SERVER QNX/CORBA DE SUPERVIZARE A UNUI MODUL DE INTREPRINDERE VIRTUALĂ

conf. dr. ing. Daniela Saru
ing. Adrian Petcu
ş.l. dr. ing. Daniel Merezeanu

Universitatea "Politehnica" Bucureşti
saru@aii.pub.ro, pady@ss.pub.ro

Rezumat: Lucrarea prezintă problemele legate de proiectarea şi implementarea unui server cu dublu rol - supervizor şi de integrare software – pentru un modul funcţional al unei întreprinderi virtuale. Tehnologia software utilizată a avut în vedere caracteristicile de funcţionare specifice, de timp real, ale modulului supervizat şi necesitatea colaborării acestuia cu alte module ale întreprinderii virtuale, situate în locaţii fizice distribuite geografic şi reprezentând, din punct de vedere software, aplicaţii eterogene. Serverul a fost realizat în limbaj C++, sub sistem de operare QNX, folosind o implementare *middleware* [3] orientată obiect performantă, de tip CORBA. El poate fi utilizat prin intermediul oricărei reţele de comunicaţie ce foloseşte protocol TCP/IP de către module client ce se execută sub sistem de operare Windows, QNX etc., cu condiţia ca aceste module client să cunoască interfaţa IDL OMG ce descrie funcţionalitatea serverului. Un astfel de modul client, cu rol de panou de comandă ce include o interfaţă utilizator ergonomică a fost descris în [9].

Cuvinte cheie: întreprindere virtuală, celulă flexibilă de fabricaţie, arhitectură client/server, sisteme distribuite eterogene, EIA (Enterprise Application Integration), timp real, QNX, middleware, CORBA, ORBACUS/E.

1. Introducere

Integrarea aplicaţiilor software în cadrul unei întreprinderi sau între mai multe întreprinderi care colaborează este un subiect de mare actualitate. Denumită în limbajul de specialitate EIA *(Enterprise Application Integration)*, integrarea aplicaţiilor software de întreprindere permite coordonarea şi sincronizarea mai multor aplicaţii eterogene atât în interiorul, cât şi în afara întreprinderilor. Conceptele de B2B *(Bussines to Bussines* – schimb comercial prin Internet), CRM (*Customer Relationship Management* – gestionarea relaţiilor cu clienţii) şi Internet, necesită acest liant [10].

EAI reprezintă, de fapt, noul stil de lucru în domeniul software. Întreprinderile au din ce în ce mai puţini informaticieni care concep şi scriu aplicaţii şi din ce în ce mai mulţi care integrează aplicaţii. Entitatea ce trebuie integrată nu mai este un obiect sau o componentă software, ci este o aplicaţie software. Prin EAI, sistemele informatice ale întreprinderilor se mulează din ce în ce mai bine pe structura procesului de afaceri. Complexitatea problemelor legate de infrastructura informatică creşte şi mai mult în cazul unei întreprinderi virtuale, formată din module (secţii, departamente, birouri etc.) cu funcţionalitate extrem de diversă şi grad de dispersie geografică oricât de mare. Granularitatea modulelor se poate situa pe o scară foarte cuprinzătoare, depinzând în mare masură atât de specificul domeniului de activitate, cât şi de posibilităţtile de organizare ale întreprinderii respective.

Tehnologiile care permit realizarea integrării aplicaţiilor în cadrul unei întreprinderi constituie ceea ce se numeşte în limbajul de specialitate *middleware* [3]. Acest cuvânt american a fost creat pentru a desemna un nivel software intermediar situat între sistemul de operare al calculatorului şi programele de aplicaţii. Deşi este un termen utilizat pentru a desemna mai multe subcategorii de tehnologii, în prezent, pentru realizarea unei aplicaţii formate din mai multe componente distribuite se utilizează cu precădere fie tehnologiile obiect reprezentate în standardele CORBA sau DCOM, fie tehnologia Java şi conceptul de componentă Java Bean. Cu ajutorul acestor tehnologii se realizează cele mai multe servere de aplicaţii din categoria *mission-critical* (ce trebuie sa satisfacă cerinţe severe de performantă şi disponibilitate etc.) [10].

Folosirea tehnologiilor *middleware* în cadrul EAI este posibilă datorită existentei următoarelor două elemente:

- ***Conceptul de interfaţă*** care caracterizează fiecare componentă conectată la magistrala *middleware* – permite descrierea ansamblului de servicii oferite de către infrastructură; în cadrul EAI, interfeţele sunt construite la nivelul adaptoarelor asociate aplicaţiilor;
- ***Metodologia orientată obiect*** – permite construirea unui model de interfeţe, care să răspundă necesităţilor utilizatorilor: prin aceasta, contribuie la reducerea nivelului de complexitate necesar creării şi gestionării sistemelor distribuite [8].

Toate aceste caracteristici ne-au determinat ca, în proiectarea soluţiei software de supervizare şi de integrare a modulului de întreprindere virtuală ales să folosim tehnologia CORBA, prin intermediul unui produs special conceput pentru aplicaţii de timp real şi sisteme încapsulate (*embedded systems*) – ORBACUS/E [4]. Modulul inclus în întreprinderea virtuală este o celulă flexibilă de fabricaţie de tip educaţional, DEGEM 2000 [2], folosită ca un prim pas în vederea valorificării ulterioare a experienţei astfel dobândite în cadrul unui sistem industrial.

Sistemul software conceput şi testat include un server de timp real (QNX), ataşat platformei celulei flexibile şi un modul client, cu rol de panou de comandă, ce poate fi instalat pe un sistem Windows. Server-ul şi clientul pot interacţiona prin intermediul oricărei reţele de comunicaţie ce foloseşte protocol TCP/IP. Sistemul software poate fi extins prin crearea altor aplicaţii client, de acelaşi tip cu cel realizat de noi [9] sau cu funcţionalitate diferită, cu condiţia de a cunoaşte interfaţa IDL OMG [11] a serverului. Aplicaţiile client pot fi realizate în limbaje diferite şi se pot executa sub sisteme de operare diferite, acesta fiind unul dintre avantajele integrării oferite de către utilizarea unei tehnologii de tip middleware.

Deoarece serverul a fost conceput special pentru a lucra sub sistemul de operare QNX, experienţa dobândită în cadrul proiectului poate fi folosită şi pentru integrarea altor module de fabricaţie virtuală, monitorizate prin sisteme software "încapsulate" (*embedded systems*), QNX oferind un suport deosebit de performant în acest sens [7]. Modulele respective pot fi, de exemplu, destinate realizării unor sarcini în mediu de lucru periculos sau nociv şi monitorizate de la distanţă prin intermediul unui sistem software similar cu cel realizat de noi.

Cu toate că principalul scop al realizării aplicaţiei a fost studierea mecanismelor de integrare a componentelor în cadrul unei întreprinderi virtuale şi oferirea unei soluţii viabile pentru activitatea industrială, rezultatul obţinut poate fi utilizat cu succes şi în cadrul activităţii educaţionale de tip clasic sau la distanţă, inclusiv în cadrul unor cursuri de perfecţionare a personalului întreprinderilor. Prin supravegherea celulei flexibile de fabricaţie cu ajutorul unei camere video ce poate fi conectată la Internet (*Web cam*) şi includerea unui modul software dedicat, aplicaţia permite cursanţilor să vizualizeze efectul comenzilor pe care le furnizează de la distanţă, serverul fiind astfel proiectat încât să nu accepte transmiterea unor comenzi conflictuale.

2. Scurtă descriere a tehnologiilor şi instrumentelor software utilizate

2.1 Tehnologia CORBA

CORBA defineşte o arhitectură şi o infrastructură deschisă, independentă de dezvoltator, care poate fi folosită de diverse aplicaţii pentru a colabora în cadrul unei reţele de comunicaţie [1]. Folosind protocolul standard IIOP (*Internet Inter-ORB Protocol*), aplicaţii implementate în limbaje diferite, ce se execută pe platforme diferite (hardware, sistem de operare, tip de reţea) şi având la bază produse software conforme CORBA, realizate de către dezvoltatori diferiţi, pot alcătui sisteme software performante, cu grad de complexitate ridicat.

Datorită simplităţii cu care sunt integrate platforme atât de diverse, de la sisteme mainframe la sisteme desktop sau handheld şi sisteme încapsulate (*embedded systems*), cu o gamă largă de producători, standardul CORBA a devenit alegerea preferată a multor profesionişti din domeniul informatic, fiind folosit pentru crearea unor aplicaţii cu destinaţii şi dimensiuni extrem de variate [11]. De exemplu, una dintre utilizările cele mai importante şi mai frecvente este în cadrul serverelor care trebuie să suporte cu succes solicitări din partea unui număr foarte mare de clienţi. Anumite versiuni specializate de CORBA rulează pe sisteme de timp real, caz în care viteza şi fiabilitatea sunt cerinţe ce trebuie respectate cu mare rigurozitate.

Modelul de interacţiune şi comunicaţie CORBA este de tip client/server [5], orientat obiect. El presupune existenţa unui obiect software (servant), care asigură anumite servicii şi căruia i se asociaza cel putin o interfaţă, reprezentând definiţia sintactică a contractului pe care acest obiect (de tip server) îl oferă clienţilor săi. Pentru specificarea interfeţelor se foloseşte un limbaj standardizat numit IDL (*Interface Definition Language*) [2]. IDL este independent de orice limbaj de programare, dar se *mapp*-ează ("se traduce") în cele mai folosite limbaje de programare existente (C, C++, Java, COBOL, Smalltalk, Ada, Lisp, PL/1, Python etc.) Magistrala software de comunicaţie este organizată în jurul "miezului" facilităţilor CORBA, numit *Object Request Broker* (ORB) [1]. ORB preia cererile venite prin reţea şi le comunică servantului sub forma unor apeluri de metode. Aplicaţiile client şi server se conectează la ORB prin intermediul unor componente numite *stub* şi, respectiv, *skeleton*, generate automat prin compilarea interfeţei IDL. Acestea realizează operaţiile de *marshalling/unmarshalling* necesare comunicării transparente prin reţea [11][9].

2.2 ORBACUS/E

Alegerea unui anumite implementări CORBA ia în calcul, în special, considerente de performanţă, de preţ sau de licenţă de utilizare. În acest context, am considerat că produsul ORBACUS, dezvoltat de *Object Oriented Concepts* (*OOC*), oferă o flexibilitate sporită, permiţând şi utilizarea gratuită în scopuri necomerciale [4]. ORBACUS este disponibil pentru Microsoft Windows ™ şi pentru o gamă largă de implementări Unix/Linux. Pentru proiectul realizat, am folosit o versiune special proiectată pentru a rula pe sisteme încapsulate (*embedded systems*) şi de timp real (de exemplu, QNX), numită ORBACUS/E, ce permite utilizarea limbajelor de

programare C++ şi Java. Ca mod de distribuţie a acestui produs, dezvoltatorii au ales soluţia *open source*, astfel încât să poată fi compilat şi instalat conform preferinţelor fiecărui utilizator (caracteristică extrem de avantajoasă în cazul aplicaţiei pe care urma să o realizăm).

În timp ce varianta ORBACUS standard permite crearea unor aplicaţii fără cerinţe speciale de viteză sau timp de reacţie, asigurând suport pentru servicii CORBA suplimentare, ORBACUS/E este proiectat după principiul *real-time ready*: oferă un set de servicii esenţiale, are preformanţe de viteză foarte bune (o întârziere de numai 20% faţă de programarea TCP/IP la nivel de socket) şi este foarte compact (are dimensiuni reduse). Varianta pentru C++ este preferabilă pentru aplicaţii de telecomunicaţii, cum ar fi *Voice-over-IP*, în timp ce implementarea pentru Java se potriveşte perfect mediilor restrânse ca posibilităţi, cum ar fi calculatoare *hand-held* sau dispozitive *wireless*.

2.3 QNX Neutrino RTOS

În cadrul sistemului software ce urma a fi proiectat, serverul de supervizare a modulului de întreprindere virtuală trebuia să îndeplinească o dublă sarcină de comunicare: cu modulul client şi cu automatul programabil, asociat celulei flexibile de fabricaţie [9]. Din acest motiv, cele mai importante caracteristici ale platformei pe care acest server urma să se execute erau fiabilitatea şi viteza ridicată de răspuns. În plus, driver-ul de comunicaţie - preluat dintr-un proiect anterior – folosit de server pentru a da comenzi automatului programabil, fusese proiectat pentru a funcţiona sub sistem de operare QNX. Toate aceste caracteristici ne-au determinat să alegem ca platformă server un sistem de operare în timp real, creat de către *QNX Software Systems*, numit QNX Neutrino RTOS [6][7].

Fiind compus dintr-un micronucleu (microkernel) şi o serie de module optionale, QNX Neutrino RTOS realizează o diviziune eficientă a sarcinilor, care îi permite să asigure realizarea performantaă a tuturor serviciilor sistemului. Cu toate că foloseşte ca platformă hardware sisteme PC standard, are latenţă scăzută şi surclasează multe sisteme performante, dar scumpe. Pentru fiecare serviciu de sistem pe care aplicaţiile îl solicită, execuţia codului nucleu (kernel) asociat este foarte rapidă, el fiind format dintr-un număr redus de instrucţiuni. QNX Neutrino RTOS oferă licenţă gratuită pentru utilizare în scopuri necomerciale şi o reţea de documentare on-line pentru creatorii de aplicaţii. Detalii despre structura şi performanţele acestui produs pot fi obţinute de la http://www.qnx.com.

2.4 Limbajul de programare şi mediile de dezvoltare

Limbajul de programare, folosit pentru implementarea serverului de supervizare a fost, în principal, limbajul C++, acesta satisfăcând cerinţele de viteză, impuse de specificul problemei, şi fiind foarte bine suportat de tehnologia CORBA [1][6].

Pentru compilarea serverului s-a folosit *GNU C++ 2.95.2*, rulând sub QNX. Deoarece proiectul a fost realizat prin preluarea unor module software, elaborate într-o etapă de cercetare anterioară [9], a fost necesară recompilarea driver-ului de comunicaţie cu automatul programabil (scris iniţial pentru WATCOM C şi o versiune mai veche de QNX) folosind *GNU C 2.95.2* şi realizarea unor mici modificări pentru adaptarea acestuia la noua versiune a sistemului de operare, printre altele impunându-se folosirea unei biblioteci speciale de adaptare.

Compilatorul folosit este foarte bine suportat de implementarea ORBacus/E, aspect esenţial în raport cu alegerea produsului CORBA deoarece produsul ORBacus/E este disponibil sub formă de cod sursă, fiind necesară compilarea sa, alături de aplicaţia ţintă [4].

3. Structura sistemului de supervizare

Deoarece principalul scop al proiectului era studierea mecanismelor ce stau la baza integrării software a diverselor module în cadrul unei întreprinderi virtuale, am ales ca exemplu practic de implementare realizarea unui sistem software de supervizare a unei celule flexibile de fabricaţie de tip DEGEM 2000, care să permită integrarea CORBA cu alte sisteme software din cadrul întreprinderii. Pentru ca exemplul să fie cât mai elocvent, am folosit varianta de integrare prin "împachetare software" (*wrapping*) parţială [11] a unei aplicaţii de supervizare, deja existente (*legacy application*), elaborate în cadrul unui proiect de cercetare anterior.

Celula flexibilă de fabricaţie DEGEM 2000 este un produs al firmei DEGEM, fiind destinată, în principal, activităţii educaţionale (de exemplu, pentru instruirea studenţilor sau a personalului unei întreprinderi) [2]. Din punctul de vedere al echipamentelor cu care este echipată, celula cuprinde patru posturi de lucru cu rol de: alimentare a sistemului cu piese, magazie, bandă rulantă, prelucrare şi asamblare a pieselor.

Modulul supervizor realizat în cadrul proiectului este destinat postului de alimentare, reprezentat schematic în figura 1. Deoarece tipurile de piese ce pot fi utilizate în cadrul celulei DEGEM 2000 sunt cilindri de tip 1 (cilindri neprelucrati), cilindri de tip 2 (cilindri semiprelucrati), paleți și paralelipipede dreptunghice, simbolurile folosite în reprezentarea postului de alimentare au următoarea semnificație:

B1 - buffer de cilindri de tip 1 (cilindri neprelucrați);

B2 - buffer de cilindri de tip 2 (cilindri semiprelucrați);

BO - buffer obiecte paralelipipedice;

BP - buffer paleți;

MA - masa de asamblare (locul în care paleții sunt încărcați fie cu piese cilindrice, fie cu piese paralelipipedice);

R1 - manipulator ce poate transporta paleți de la masa de asamblare la conveior sau de la conveior la masa de asamblare;

R2 - manipulator ce poate transporta cilindri de ambele tipuri spre masa de asamblare.

Cele 22 de comenzi pe care automatul programabil asociat celulei flexibile de fabricație este capabil să le execute pot fi inițiate prin setarea la valoarea 1 a unei stări specifice a automatului. Acțiunile realizate ca urmare a acestor comenzi și stările corespunzătoare din automatul programabil sunt:

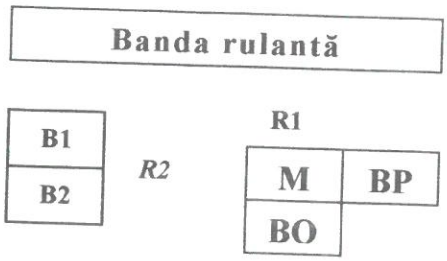

Figura 1. Postul de alimentare

1. ridicare a R1 (starea 401)
2. coborăre a R1 (starea 402)
3. deplasare a lui R1 către palet (starea 403)
4. deplasare a lui R1 către conveior (starea 404)
5. extindere a brațului lui R1 (starea 405)
6. revenire a brațului lui R1 (starea 406)
7. deschidere a gripper-ului lui R1 (starea 407)
8. închidere a gripper-ului lui R1 (starea 408)
9. coborâre a R2 (starea 409)
10. ridicare a R2 (starea 410)
11. deplasare a lui R2 către palet (starea 411)
12. deplasare a lui R2 către conveior (starea 412)
13. extindere a brațului lui R2 (starea 413)
14. revenire a brațului lui R2 (starea 414)
15. deschidere a gripper-ului lui R2 (starea 415)
16. închidere a gripper-ului lui R2 (starea 416)
17. eliberare a unui cilindru de tip 1 (starea 417)
18. eliberare a unui cilindru de tip 2 (starea 418)
19. eliberare a unui paralelipiped (starea 419)

20. eliberare a unui palet (starea 420)
21. sesizare a unei stări de avarie (starea 498)
22. ieşirea din starea de avarie (starea 499)

Structura vechiului sistem software de supervizare cuprindea trei module prin care se realiza conducerea posturilor de lucru [9], ca în figura 2.

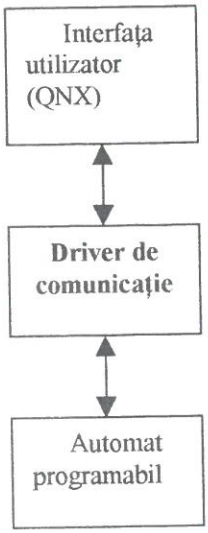

Figura 2. Structura vechiului sistem de conducere

Primul modul, diagrama ladder din automatul programabil, reprezintă comenzile ce pot fi îndeplinite de către postul de lucru în conformitate cu cerinţele utilizatorilor. Fiecare mişcare elementară este iniţiată de o stare care poate fi setată în automatul programabil prin informaţii transmise de către calculator.

Al doilea modul, drive-ul de comunicaţie cu automatul programabil, are rolul de a transmite către automatul programabil mesaje despre acţiunile pe care utilizatorul doreşte să le realizeze asupra instalaţiei, specificate într-un mod pe care automatul să-l poată înţelege. Comunicaţia se realizează folosind portul serial al calculatorului. În cazul în care se schimbă tipul de automat programabil, singura modificare necesară este scrierea unui nou driver.

Al treilea modul, cel de interfaţă grafică cu utilizatorul, permite transmiterea comenzilor de la utilizator către automatul programabil, prin intermediul driver-ului de comunicaţie. Modulul se execută local, sub sistem de operare QNX, pe calculatorul conectat direct cu automatul programabil al celulei de fabricaţie.

Structura noului sistem de supervizare şi integrare, elaborat în cadrul proiectului nostru de cercetare [9], este prezentată în figura 3.

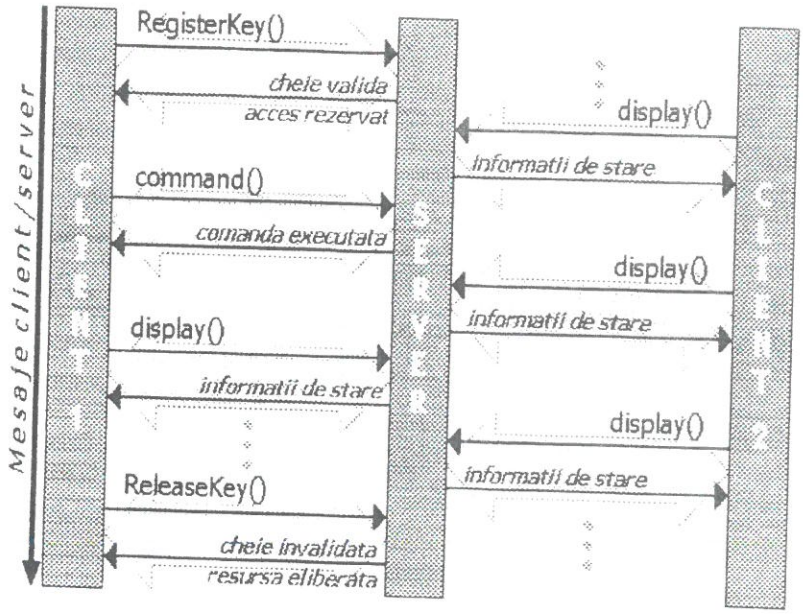

Figura 3. Noua structură a sistemului de conducere

Se observă că modulul de interfață grafică ce se execută sub sistem de operare QNX a fost înlocuit cu un *modul server CORBA*, care execută operații de intermediere a comunicației dintre automat (folosind pentru aceasta modulul driver) și modulul client, numit *Remote Cell Control* [9]. Clientul și serverul sunt despărțite printr-o rețea bazată pe protocol TCP-IP. Rolul tehnologiei CORBA este acela de liant care face posibilă comunicarea transparentă a celor două tipuri de module. Practic, serverul va exporta (va pune la dispoziția clienților interesați) un obiect care îl reprezintă și care descrie cât mai bine serviciile pe care le oferă. Indiferent unde este situat în rețea, clientul care folosește CORBA pentru a accesa obiectul exportat de către server, va folosi acest obiect ca și când ar fi un obiect local.

Metodele obiectului servant (funcțiile ce exprimă serviciile oferite) trebuie să reprezinte cât mai bine și mai complet capabilitățile de comandă și de supervizare ale serverului destinat conducerii celulei flexibile de fabricație, alese ca exemplu practic de implementare. Acestea sunt descrise de interfața OMG IDL, din fișierul **Cellcom.idl** (prezentat în secțiunea 4.1).

Un exemplu de implementare pentru un modul client, cu rol de panou de comandă, destinat să lucreze sub sistem de operare Windows și implementat în cadrul aceluiași proiect de cercetare, este prezentat în [9]. Se pot crea, însă, pe baza aceleiași interfețe OMG IDL module client cu alta funcționalitate (câteva sugestii sunt prezentate în secțiunea destinată concluziilor), implementate în diverse limbaje de programare și gândite pentru a lucra sub diverse sisteme de operare, toate conforme standardului de integrare CORBA.

4. Particularități de proiectare și implementare a serverului QNX/CORBA

4.1. Detalii legate de proiectare

Principala sarcină a sistemului de supervizare este de a vizualiza și de a influența, prin intermediul unor comenzi, evoluția celulei flexibile de fabricație. Din acest punct de vedere, este necesar ca în cadrul interfeței ce descrie funcționalitatea serverului să fie incluse metode pentru citirea stărilor serverului și pentru transmiterea codului comenzilor. Deoarece serverul se execută sub sistem de operare QNX și aplicația se desfășoară în timp real, apelurile de metode trebuie să fie foarte rapide. Informația transmisă trebuie să fie succintă pentru a permite o funcționare performantă, chiar și în cazul dispunerii geografice distincte a modulelor client și server.

Aplicațiile cu rol de client pot avea funcționalitate diversă: panou de comandă al celulei flexibile [9], analiza funcționării pe o anumită perioadă de timp etc. Aceste aplicații pot necesita informații legate de timp, dar și anumite măsuri de precauție, care să gestioneze în mod corespunzător accesul clientului la serviciile oferite de către server. Se are în vedere, în mod special, preîntâmpinarea situațiilor de conflict în folosirea resurselor, de genul celor în care mai mulți clienți încearcă să comande simultan celula de fabricație.

Motivele enumerate anterior ne-au determinat să includem în interfaţa proiectată pentru serverul de supervizare şi integrare mai multe variante de metode de apel, grupate în următoarele categorii: interogare, comandă, solicitare şi cedare a accesului exclusiv la celula flexibilă de fabricaţie. Se poate observa, în continuare, structura fişierului **Cellcom.idl**, cu metodele comentate în mod corespunzător:

```
// timpul întors de server
struct inftime {
        short msec;
        short sec;
        short min;
        short hour;
        short day;
        short month;
        short year;
};
interface cellcom {
        // întoarce o cheie de acces pentru server, dacă serverul este liber; în caz de eşec, întoarce 0
        long RegisterKey();
        //eliberează cheia de acces, iar serverul devine liber pentru alţi clienţi
        void ReleaseKey(în long key);
        // preia informaţiile de stare; întoarce 0 în caz de succes
        short display(în long key, out long states);
        // preia informaţiile de stare şi de timp; întoarce 0 în caz de succes
        short display_t(în long key, out long states, out inftime time);
        // trimite o comandă celulei; întoarce 0 în caz de succes şi >0 pentru alte erori
        short command(în long key, în short cmd);
        // trimite o comandă şi preia informaţiile de stare; întoarce 0 în caz de succes
        short display_cmd(în long key, în short cmd, out long states);
        // trimite o comandă şi preia informaţiile de stare şi timp; întoarce 0 în caz de succes
        short display_cmd_t(în long key, în short cmd, out long states, out inftime time);
};
```

Metoda **RegisterKey()** returnează o cheie exclusivă de acces la server, cheie care va putea fi folosită de către client pentru a apela celelalte metode din interfaţă. Serverul va returna o nouă cheie numai după ce verifică faptul că nu există deja o cheie validă, folosită de către un alt client. Se consideră că o cheie este în curs de folosire din momentul în care a fost rezervată de un client şi până ce acesta apelează metoda **ReleaseKey()** pentru a o ceda.

Dacă, din anumite motive, un client care a rezervat o cheie nu îşi mai poate continua comunicarea cu serverul, pentru a permite serverului să răspundă solicitărilor formulate de alţi eventuali clienţi, se foloseşte un mecanism de urmărire a duratei de valabilitate a cheilor ("de expirare a cheilor"). La fiecare apelare a metodei **RegisterKey()**, serverul verifică timpul scurs de la ultima folosire a cheii curente. Dacă valoarea depăşeşte 5 secunde, atunci cheia curentă este considerată invalidă ("expirată") şi se permite din nou realizarea apelurilor de rezervare a cheilor. Dacă însă cheia este validă, metoda va întoarce ca rezultat valoarea 0, testarea rezultatului fiind obligatorie pentru client. Pentru a evita expirarea cheii pe care tocmai a rezervat-o, un client va trebui să interogheze în mod continuu serverul cu o perioadă mai mica de 5 secunde. Acest mod de funcţionare respectă caracteristicile rezultate în urma analizei problemei şi nu afectează performanţele sistemului software realizat.

Informaţiile de stare specifice celulei flexibile de fabricaţie sunt obţinute de către client prin interogarea periodică a serverului, folosind metoda **display(în long key, out long states)**. După apel, stările pot fi identificate în cadrul variabilei **states**, de tip CORBA **out**, adică returnată de server către aplicaţia client. Metoda întoarce valoarea 0 în caz de succes şi un cod de eroare în caz contrar. Deoarece nu creează probleme speciale în legătură cu partajarea accesului la celulă (metoda doar citeşte stările, nu le schimbă) cheia care trebuie trimisă ca prim parametru poate avea orice valoare, permiţând clienţilor care doresc doar să supravegheze celula să funcţioneze în paralel cu clientul care a obţinut cheia pentru comandă. Singurul motiv pentru care cheia se trimite totuşi ca parametru este ca ea să nu expire, atunci când este vorba despre cheia unui client înregistrat.

Pentru a permite crearea unor aplicaţii client care să folosească atât informaţii de stare cât şi informaţii legate de timpul de funcţionare, în interfaţa serverului a fost inclusă o metodă similară celei descrise anterior, dar care

furnizează ca rezultat, prin variabila **time,** informaţii despre momentul exact de timp la care s-a citit starea: **display_t(în long key, out long states, out inftime time).** Aceste informaţii sunt utile, de exemplu, în cazul folosirii unei baze de date în care să se păstreze un istoric al evoluţiei modulului de fabrică virtuală supervizat.

Metoda **command(în long key, în short cmd)**, permite transmiterea unei comenzi destinate celulei flexibile de fabricaţie, parametrul **cmd** fiind chiar codul comenzii respective. Primul parametru are în acest caz un rol esenţial, el fiind cheia folosită de către clientul înregistrat pentru a avea drept de comandă asupra celulei.

Ultimele două metode incluse în interfaţă, **short display_cmd(în long key, în short cmd, out long states)** şi **short display_cmd_t(în long key, în short cmd, out long states, out inftime time)** permit aplicaţiilor client să dea o comandă şi să obţină, ca rezultat al unui singur apel, informaţiile de stare, respectiv informaţiile de stare şi de timp. Combinarea comenzii şi a citirii stării într-o singură metodă duce la o execuţie mai rapidă decât apelarea succesivă a două metode separate. Motivul care a condus la proiectarea acestor variante de metode este dorinţa de a oferi posibilitatea creării unor aplicaţii client cât mai diverse, cu grad de performanţă adaptat cerinţelor specifice aplicaţiei.

Figurile 4 şi 5 ilustrează două dintre scenariile de interacţiune server-clienţi. Figura 4 prezintă apelurile CORBA (sub formă de cerere-răspuns) făcute de către un client care solicită o cheie de acces şi o primeşte, după care trimite o comandă destinată celulei flexibile şi citeşte stările din server. În partea dreaptă a figurii, este abstractizat un alt potenţial client care, fără a înregistra o cheie, poate doar supraveghea evoluţia serverului (a celulei flexibile de fabricaţie supervizate de acesta), cvasi-simultan cu executarea operaţiilor implicate de apelurile primului client. În acest fel, cel de-al doilea client poate, de exemplu, obţine informaţii de stare pe care să le stocheze apoi într-o bază de date. Figura 5 ilustrează situaţia în care un al doilea client solicită o cheie în timp ce serverul se ocupă deja de satisfacerea cererilor formulate de către un alt prim client. Cererea celui de-al doilea client nu va putea fi satisfacută, cheia dorită nu i se va acorda. Figura ilustrează mecanismele de protecţie implementate în server şi erorile receptionate ca răspuns de către cel de-al doilea client. Folosind o cheie invalidă, acesta nu va reuşi să iniţieze o comandă destinată celulei flexibile, fiindu-i permisă doar citirea stărilor.

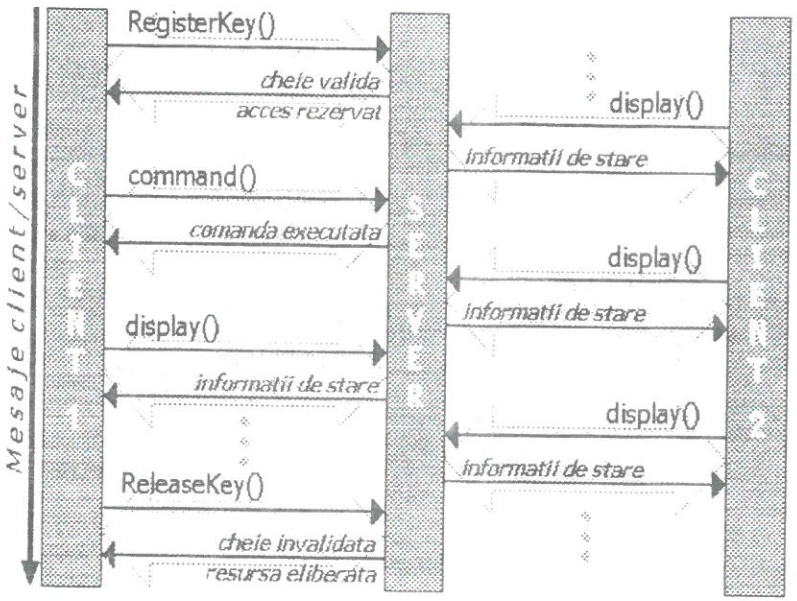

Figura 4. Un posibil scenariu de funcţionare

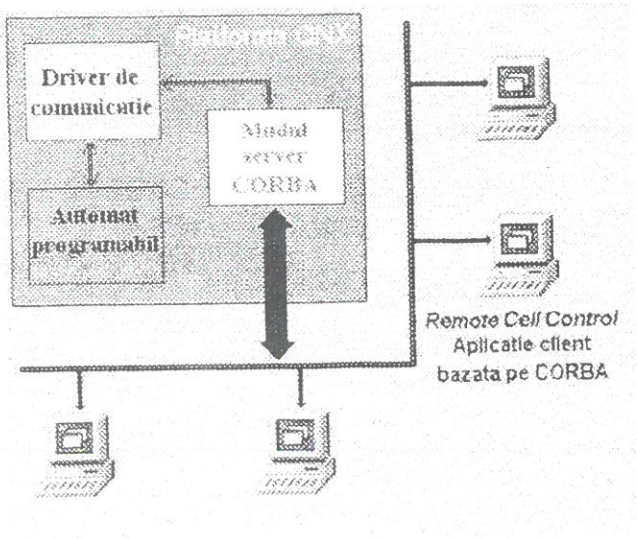

Figura 5. Un alt posibil scenariu de funcţionare

4.2. Detalii legate de implementare

Deoarece sistemul software proiectat se conformează standardului CORBA (*middleware* orientat obiect), etapele parcurse pentru implementare au fost cele prevazute în acest standard [1][3][8][11]. Astfel, codul sursă al aplicaţiei server a fost încapsulat sub forma unui obiect, fiecare metodă oferind câte un serviciu specific. Ca efect al compilării fişierului **cellcom.idl** (interfaţa OMG IDL descrisă în secţiunea 4.3), folosind utilitarul **eidl** [4], au fost generate mai multe fişiere sursă (.cpp) şi fişiere de tip header (.h), limbajul de implementare ales fiind C++. Incluzând opţiunea **–impl-all** în linia de comandă, s-a folosit facilitatea oferită de către compilatorul **eidl** de a genera fişiere ajutătoare pentru etapa de scriere cod C++. Unul dintre acestea (fişier sursă, numit **cellcom_impl.cpp**) conţinea, iniţial, numai definiţiile incomplete ale metodelor specificate în interfaţa IDL (câteva definiţii de parametri, de exemplu, cei returnaţi de către server), completarea cu cod fiind realizată ulterior de către programator.

Deoarece standardul CORBA permite adăugarea unor funcţii interne suplimentare faţă de cele precizate în interfaţă [1] atât pentru aplicaţia server, cât şi pentru cea client, am inclus în funcţionalitatea serverului şi alte câteva metode ce vor fi descrise pe scurt în cele ce urmează, impreună cu funcţiile specificate în interfaţă.

Constructorul [8] obiectului server este o metodă deosebit de importantă, conţinând codul program de iniţializare a acestui obiect. Folosind ca parametru valoarea curentă a timpului, se realizează iniţializarea generatorului de numere aleatoare, iar variabila globală **lastaccess** este folosită ulterior pentru verificarea timpului de expirare a cheii de acces. Tot în cadrul constructorului, se apelează funcţia de iniţializare pentru biblioteca **mig4nto.** Această bilioteca a fost legată (link-editată) la modulul server şi la driver pentru a menţine suportul oferit anumitor funcţii de sistem, existente în versiuni anterioare ale sistemului de operare QNX. Opţiunea este motivată de faptul că s-a dorit integrarea CORBA a unei aplicaţii prin "împachetare" a codului (*wrapping*), aşa cum s-a menţionat în secţiunea 3.

```
srand( (int) time(NULL));

lastaccess = time(NULL);

mig4nto_init();
```

Pentru ca funcţiile înglobate în aplicaţia server să poată colabora cu driver-ul automatului programabil, s-a inclus în server o funcţie numită **qnx_name_locate()**, care caută implementarea driver-ului şi returnează identificatorul de proces asociat. Valoarea –1 a identificatorului indică faptul că driver-ul nu există ca proces la momentul curent, deci execuţia serverului nu poate continua.

```
spid = qnx_name_locate (0,"qfc/mod_drv",0,NULL);
if (spid == -1) {
    printf ("Server not running\n");
    exit (-1);
}
```

Pentru verificarea validității cheii folosite de către un anumit client se folosește funcția **int check_key(unsigned long testkey)**. Rezultatul returnat are valoarea 1, dacă variabila *testkey* reprezintă o cheie validă și valoarea 0 în caz contrar. Atunci când cheia testată se dovedește a fi validă, funcția reactualizează timpul de acces asociat. Cheia poate expira, dacă nu este folosită un număr de secunde mai mare decât valoarea maximă, stabilită pentru timpul de acces, **MAXIDLE** (implicit, 5 secunde):

```
#define MAXIDLE 5

// Verifică dacă o cheie este validă. Dacă da, se schimbă timpul de acces al cheii și se
// întoarce 1. Dacă nu, se întoarce 0.
int check_key(unsigned long testkey){
  if(validkey){
    time_t timecrt = time(NULL);
    if(difftime(timecrt, lastaccess) > MAXIDLE){
      validkey = 0;
      return 0;
    }
    if(testkey != key)
      return= 0;
  }
  else return 0;
    lastaccess = time(NULL);
  return 1;
}
```

Gestionarea cheilor de acces se face folosind două metode accesibile din exteriorul serverului: **RegisterKey()** și **ReleaseKey()**. Rolul primei metode este de a aloca o cheie pentru clientul apelant, dacă acest lucru este posibil. Variabila **_r**, de tip **CORBA::Long**, este folosită pentru a memora noua cheie care va fi trimisă ca răspuns clientului. O cheie cu valoarea 0 semnifică imposibilitatea momentană de acordare a accesului și va fi returnată în cazul în care exista deja o cheia validă în momentul formulării cererii de către client. Cheia existentă va ramâne în continuare validă.

```
CORBA::Long _r = 0;
  static unsigned short counter = 0;
  if(difftime(time(NULL), lastaccess) > MAXIDLE)
    validkey = 0;
  if(validkey) return _r;
```

Cheile folosite sunt formate din câte 32 de biți, 16 dintre aceștia fiind generați în mod aleator. Variabila **counter,** de tip **unsigned short**, memorează ceilalți 16 biți, incrementându-și valoarea după fiecare operație de generare a unei chei. Serverul se asigură astfel că nu pot exista două chei consecutive identice. De fapt, se garantează că toate cele 65535 de chei consecutive, generate pe parcursul duratei de viață a unui proces server, vor fi distincte și create pseudoaleator. Această opțiune de implementare elimină situații nedorite de tipul celor în care un client ce posedă o cheie expirată ar putea accesa serverul, intrând în conflict cu un alt potențial client.

```
counter++;
  if(counter == 0) counter = 1;
  *(unsigned short *)&key = rand();
  *((unsigned short *)&key+1) = counter;
```

```
validkey = 1;
lastaccess = time(NULL);
_r = key;
```

Când un client doreşte să încheie sesiunea de lucru cu serverul supervizor, va apela metoda **ReleaseKey()**, folosind ca parametru cheia care i-a fost alocată anterior. Metoda va invalida cheia curentă. Clienţii pot invalida în acest mod numai cheile care le aparţin.

Citirea stărilor şi scrierea comenzilor în automatul programabil se execută folosind alte două funcţii interne, neaccesibile prin interfaţa obiectului, şi anume **ReadStates()** şi respectiv **SendCommand()**. Aceste funcţii sunt apelate din cadrul metodelor de interfaţă ale serverului.

Funcţia **ReadStates()** întoarce un parametru de tip **unsigned long**, care este de fapt o mască de biţi reprezentând cele 22 de stări descrise în cadrul secţiunii 3. Funcţia implementează comunicaţia cu driver-ul automatului programabil folosind câteva variabile buffer:

```
long int buff[32];
char buf[100];
struct _mxfer_entry s_mx[2], r_mx[2];
mod_qry snd;
mod_rpl rec;
```

Structura **snd** conţine un mesaj pentru automatul programabil. Câmpul **snd.nod** conţine codul de identificare al postului de lucru, care urmează să recepţioneze mesajul de la driver (de exemplu, 20 reprezintă postul de alimentare), iar câmpul **snd.len** specifică faptul că mesajul este compus din două părţi, prima fiind chiar structura **snd**, iar a doua fiind buffer-ul **long int buff[32]**. În acest buffer, se va primi răspunsul automatului. Iniţial el conţine doar un prim element, cu semnificaţia de cerere de citire a stării 10001 (codul intern prin care automatul îşi reprezintă prima stare). Al doilea element din buffer va reprezenta numărul de stări care se vor citi, începând cu starea specificată. Ultimul câmp, **snd.cmd**, informează automatul programabil că se va executa o operaţie de citire, reprezentată prin valoarea 1.

```
snd.nod = 20;
snd.len = 2;
snd.cmd = 1;

buff[0] = 10001; // prima stare care se citeşte
buff[1] = 32;    // numărul total de stări care se citesc.
```

Mesajele trimise şi receptionate au două părţi, pentru ele folosindu-se variabilele structură **snd** şi **rec** şi respectiv buffer-ele **buff** şi **buf**. Pentru cele 4 zone de memorie, se creează identificatori de bloc, folosind funcţia sistem **_setmx**, apoi se foloseşte o altă funcţie sistem, **Sendmx**, pentru trimiterea mesajului spre automat [6][7]. Ca parametri, se specifică identificatorul procesului driver, numărul de părţi din care sunt compuse mesajele, precum şi vectorii care conţin identificatorii creaţi pentru zonele de memorie. Funcţia aşteaptă rezultatul, iar când acesta este disponibil va fi returnat în buffer-ul specificat pentru ieşire.

```
_setmx( &s_mx[0], &snd, sizeof (snd));
_setmx( &r_mx[0], &rec, sizeof (rec));
_setmx( &s_mx[1], buff, sizeof(long int) * 2);
_setmx( &r_mx[1], buf, sizeof (char) * 100);
Sendmx (spid, 2, 2, s_mx, r_mx);
```

La revenirea din funcţia **Sendmx**, variabila **buf** va conţine, într-o formă codificată, cele 22 de stări ale automatului. Stările se regăsesc în octeţii cu indice multiplu de 4 din buffer-ul utilizat pentru memorarea răspunsului, codificate fiecare pe câte un bit. Deoarece ordinea lor nu este aceeaşi cu ordinea de pe panoul de comandă, se foloseşte un vector numit **translate** cu ajutorul căruia se realizează corespondenţa. Acesta conţine indexul biţilor din buffer, care corespund fiecărei stări, de la prima până la cea de-a 22-a. Rezultatul final este masca pe biţi a stărilor, unificată într-o singură variabilă de tip *unsigned long*.

```
int translate[22]={
    14, 14, 3, 2, 0, 1, 4, 5, 22, 23, 27, 21, 19, 19, 10, 11, 8, 9, 12, 13, 20, 24
};
```

```
int i;
unsigned long mask = 1;
unsigned long states = 0;
for(i = 0; i < 22; i++){
  if( buf[ (translate[i]/8) * 4 ]  &  ( ((unsigned char)1) << (translate[i]%8) ))
      states |= mask;
  mask <<= 1;
}
```

Poziţiile 0 şi 1, respectiv 12 şi 13 din vectorul translate corespund unor stări complementare. Automatul transmite, practic, un singur bit pentru fiecare pereche de stări, bit care va trebui complementat pentru fiecare dintre cele două stări pereche:

```
if(states & 1)
    states &= ( states - (unsigned long)1 << 1 );
else
    states |= (unsigned long)1 << 1;
if( states & (unsigned long)1 << 12 )
    states &= ( states - (unsigned long)1 << 13 );
else
    states |= (unsigned long)1 << 13;
```

Funcţia **SendCommand()** primeşte ca parametru un întreg, reprezentând o comandă de stare, pe care o transmite mai departe driver-ului. Se folosesc aceleaşi structuri de date şi buffere ca şi în cazul citirii stărilor automatului (funcţia **ReadStates()**). De data aceasta, câmpul **snd.cmd** va avea valoarea 2, reprezentând scrierea unei stări în automat. Asociate celor 22 de stări, există 22 de comenzi pe care automatul programabil le poate interpreta. Aceste comenzi sunt asociate stărilor folosind un vector, numit **translatecom**. Conform vectorului, pentru comanda stării 0, se va folosi codul de comandă 407, pentru comanda stării 1 va fi folosit codul 408, apoi 403, şi aşa mai departe, până la ultima stare. Comanda se va memora în variabila **buff[0]**, iar **buff[1]** va conţine valoarea 1, care este simbolul unei stări active. Automatul va recunoaşte codul şi va acţiona conform trecerii în starea respectivă.

```
int translatecom[22] = {
    407, 408, 403, 404, 401, 402, 405, 406, 417, 418, 420, 419, 415, 416,
    411, 412, 410, 409, 413, 414, 500, 499
  };

buff[0] = translatecom[cmd];  // codul translatat al stării
buff[1] = 1;                  // noua valoare pentru stare

// se trimite mesajul de comandă către driver
Sendmx (spid, 2, 2, s_mx, r_mx);
```

Metodele **display()** şi **display_t()** returnează masca de biţi ce conţine stările automatului. Pentru interogarea acestuia se foloseşte funcţia **ReadStates()**, descrisă anterior. În plus faţă de metoda **display()**, metoda **display_t()** întoarce o structură ce conţine timpul şi data citirii stărilor, valori măsurate de către server. Structura (generată conform specificaţiei din fişierul **cellcom.idl**) se numeşte **time,** este de tipul **inftime_out** şi conţine informaţii despre an, luna, zi din luna, ora, minut, secunda şi milisecunda momentului măsurării. Pentru măsurarea precisă şi translatarea timpului pe platforma QNX [6][7], se folosesc funcţiile **clock_gettime()**, respectiv **localtime()**:

```
struct timespec tims;
   struct tm *tlowres;
   int miliseconds;
   clock_gettime(CLOCK_REALTIME, &tims);
   tlowres = localtime(&tims.tv_sec);
   miliseconds = tims.tv_nsec / 1000000;
```

```
time.msec = miliseconds;
time.sec = tlowres->tm_sec;
time.min = tlowres->tm_min;
time.hour = tlowres->tm_hour;
time.mday = tlowres->tm_mday;
time.month = tlowres->tm_mon
time.year = tlowres->tm_year;
```

Metoda folosită pentru transmiterea de comenzi adresate celulei flexibile de fabricaţie este **command()**. După ce verifică validitatea cheii asociate clientului ce iniţiază comanda şi după încadrarea codului comenzii în intervalul valoric permis, metoda apelează funcţia **SendCommand()** pentru a trimite informaţia către automatul programabil.

Ultimele două metode specificate în interfaţa asociată serverului de supervizare, **display_cmd()** şi **display_cmd_t(),** reprezintă o variaţie a metodelor de citire a stărilor sau de comandă descrise anterior şi au fost deja prezentate ca funcţionalitate în cadrul secţiunii 4.2.

5. Concluzii

Sistemul software, descris în această lucrare, a fost gândit astfel încât să ofere o soluţie performantă, standardizată, de integrare a unui modul în cadrul unei întreprinderi virtuale. Modulul ales a fost celula flexibilă de fabricaţie de tip educaţional, DEGEM 2000, folosită ca un prim pas în vederea valorificării ulterioare a experienţei astfel dobândite în cadrul unui sistem industrial.

Aplicaţia include un server de timp real (QNX), ataşat platformei celulei flexibile (descris în cadrul acestei lucrări), şi un modul client (descris în [9]), cu rol de panou de comandă, ce poate fi instalat pe un sistem Windows. Datorită tehnologiei software utilizate (CORBA), serverul şi clientul pot interacţiona prin intermediul oricărei reţele de comunicaţie ce foloseşte protocol TCP/IP. Produsul CORBA, ales pentru implementare, este un produs special conceput pentru aplicaţii de timp real şi sisteme încapsulate (*embedded systems*) – ORBACUS/E.

Sistemul software proiectat poate fi extins prin crearea altor aplicaţii client, de acelaşi tip cu cel realizat de noi sau cu funcţionalitate diferită, cu condiţia de a cunoaşte interfaţa IDL OMG a serverului. Aplicaţiile cu rol de client pot avea funcţionalitate diversă: panou de comandă al celulei flexibile [9], citirea şi stocarea informaţiilor de stare într-o bază de date, analiza funcţionării pe o anumită perioadă de timp etc. Aceste aplicaţii necesită informaţii legate de timp, dar şi anumite măsuri de precauţie, care să gestioneze în mod corespunzător accesul clientului la serviciile oferite de către server. Din acest motiv, în proiectarea interfeţei serverului de supervizare s-a urmărit în mod special preîntâmpinarea situaţiilor de conflict în folosirea resurselor, de genul celor în care mai multi clienţi încearcă să comande simultan celula flexibilă de fabricaţie.

Aplicaţiile client pot fi realizate în limbaje diferite şi se pot executa sub sisteme de operare diferite, acesta fiind unul dintre avantajele integrării oferite de către utilizarea unei tehnologii de tip *middleware* [8][11]. Deoarece serverul a fost conceput special pentru a lucra sub sistem de operare QNX, experienţa dobândită în cadrul proiectului poate fi folosită şi pentru integrarea altor module de fabrică virtuală, monitorizate prin sisteme software "încapsulate" (*embedded systems*) sau nou create, QNX oferind un suport deosebit de performant în acest sens [7]. Modulele respective pot fi, de exemplu, destinate realizării unor sarcini în mediu de lucru periculos sau nociv şi monitorizate de la distanţă prin intermediul unui sistem software similar cu cel realizat de către noi.

Cu toate că principalul scop al realizării aplicaţiei a fost studierea mecanismelor de integrare a componentelor în cadrul unei întreprinderi virtuale şi oferirea unei soluţii viabile pentru activitatea industrială, rezultatul obţinut poate fi utilizat cu succes şi în cadrul activităţii educaţionale de tip clasic sau la distanţă, inclusiv în cadrul unor cursuri de perfecţionare a personalului întreprinderilor. Prin supravegherea celulei flexibile de fabricaţie cu ajutorul unei camere video ce poate fi conectată la Internet (Web cam) şi includerea unui modul software dedicat, aplicaţia permite cursanţilor să vizualizeze efectul comenzilor pe care le furnizează de la distanţă, serverul fiind astfel proiectat încât să nu accepte transmiterea unor comenzi conflictuale.

Bibliografie

1. ***: CORBA documentation, http://www.corba.org , documentaţie Internet.
2. ***: Documentaţia pentru celula flexibilă de fabricaţie DEGEM 2000.

3. ***: Object Management Group, http://www.omg.org , documentaţie Internet.

4. ***: Object Oriented Concepts Inc., http://www.ooc.com , documentaţie Internet .

5. **ORFALI, R., D. HARKEY, J. EDWARDS**: Client/Server Survival Guide, Wiley Computer Publishing Group, New York, USA, 1999.

6. ***: QNX – Watcom C – Library Reference Manual, documentaţie de firma.

7. ***: QNX – http://www.qnx.com , documentaţie Internet.

8. **SARU, D., A.D. IONIŢĂ**: Sisteme de programe orientate pe obiecte, Editura ALL Educaţional, Bucureşti, Romania, 2000.

9. **SARU, D., A. PETCU**: Sistem software de monitorizare în timp real, realizat prin integrare QNX, CORBA, Windows. În: Revista Romana de Informatica şi Automatica, vol. 12, nr.2, 2002, pp. /////.

10. **SERAIN, D.**: Enterprise Application Integration. L'architecture des soluţions e-business, Dunod, Paris, 2001.

11. **SIEGEL, J.**: CORBA 3 Fundamentals and Programming. Second Edition, Wiley Computer Publishing Group, New York, USA, 2000.