pot avea un format fix, predefinit și specific, de genul tipurilor de structuri care se pot defini în mod nativ în limbajele de programare de nivel înalt (API static), sau pot aveau un format variabil, cum se poate observa în exemplu de mai sus (API auto-descriptiv). Acest gen de API permite formarea de mesaje în mod dinamic, prin crearea de câmpuri definite prin tripletul:

<nume câmp> <tip dată> <valoare>

În exemplul prezentat anterior, specificarea tipului de date în cadrul mesajului nu s-a dovedit necesară datorită faptului că mesajele sunt prelucrate de ambele aplicații ca șiruri de caractere, decelarea tipului de dată conținut de fiecare câmp realizându-se la un nivel superior al aplicațiilor în cauză, pe baza unor tabele de asociere (*hash table*) între eticheta câmpului și tipul acestuia.

6.3 Soluții arhitecturale

Am prezentat anterior componentele esențiale ale unui sistem informatic distribuit de asistarea a tranzacțiilor bursiere, indiferent de arhitectura pentru care se optează.
Vom rafina în continuare prezentarea diverselor arhitecturi care pot fi luate în considerare, reliefând diferențele dintre ele, avantajele și dezavantajele implementării unei arhitecturi sau a alteia.

6.3.1 Arhitecturi de sisteme cu module puternic interdependente (arhitecturi statice)

Un exemplu care își găsește frecvente implementări practice și care reprezintă o variațiune pe modelul arhitectural generic prezentat anterior, este ilustrat în ***Figura 6.3***.
Această arhitectură se încadrează în categoria abordărilor arhitecturale care presupun o strânsă interdependență între modulele sistemului, datorită faptului că vehicularea mesajelor se realizează în mod fundamental prin deschiderea de conexiuni *socket* care sunt destinate să prelucreze tipuri dedicate de mesaje, la care atât partea care transmite cât și partea care recepționează aceste mesaje (în multe situații același modul joacă ambele roluri, în diferite contexte) trebuie să adere la o paradigmă care nu permite altă posibilitate de interconectare în interiorul unei astfel de arhitecturi.

Comparând această arhitectură cu cea prezentată anterior, îmbunătățirea constă în realizarea unui nivel mai înalt de flexibilitate prin separarea atribuțiilor serverului unic (OMS) de gestiune a cererilor clienților și de direcționare a ordinelor specifice către liniile de comunicație cu bursa, introducându-se două servere distincte, specializare fiecare în satisfacerea de cereri grupate pe categorii de clienți.

Un server (OMS) va gestiona cererile aplicațiilor client asociate utilizatorilor sistemului (traderi, brokeri) și un alt server (*Market Gateway Server* - MGS) va fi însărcinat cu managementul comunicării cu piața, prin linii specializate.

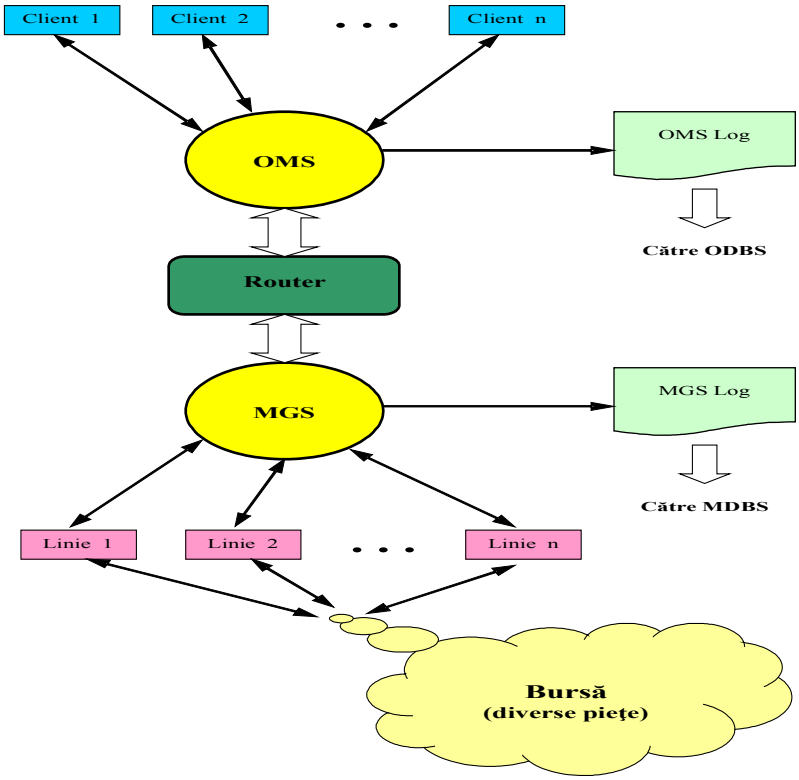

Figura 6.3 – Arhitectură cu module puternic interconectate

Arhitectura internă a celor două servere este similară, fiecare însă manipulând structuri de date specifice comunicației cu clienții de care răspunde.
Pasarea mesajelor între cele două zone, separate din punct de vedere logic și funcțional, se realizează prin intermediul unui program de rutare a pachetelor (numit generic *Router*).
Prin acest model arhitectural se introduce o întârziere în trimiterea ordinelor către piață însă această întârziere este neglijabilă comparativ cu avantajele ce decurg din descentralizarea modelului – modularitate, flexibilitate în întreținere și posibilități de extindere.
Nu vom insista asupra acestui model. Principial, acest model presupune ca fiecare modul component să fie proiectat să comunice cu un anumit modul sau module, și aceasta se realizează încă din faza de proiectare a modului, întreaga sa arhitectură internă decurgând și fiind subordonată rolului pe care urmează să îl joace printre celelalte module ale sistemului. Aceasta presupune că, în situația în care ulterior se dorește interconectarea sa cu alte module decât cele cu care a fost proiectat să interacționeze, impactul va fi nu numai de natura unor adaptări la nivelul API-ului de comunicație, dar însăși fluxul prelucrării, întreaga sa logică de funcționare va trebui reabordată. Această trăsătură îi conferă arhitecturii o anumită greutate și închistare, fiecare modul având o filosofie specifică de comunicare cu modulele cu care a fost intenționat să comunice și de aici o puternică

interconectare a acestora, fapt care nu este de natură să faciliteze eventuale extinderi ulterioare.

6.3.2 Arhitecturi cu sisteme deschise – componente branșabile dinamic (arhitecturi dinamice)

O abordare fundamental diferită constă în ideea de nu avea, în mod necesar, o conexiune unu-la-unu (TCP), între fiecare din componentele sistemului, ci posibilitatea ca fiecare din ele să comunice cu toate celelalte deodată, fiecare componentă interpretând mesajele primite în funcție de rolul specific pe care îl joacă în sistem. Acest model este ilustrat în ***Figura 6.4***. În locul unei abordări *liniare* a fluxului de date între aplicațiile client și liniile de comunicație cu bursa, se introduce flexibilitatea unei concepții de comunicare unu-la-toți sau unu-la-mai-mulți, fiecare modul putând fi conectat sau debranșat, în funcție de nevoile situației concrete în care se află sistemul.

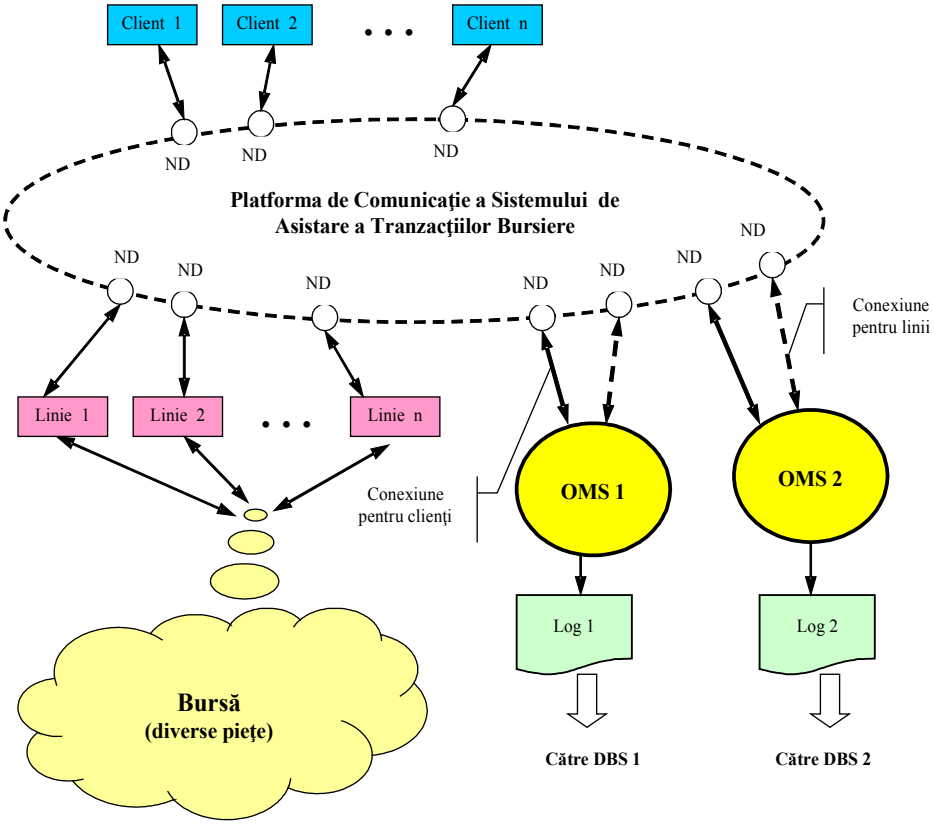

Figura 6.4 – Arhitectură cu module interconectabile în mod dinamic

Elementul esențial al acestei arhitecturi îl constituie posibilitate de *broadcasting* (poate fi realizată și o implementare *multicasting*) a mesajelor fiecărui modul către toate celelalte module, asigurându-se, în același timp, mecanisme de control al transmisiei și de retransmitere a eventualelor pachetelor pierdute. Se folosesc în acest model ambele protocoale – UDP și TCP – permițându-se o comunicare rapidă și asigurându-se o arhitectură extrem de flexibilă. Componenta de comunicație (*middleware*) este separată

integral de modulele funcționale și funcționează ca un strat intermediar între nivelul de comunicare realizat prin TCP/UDP *sockets* și nivelul aplicațiilor utilizator. Vom denumi în continuare această componentă drept platformă de comunicație (*Network Deamon* – ND).

Această platformă de comunicație asigură conectarea tuturor aplicațiilor care rulează pe o mașină din rețea la sistemul integrat de *trading*, fiind la rândul său o aplicație implementată într-o manieră care să permită portarea ei pe multiple sisteme de operare, ce ar putea fi utilizate în condiții concrete. Platforma de comunicație respectă caracteristicile paradigmei *Publisher–Subscriber*, conform căreia fiecare aplicație conectată în sistem prin intermediul acestei platforme va subscrie la un anumit set de mesaje care vor fi în acest fel publicate pentru ea de către alte aplicații din sistem, și pe care aplicația în cauză le va consuma (prelucra) pe măsură ce le primește, transmițând (publicând) la rândul ei mesaje către celelalte aplicații ale sistemului, prin intermediul aceleași platforme de comunicație.
Pentru a fi performantă, această aplicație (*Network Deamon*), care realizează în fapt o platformă de comunicație, trebuie să aibă fire de execuție distincte pentru ascultarea rețelei, pentru primirea și prelucrarea mesajelor recepționate de la aplicațiile care rulează pe mașina locală și pentru broadcast în rețeaua logică a sistemului distribuit, către celelalte module componente ale acestuia din urmă **[STE2][LEW]**.

Principial, modulele sistemului își mențin funcționalitatea. Cele două servere de tranzacții ilustrează diferența fundamentală în privința flexibilității unui astfel de model arhitectural: posibilitatea conectării mai multor servere de gestiune a tranzacțiilor, în același timp, asigurându-se o repartizare a încărcării pe mai multe componente cu aceeași funcționalitate, pe de o parte, și realizarea unui mecanism de prevenire a căderii întregului sistem în cazul unei disfuncționalități a unuia dintre servere – celelalte servere preluând în mod consistent atribuțiile acestuia – pe de altă parte. Trebuie remarcat că fiecare din cele două OMS poate fi conceput ca o *baterie de servere,* după modelul prezentat în arhitectura liniară, cele două conexiuni cu platforma de comunicație deservesc, pe de o parte, serverul specializat în tranzacții cu aplicațiile utilizator (OMS) iar, pe de altă parte, serverul dedicat transmiterii ordinelor către liniile de comunicație cu bursa (MGS).

O rafinare în plus se poate realiza prin disocierea întregului flux de mesaje în două sub-rețele distincte (două segmente de rețea), așa cum ilustrează ***Figura 6.5***. Prin această separare se previne încărcarea excesivă a unei rețele sistem unice, prin broadcast-ul realizat de toate modulele implicate. În acest model posibilitatea înlocuirii fiecărui server cu o baterie de servere specializate devine și mai pregnantă. Poate fi folosit același model de comunicare cu baza de date ca și cel descris în modelul liniar sau, datorită facilității de broadcast*,* se poate realiza o arhitectură în stea, în jurul unei bazei de date, fiecare componentă a sistemului având posibilitatea de a accesa în mod nemijlocit baza de date. Consistența accesului se asigură printr-o politică de blocare a înregistrărilor și deblocare a acestora după modificarea câmpurilor avute în vedere (***Figura 6.6***). Trebuie însă avute în vedere mecanisme de control și rezolvare a situațiilor de *deadlock* la accesarea bazei de date.

Modelul prezentat în ***Figura 6.3*** diferă esențial de modelul din ***Figura 6.6*** în ceea ce privește filosofia de transmitere a mesajelor între module. Datorită posibilității accesului nemediat al tuturor componentelor sistemului la o bază de date comună, nu mai este necesar un broadcast de pachete care să conțină întreaga informație utilă legată de o tranzacție, ci este suficient să se trimită un pachet minimal conținând o cheie de regăsire a articolului vizat în baza de date – componenta interesată în acea tranzacție urmând a prelua

din baza de date întreaga informaţie necesară. Modelul prezintă avantajul unei simplificări generale a arhitecturii însă întregul flux de comunicaţie este concentrat în jurul acestei baze de date unice, fapt care poate genera strangulări în funcţionare şi timpi de răspuns în general mai mari.

Totuşi, dacă pentru anumite instrumente bursiere activitatea de *trading* nu se bazează atât de mult pe un timp de răspuns extrem de mic (diferenţele pot fi de ordinul secundelor sau zecimilor de secunde între cele două abordări, funcţie de performanţa componentei de stocare persistentă), cum ar fi cazul tranzacţiilor de instrumente *futures* şi *options,* atunci modelul în stea, în jurul unei bazei de date centrale, poate fi o soluţie mai facil de implementat, de menţinut şi cu un nivel ridicat de flexibilitate.

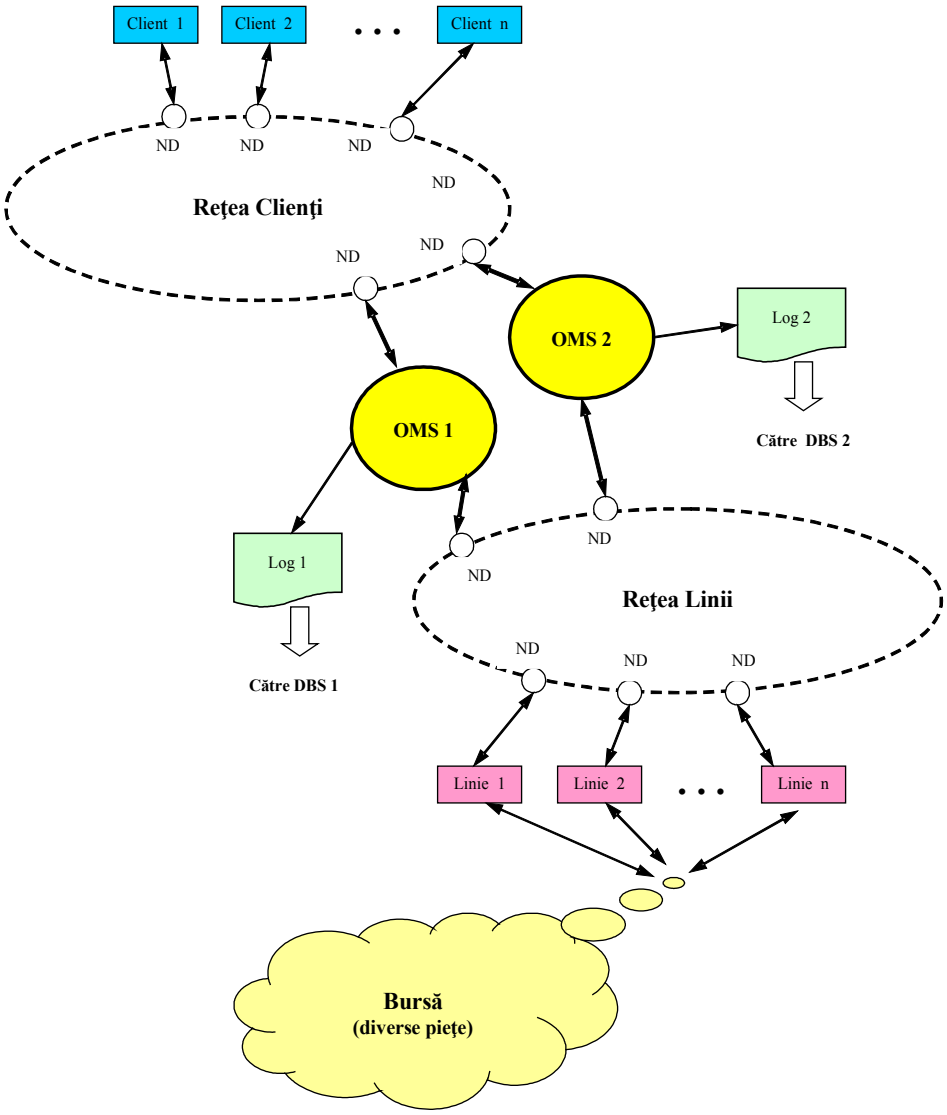

Figura 6.5 – Arhitectură cu broadcast în subreţele distincte

Această soluţie arhitecturală permite o mare flexibilitate în configurarea în mod dinamic a sistemului, funcţie de necesităţi. Dacă încărcarea serverul de procesarea a tranzacţiilor este

prea mare, atrăgând după sine un timp de răspuns la cererile clienților nesatisfăcător, simpla conectarea, în mod dinamic, fără oprirea prelucrării, a unuia sau mai multor servere este de cele mai multe ori o soluție la îndemână, elegantă și care poate fi la rândul ei automatizată prin intermediul unor componente de monitorizare a traficului în rețeaua sistemului și de detectare (eventual ajustare, prin acțiuni corespunzătoare) a timpului de răspuns a sistemului în ansamblul său.
De asemenea, proiectarea și integrarea unor noi module implică resurse și efort sensibil reduse, întrucât platforma de comunicație este standardizată și disponibilă pentru orice nouă implementare; noua componentă va avea identificatorul ei unic în sistem, iar modulele care vor fi implicate în comunicarea cu această nouă componentă nu trebuie să știe decât acest identificator unic al aplicației (modulului) și mesajele vor fi direcționate, într-o manieră transparentă, către această nouă componentă și viceversa.

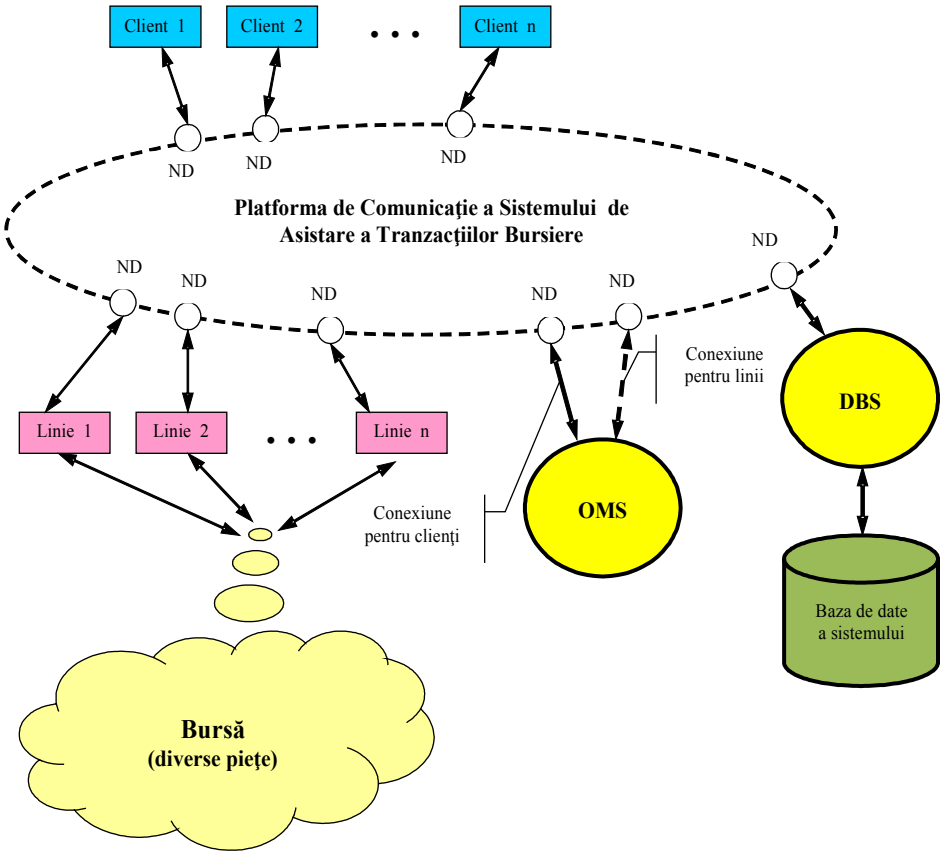

Figura 6.6 – Arhitectură deschisă cu bază de date unică

Avantajele și deschiderile unei astfel de arhitecturi sunt imediate: mesajele vehiculate având un format standardizat, conținutul acestora este decelat de modulul receptor numai dacă a fost *informat* despre existența mesajului respectiv, altfel mesajul este ignorat (o filtrare a mesajelor la recepție pentru toate modulele componente ale sistemului).

Ca tendință generală, din punctul de vedere al arhitecturii de ansamblu al unui sistem informatic destinat asistării tranzacțiilor bursiere, cele prezentate până la acest moment trasează liniile generale ale manierei în care este abordată problema unui sistem integrat de

trading într-o firmă specializată în efectuarea unor astfel de tranzacţii. Principial, considerentele de viteză sunt primordiale însă, la o implementare atentă a oricăruia dintre modelele prezentate, se constată o uniformizare generală a timpilor de răspuns, astfel încât aspectele legate de flexibilitatea arhitecturii, de satisfacerea nevoilor concrete ale unui client particular, au ajuns în prezent să prevaleze. Cu toate că modelul liniar are implementări performante, incluzând mecanisme de conectare a mai multor servere de tranzacţii în paralel, pentru asigurarea continuităţii activităţii tranzacţionale în cazul în care se produce o cădere *hardware* (un aşa numit mecanism *hot-standby*, prin care fiecare server – primar - este dublat de un al doilea – secundar - pregătit în orice moment să preia sarcina serverului primar în cazul căderii acestuia) acest model prezintă limitări în ceea ce priveşte extinderile dinamice, ce se pot realiza pe parcursul desfăşurării *tradingului*, funcţie de încărcarea sistemului.

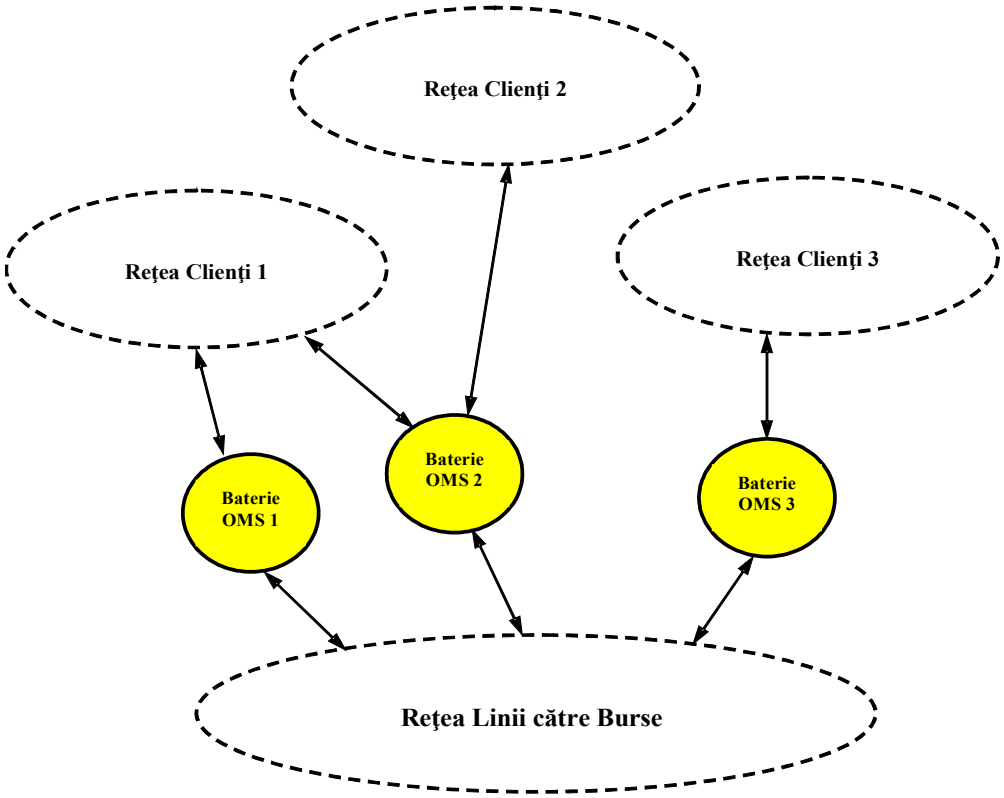

Figura 6.7 – Flexibilitatea de interconectare a componentelor într-o arhitectură deschisă

De asemenea, facilitatea de *broadcasting* elimină protocoluri specifice de comunicaţie între diferite componente particulare, pachetele ce se vehiculează având o structură standardizată, fiecare aplicaţie extrăgându-şi componenta de date în care este interesată. Posibilităţile de interconectare a componentelor permit configuraţii care pot veni în întâmpinarea unei largi palete de contexte reale, aşa cum ilustrează ***Figura 6.7***.

Bateriile de servere pot satisface subreţele separate, atât la nivel logic cât şi fizic (segmente diferite de reţea), permiţând o separare a utilizatorilor pe clase de aplicaţii şi tipuri de activităţi tranzacţionale, obţinându-se în acest fel atât îmbunătăţiri în ceea ce priveşte încărcarea reţelei cât şi mecanisme de securitate şi de acces la date mult mai uşor

de controlat și întreținut. Principial, se poate *alimenta* cu mesaje pentru piață o subrețea unică de linii de conectare la instituția bursieră sau, în cazul unei multitudini de linii conectate la piețe diferite, se pot face rafinări arhitecturale și la acest nivel, avantajul fiind că sistemul poate fi configurat dinamic, fără un efort suplimentar de proiectare și implementare de componente auxiliare necesare interconectării.