UNIVERSITATEA TEHNICĂ CLUJ-NAPOCA

DEPARTAMENTUL ELECTROTEHNICĂ ŞI MĂSURĂRI

Cursul 14 INSTRUMENTE VIRTUALE DE COMUNICAŢIE BAZATE PE MEDIUL DE PROGRAMARE LABVIEW

Introducere curs	În acest curs sunt prezentate instrumente virtuale de comunicaţie bazate pe mediul de programare LABVIEW.

Obiective curs	• Noţiuni introductive despre aplicaţiile distribuite.
	  • Elemente de bază ale modelului „client-server”.
	  • Introducere în instrumentaţia virtuală bazată pe LabVIEW.
	• Instrumentul virtual bazat pe mediul de programare LabVIEW.
	  • Componente de comunicaţie la nivelul de transport (OSI 4), bazate pe protocolul TCP.
	  • Componente de comunicaţie la nivelul de sesiune, prezentare şi aplicaţie (OSI 5, 6, 7), bazate pe server-ul de comunicaţie DataSocket.
	• Pagină de web interactivă folosind DataSocket.

Durată medie de studiu individual	Durata medie de studiu individual: 100 minute. Acest interval de timp presupune parcurgerea continutului cursului şi rezolvarea testelor de autoevaluare

Cuprins:

14.1. Noţiuni generale despre aplicaţiile distribuite.....2
14.1.1. Elemente introductive. 2
14.1.2. Elemente de bază ale modelului „client-server”. 2
14.1.3. Introducere în instrumentaţia virtuală bazată pe LabVIEW. 4
14.2. Instrumentul virtual bazat pe mediul de programare LabVIEW. 5
14.2.1. Funcţii de comunicaţie la nivelul de transport (OSI 4), bazate pe protocolul TCP. 7
14.2.2. Componente de comunicaţie la nivelul de sesiune, prezentare şi aplicaţie (OSI 5,6,7), bazate pe server-ul de comunicaţie DataSocket. 10
14.3. Pagină de web interactivă folosind DataSocket. 16
TESTE DE AUTOEVALUARE 24
RĂSPUNSURI 25
Bibliografie.....25

14.1. Noţiuni generale despre aplicaţiile distribuite.

14.1.1. Elemente introductive.

Scopul acestui capitol îl constituie prezentarea unor soluţii de monitorizare, sau de control distribuit în reţea, utilizând instrumente virtuale construite în mediul de programare **LabVIEW**. Aceste instrumente se încadrează în categoria aplicaţiilor client/server.

Sistemele de operare de reţea permit partajarea datele şi a aplicaţiilor stocate pe terminalele din reţea. Un mediu de calcul distribuit este caracterizat de trei funcţii de bază:

- administrarea datelor,
- procesarea,
- prezentarea către utilizator.

Cele mai des folosite metode de realizare a aplicaţiilor distribuite sunt:

1. Metoda interfeţelor de programare a aplicaţiilor (**API** – *Application Programming Interface*). Interfaţa de programare a aplicaţiilor (**API**) este o metodă folosită în mod curent pentru partajarea informaţiei, într-o reţea client/server. Acestea sunt funcţii, oferite de către producătorii de software, care permit programatorilor de aplicaţii să acceseze resursele reţelei, într-o manieră standard. **API**-urile de reţea permit, de asemenea, interconectarea diferitelor aplicaţii care rulează sub acelaşi sistem de operare.

2. Metoda de schimb a informaţiilor prin utilizarea aplicaţiilor server de baze de date. În această situaţie, aplicaţiile de tip client trebuie să furnizeze aplicaţiilor server de baze de date informaţiile obţinute din măsurare sau supraveghere a procesului controlat. Aplicaţiile de monitorizare, de tip client, preiau informaţiile stocate şi oferite (publicate) de către aplicaţiile de tip server de baze de date, pe care le folosesc apoi în deciziile de control. Astfel, informaţia este transferată între clienţi, doar prin intermediul aplicaţiilor server de baze de date.

3. Metoda manipulării de la distanţă a terminalelor text *(Remote Login),* sau a „ferestrelor” *(remote windowing).* Metoda foloseşte componente care permit transferul ecranului de control al unui sistem de operare şi se utilizează atunci când aplicaţia de măsurare, sau control, operează doar local şi nu permite transferul informaţiilor în reţea. Pentru transferul ecranelor text (console text) se folosesc aplicaţii de tipul **Telnet** sau **SSH** *(Secure Shell)*, în cazul transferului securizat al datelor. Datele de intrare şi de ieşire sunt transferate sub formă de expresii text, comenzi-linie sau şiruri de caractere.

În cazul sistemelor de operare cu interfeţe grafice (**GUI** - *Graphic User Interface*), se transferă la distanţă gestiunea ferestrelor şi a aplicaţiilor. Un astfel de exemplu îl reprezintă, în cazul sistemelor de operare din familia Microsoft Windows, **Remote Desktop**, iar în cazul sistemelor Unix, Linux **XWindows**. În toate aceste situaţii, aplicaţia este lansată pe terminalul pe care se face achiziţia, sau controlul, spre terminalul de la distanţă transferându-se doar o imagine a datelor obţinute în urma procesului de achiziţie.

4. Metoda utilizării de către aplicaţii a apelurilor de tip **RPC** *(Remote Procedure Call)*. Acestea sunt programe complementare cu capabilităţi de interconectare între aplicaţii distribuite, între diferite platforme *hardware* sau *software*. Aplicaţiile care sunt implementate în reţea pot folosi aceste proceduri pentru ordonarea mesajelor, traducerea diferitelor coduri şi menţinerea integrităţii protocolului de comunicaţie, micşorând în acest fel complexitatea aplicaţiilor.

14.1.2. Elemente de bază ale modelului „client-server”.

O arhitectură client/server este un model de comunicaţie în care aplicaţiile software sunt distribuite, între entităţi diferite, dintr-o reţea locală, reţea publică, sau combinaţii ale acestora. Aplicaţiile client, iniţiază conexiunea şi cer, sau transmit informaţii de la, sau spre, una sau mai multe aplicaţii de tip server.

Modelul „client/server” permite distribuţia acestor funcţii între mai multe echipamente din reţea.

Orice sistem de calcul din reţea poate avea funcţiuni specifice unei aplicaţii de tip client, sau de tip server. Aplicaţia client este entitatea care cere deschiderea sesiunii de lucru în reţea şi execuţia sarcinii. Aplicaţia de tip server este entitatea care aşteaptă o cerere, venită din partea unei entităţi client şi execută un set de sarcini.

Realizarea conexiunilor, necesare comunicaţiei între aceste entităţi, se poate face direct la nivelul **OSI 4** (transport), sau prin utilizarea unor componente ale nivelului **OSI 5** (sesiune) de tipul proceduri de apel de la distanţă (**RPC** – *Remote Procedure Calls*).

Nu toate informaţiile dintr-o arhitectură client/server sunt stocate şi controlate de aplicaţia server, aplicaţiile client pot fi capabile de a stoca şi procesa date, local.

Într-o arhitectură „client/server" identificarea părţilor (client sau server) nu depinde de rolul funcţional al aplicaţiilor, ci de modul în care aceste aplicaţii interacţionează între ele şi cu nivelul **OSI 4** (transport). Se consideră un sistem de două terminale pe care operează câte o aplicaţie client şi una server, aşa cum se poate vedea în **figura 14.1.**

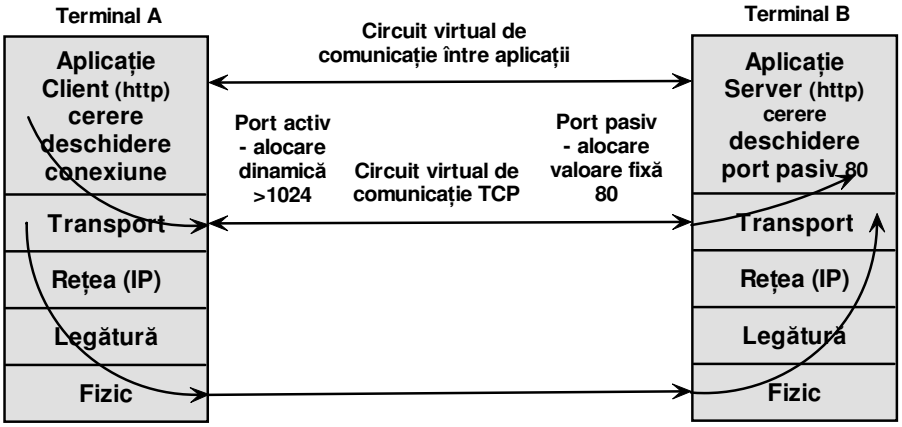

Fig. 14.1. Principiul comunicaţiei între perechea de aplicaţii "client/server".

În timpul lansării în execuţie a unei aplicaţii server, aceasta face o cerere, nivelului de transport (în cazul de faţă componentei **TCP** de pe terminalul B), de a „deschide" un port pasiv cu valoare fixă şi de a-i transfera eventualele cereri primite pe acesta, de la o aplicaţie client. Valoarea fixă a portului (număr de identificare a aplicaţiei deservită de componenta de transport) este necesară pentru ca aplicaţia pereche client să ştie unde să trimită eventualele cereri.

Pentru o aplicaţie server de web, portul de identificare standard este 80 (detalii suplimentare se pot găsi în capitolul 12). În cazul în care aplicaţia client (care operează pe terminalul A) are nevoie să transfere date spre aplicaţia server (de pe terminalul B) va cere componentei de transport deschiderea unui port activ (alocat dinamic) prin care să stabilească o conexiune cu componenta de transport de pe terminalul destinaţie. Altfel spus, singura care poate iniţia o cerere de conexiune, la nivel de transport, este aplicaţia client. Aplicaţia server poate doar să aştepte, transferul unor date, realizat printr-o conexiune stabilită între componentele de transport, care operează pe cele două terminale.

Iniţierea, menţinerea şi închiderea conexiunii, între terminale, este sarcina exclusivă a nivelului de transport. După stabilirea acestei conexiuni, transferul datelor, între aplicaţii se poate face în ambele sensuri fără să mai conteze cine a iniţiat cererea de conexiune. O conexiune, stabilită între componentele omologe ale nivelului **OSI 4** (transport), este univoc determinată prin câte două elemente, specifice fiecărui terminal, adresa de reţea şi portul de identificare a aplicaţiei. Această pereche se numeşte *socket*.

O aplicaţie care operează pe un terminal din reţea, este determinată global de un *socket* unic, indiferent dacă este client sau server. Acest lucru este valabil deoarece adresa de reţea a unui terminal este unică (în afara cazului în care este vorba despre terminale aparţinând unor reţele private izolate), iar numărul portului, de identificare a aplicaţiei de către componenta de transport, este de asemene unic.

Aşa cum s-a putut vedea în capitolul 4, pot exista conexiuni stabilite între diferite aplicații client şi aceeaşi aplicație server, identificată de un singur socket. Pentru deservirea simultană a mai multor aplicații client, aplicația server utilizează componente ale nivelului OSI 5, numite variabile de sesiune.

14.1.3. Introducere în instrumentația virtuală bazată pe LabVIEW.

Termenul de „instrumentație virtuală" a apărut odată cu necesitatea de a combina un instrument, de măsurare programabil, cu un computer personal, în vederea obținerii unui nou echipament cu flexibilitate şi performanțe mari. Arhitectura închisă a instrumentelor tradiționale necesită, pentru o aplicație de anvergură, un număr mare de aparate specializate. În viziunea actuală, funcționalitatea unui instrument trebuie definită de către utilizator şi nu de către producător. Reconfigurarea sa ulterioară, pentru alte aplicații, devine o problemă relativ uşoară, operația rezumându-se la elaborarea unei noi componente software, la nivel de aplicație, suportul hardware rămânând, în general, acelaşi.

Instrumentul virtual reprezintă o asociere între echipamente hardware flexibile (sisteme de achiziție de date, sau aparate de măsură programabile) ataşate unui sistem de calcul, împreună cu aplicația software care implementează funcțiile aparatului.

În contextul rețelelor digitale de comunicație, aceste instrumente virtuale pot opera, în mod distribuit, pe mai multe terminale, aşa cum se poate vedea în **figura 14.2**.

Instrumentele virtuale distribuite pot opera ca aplicații de sine stătătoare, care interacționează direct sub forma perechilor de aplicații „client/server", sau prin intermediul unor aplicații server de comunicație, sau baze de date [2, 4].

O soluție interesantă şi cu avantaje multiple, este includerea instrumentelor virtuale, sub forma unor secvențe de cod, în pagini web, care pot fi lansate în execuție cu ajutorul aplicațiilor de navigare pe web *(browser web)*.

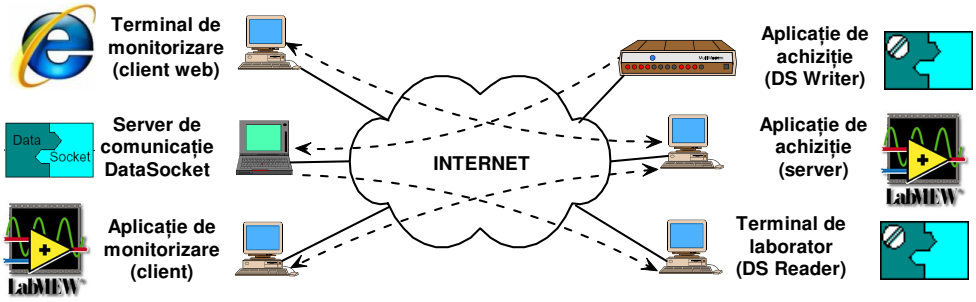

Fig. 14.2. Exemplu de aplicații „client/server" şi DataSocket folosite de mediul LabVIEW.

Instrumentul virtual combină, într-un mod transparent pentru utilizator:

- resursele sistemului de calcul,
- caracteristicile de măsurare şi control ale echipamentului hardware şi a componentei software utilizate pentru analiza datelor,
- comunicația între procese,
- prezentarea rezultatelor.

În aplicații de control a proceselor, după colectarea datelor de intrare (caracteristica stării unui sistem) se generează, după un anumit algoritm dat, semnale electrice transmise la ieşirea instrumentului virtual. Acestea pot fi folosite, într-un sistem de automatizare, pentru comanda şi controlul elementelor de execuție.

Rolul componentei software, într-un sistem de instrumentație virtuală, este multiplu:

- asigură o interfață, între operator şi echipamente, uşor de folosit;
- controlează echipamentele şi diferitele componente hardware,
- realizează prelucrarea matematică a datelor,
- stochează şi administrează datele;
- prezintă rezultatele într-o formă sugestivă pentru operator,

- coordonează resursele disponibile pentru implementarea funcţiilor impuse instrumentului virtual.
- asigură suportul necesar comunicaţiei între diferitele echipamente distribuite.

Soluţiile software pleacă de la implementări tradiţionale, în care informaţiile iniţiale sunt de tip text, până la abordări de dezvoltare grafică. Mediul de programare, de achiziţie şi analiză, **LabVIEW** face parte din ultima categorie.

LabVIEW *(Laboratory Virtual Instrumentation Engineering Workbench)* a fost creat începând cu anul 1986, în laboratoarele universităţii din Austin *(University of Texas)* S.U.A, fiind dezvoltat de către compania National Instruments.

LabVIEW este o platformă de instrumentaţie virtuală foarte complexă, disponibilă pentru majoritatea sistemelor de operare, implementarea aplicaţiilor realizându-se prin interfaţă grafică.

Acesta oferă, de asemenea, un suport important pentru comunicaţie şi interconectare a instrumentelor de măsurare distribuite în reţele digitale, fundamentat pe modelul de referinţă **OSI**.

În conformitate cu acest model, funcţiile de comunicaţie sunt împărţite şi aşezate într-o structură verticală pe nivele. Fiecare nivel realizează un set de funcţii primitive, numite servicii de nivel *N,* necesare comunicaţiei cu un alt sistem şi este deservit de către nivelul imediat inferior.

În acelaşi timp, fiecare nivel este furnizor de servicii pentru nivelul imediat superior.

În mod ideal, funcţiile ar trebui să fie astfel definite (şi împărţite) încât eventualele modificări din cadrul unui nivel să nu atragă după sine modificări în funcţiile celorlalte nivele. În acest fel, o problemă complexă se descompune într-un număr de subprobleme mai uşor de rezolvat [1].

Modelul rezultat are şapte nivele, aşa cum se vede în **figura 14.1.** Fiecare partener de comunicaţie trebuie să dispună de o aceeaşi arhitectură de protocoale (cunoscută şi sub numele de stivă de protocoale) care realizează aceleaşi funcţii.

Particularizând aceste principii pentru mediul de programare pentru achiziţia şi analiza numerică a datelor **LabVIEW**, se obţine modelul reprezentat în **figura 14.3** [2, 4].

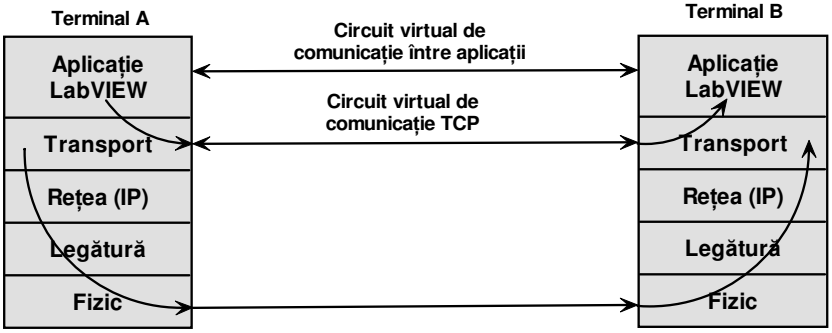

Fig. 14.3. Comunicaţia între două aplicaţii în reţea pe baza modelului de referinţă ISO - OSI.

14.2. Instrumentul virtual bazat pe mediul de programare LabVIEW.

Mediul de programare **LabVIEW** utilizează funcţii speciale care implementează componente ale nivelului **OSI 4** (transport) reprezentate prin protocolul **TCP**, sau a unor componente de nivel **OSI 5,6,7** (sesiune, prezentare şi aplicaţie), reprezentate prin server-ul de comunicaţie **DataSocket**. Acestea se găsesc sub forma unor instrumente virtuale (**VI**) care sunt prezentate în cele ce urmează. Alegerea soluţiilor de interconectare depinde de arhitectura reţelei de interconectare şi de complexitatea aplicaţiei. În **figura 14.4** se prezintă o comunitate de terminale formată din două reţele private *(intranet)* şi o reţea publică *(Internet)*.

Conectarea reţelelor private, la reţeaua publică, se face prin intermediul ruterelor **R1**, respectiv **R2**, nu prin rutare directă ci printr-un procedeu care se numeşte **NAT** *(Network Address Translation)*. Acest procedeu permite transferul pachetelor de date, dinspre un terminal aflat în reţeaua privată spre reţeaua publică, prin înlocuirea adresei sursă (de nivel OSI 3 - reţea) cu adresa publică a ruterului (translatare) care asigură conectarea. Acest lucru nu permite ca o aplicaţie care

operează pe un terminal din rețeaua publică, sau dintr-o altă rețea privată, să acceseze direct o aplicație care operează pe un terminal aflat într-o rețea privată [3].

Pentru a asigura totuşi comunicația între un terminal din rețeaua privată şi unul din rețeaua publică, ruterul ține o evidență (tabelă de NAT) a tuturor cererilor venite din rețeaua privată, spre terminale din rețeaua publică.

Când ruterul primeşte pachete de răspuns, sosite de la acestea, le va întoarce spre terminalele private care au inițiat acele cereri, ținând seama de informațiile înregistrate în tabela de **NAT**.

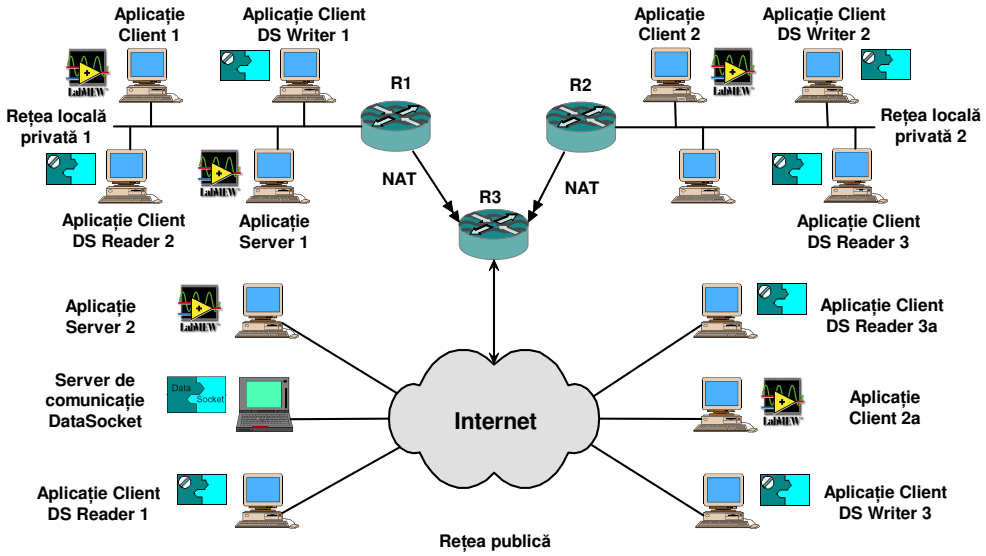

Fig. 14.4. Instrumente virtuale distribuite, bazate pe aplicații "client/server", folosind LabVIEW.

Prin acest procedeu, deşi se permite accesul, spre rețeaua publică, a terminalelor din rețeaua privată, acestea sunt izolate adresele lor fiind ascunse terminalelor din rețeaua publică. Astfel, pot exista mai multe rețele private care să folosească aceleaşi adrese pentru terminale, situație care nu se întâlneşte în cazul rețelei publice.

Inițierea unei cereri de conectare o poate face doar o aplicație client, aplicația server fiind aptă doar să răspundă la aceasta, consecința a modului de stabilire a conexiunilor de către componentele nivelului **OSI 4** (transport). De aceea, aplicațiile server trebuie să opereze pe terminale aflate în rețeaua publică, eventuale aplicații server aflate în rețeaua privată nu pot fi accesate decât de aplicații client care operează în aceeaşi rețea.

Analizând cazul interconectării a două instrumente virtuale de achiziție şi monitorizare, în contextul celor prezentate mai sus, se pot identifica 3 cazuri:

1. Ambele instrumente operează pe terminale din aceeaşi rețea, privată sau publică. În această situație, aplicațiile se pot interconecta direct, pachetele de rețea fiind rutate în ambele sensuri. Se pot utiliza atât perechi „client/server” (Client 1 –Server 1 sau Client 2a – Server 2) cât şi perechi de aplicații client, *writer* şi *reader* (DS Writer 3 - DS Reader 3a), interconectate prin server de comunicație **DataSocket**, care operează pe un terminal din acea rețea privată, sau pe unul din rețeaua publică.
2. Un instrument operează în rețeaua privată, iar celălalt în rețeaua publică. În această situație, se pot folosi perechi „client/server” doar în cazul în care aplicația server operează pe un terminal din rețeaua publică, iar cea client pe un terminal din rețeaua privată (Client 2 – Server 2).

   Se pot folosi şi perechi de aplicații client, *writer* şi *reader* (DS Writer 1 – DR Reader 1), interconectate prin server de comunicație **DataSocket**, situație în care acest server trebuie să opereze pe un terminal din rețeaua publică.
3. Instrumentele virtuale operează în două rețele private. În această situație este obligatorie utilizarea de perechi de aplicații client, *writer* şi *reader* (DS Writer 2 – DS Reader 2), interconectate printr-un server de comunicație **DataSocket** care operează pe un terminal din

reţeaua publică.În toate situaţiile prezentate mai sus, aplicaţia server de comunicaţie trebuie să fie accesibilă de fiecare dintre aplicaţiile client.

14.2.1. Funcţii de comunicaţie la nivelul de transport (OSI 4), bazate pe protocolul TCP.

Componentele de comunicaţie ale protocolului de transport **TCP** realizează circuite virtuale între nivelele **OSI 4** aflate pe cele două terminale pe care operează aplicaţiile distribuite, aşa cum se poate vedea în **figura 14.5.**

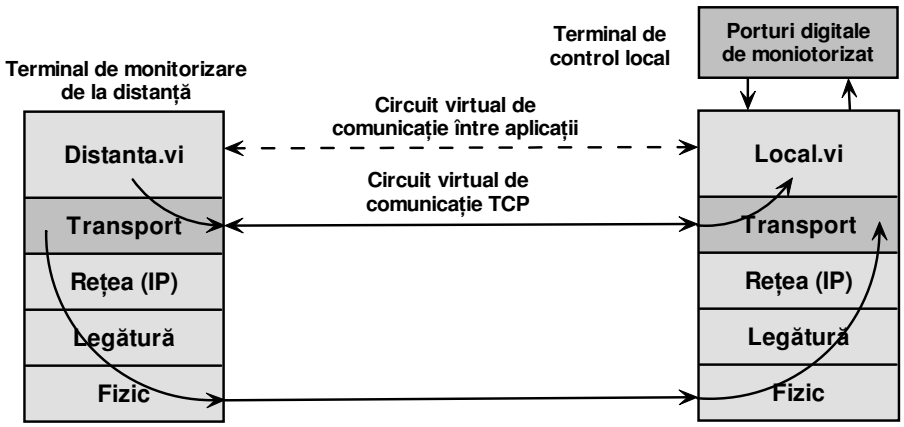

Fig. 14.5. Conexiunea virtuală a nivelului **OSI 4 - Transport.**

Acestea pot fi utilizate, în construcţia instrumentelor virtuale, apelând funcţiile prezentate în continuare [4].

Funcţii TCP.

• **TCP Open Connection** este funcţia care deschide o conexiune **TCP**, folosind adresa terminalului destinaţie şi portul aplicaţiei specificate, având următorii parametri:

- **address** este adresa **IP** a terminalului cu care se doreşte stabilirea conexiunii. Adresa poate fi în formată numeric, zecimal cu punct, sau folosind numele simbolic, caz în care este nevoie ca acesta să fie furnizat de un server de nume (**DNS**).
- **remote port** este numărul portului care identifică aplicaţia de pe terminalul de la distanţă cu care vrem să stabilim conexiunea.
- **timeout ms** este timpul de aşteptare în milisecunde. Dacă operaţia nu se termină în acest timp funcţia îşi încheie execuţia şi returnează un mesaj de eroare. Are valoarea implicită 60.000 ms (un minut). Valoarea –1 defineşte timp de aşteptare nelimitat.
- **error in** este conectorul de intrare pentru cluster-ul care descrie erorile apărute în timpul execuţiei programului până la intrarea în această funcţie. Valoarea implicită este "no error". Cluster-ul **error in** conţine următorii parametrii:
- **status** are valoarea TRUE în cazul apariţiei unei erori. Dacă **status** are valoarea TRUE, funcţia nu se mai execută, iar **error out** conţine aceeaşi informaţie ca şi **error in**.
- **code** este codul erorii. Valoarea 0 indică absenţa unei erori.
- **source** reprezintă sursa erorii **şi** are ca valoare numele funcţiei în care a apărut eroarea, urmat de mesajul de eroare, iar în cazul execuţiei fără erori, numele ultimei funcţii executate, urmat de mesajul „no error".
- **local port** este numărul portului local pe care se deschide conexiunea. Unele servere permit conexiuni numai clienţilor care folosesc numere de port într-un anumit domeniu. Dacă valoarea este 0, sistemul de operare alocă dinamic un port nefolosit.
- **connection ID** este identificatorul conexiunii TCP.

- **error out** conţine informaţii despre erorile apărute. Dacă **error in** indică prezenţa unei erori atunci **error out** conţine aceleaşi informaţii ca şi **error in**; altfel, descrie erorile apărute la execuţia funcţiei curente.

• **TCP Listen** este funcţia care deschide un port de ascultare ce aşteaptă o cerere de conexiune din partea unei aplicaţii client.
Când se lansează un proces de tip server, de ascultare la un port, nu se mai poate folosi într-un alt instrument virtual *(VI)* funcţia TCP Listen care să asculte la acelaşi port. Dacă un VI conţine două funcţii TCP Listen, în diagrama bloc, acestea trebuie să folosească numere de port diferite între ele şi diferite de alte porturi deschise pe terminalul respectiv. Funcţia are următorii parametri:

- **port** este numărul portului la care se va aştepta cererea de conexiune.
- **timeout ms** este timpul de aşteptare în milisecunde. Dacă operaţia nu se termină în acest timp, funcţia îşi încheie execuţia şi returnează un mesaj de eroare. Are valoarea implicită 25.000. Valoarea –1 defineşte un timp de aşteptare nelimitat.
- **error in** este conectorul de intrare pentru un cluster care descrie erorile apărute în execuţia programului, până la intrarea în această funcţie. Valoarea implicită este "no error". Cluster-ul **error in** conţine următorii parametrii:
- **status** are valoarea TRUE în cazul apariţiei unei erori. Dacă **status** are valoarea TRUE, funcţia nu se mai execută iar **error out** conţine aceeaşi informaţie ca şi **error in**.
- **code** este codul erorii. Valoarea 0 indică absenţa unei erori.
- **source** este sursa erorii. **source** are ca valoare numele funcţiei în care a apărut eroarea urmat de mesajul de eroare sau, în cazul execuţiei fără erori, numele ultimei funcţii executate urmat de mesajul „no error".
- **connection ID** este identificatorul conexiunii **TCP**.
- **remote address** este adresa **IP** a terminalului de la distanţă asociată conexiunii **TCP**. Adresa **IP** este în format numeric zecimal cu punct.
- **remote port** este portul asociat conexiunii TCP asociat aplicaţiei de pe terminalul de la distanţă.
- **error out** conţine informaţii despre erorile apărute. Dacă **error in** indică prezenţa unei erori atunci **error out** conţine aceleaşi informaţii ca şi **error in**; altfel, descrie erorile apărute la execuţia funcţiei curente.

• **TCP Read** este funcţia care citeşte un anumit număr de octeţi, proveniţi de la o conexiune **TCP**, returnând rezultatul, sub forma unui şir, la terminalul **data out** şi are următorii parametri:

- **mode** indică comportamentul funcţiei de citire. Există patru opţiuni:
- **Standard** (implicit). Aşteaptă până la recepţionarea întregului număr de octeţi cerut, sau până la expirarea timpului specificat. Returnează numărul de octeţi primiţi până la momentul respectiv. Dacă s-a primit un număr mai mic decât cel aşteptat returnează un mesaj de eroare de timeout.
- **Buffered**. Aşteaptă până la recepţionarea întregului număr de octeţi cerut, sau până la expirarea timpului specificat. Returnează numărul de octeţi cerut sau nimic. Dacă nu s-a primit numărul de octeţi cerut returnează o eroare de timeout.
- **CRLF.** Aşteaptă până la recepţionarea unui **CR** (carriage return) urmat de **LF** (line feed), sau până la expirarea timpului specificat. Returnează octeţii primiţi până la **CR** şi **LF**, inclusiv **CR** şi **LF** sau nimic. Dacă nu găseşte **CR** urmat de **LF** returnează un mesaj de eroare de timeout.
- **Immediate.** Aşteaptă până când primeşte ceva şi returnează datele. Aşteaptă până la expirarea timpului specificat de timeout, iar dacă nu primeşte nimic în acest timp returnează o eroare de timeout.
- **connection ID** este identificatorul conexiunii **TCP**.
- **bytes to read** specifică numărul maxim de octeţi care trebuie citiţi de la conexiunea **TCP**.
- **timeout ms** este timpul de aşteptare în milisecunde. Dacă operaţia nu se termină în acest timp, funcţia îşi încheie execuţia şi returnează un mesaj de eroare. Are valoarea implicită 25.000. Valoarea –1 defineşte un timp de aşteptare nelimitat.

- **error in** este conectorul de intrare pentru cluster-ul care descrie erorile apărute în execuţia programului până la intrarea în această funcţie. Valoarea implicită este "no error". Cluster-ul **error in** conţine următorii parametrii:
- **status** are valoarea TRUE în cazul apariţiei unei erori. Dacă **status** are valoarea TRUE, funcţia nu se mai execută iar **error out** conţine aceeaşi informaţie ca şi **error in**.
- **code** este codul erorii. Valoarea 0 indică absenţa unei erori.
- **source** este sursa erorii. **source** are ca valoare numele funcţiei în care a apărut eroarea, urmat de mesajul de eroare, sau în cazul execuţiei fără erori, numele ultimei funcţii executate urmat de mesajul „no error".
- **connection ID out** are aceeaşi valoare ca şi **connection ID.**
- **data out** este terminalul de ieşire al datelor citite de la conexiunea TCP.
- **error out** conţine informaţii despre erorile apărute. Dacă **error in** indică prezenţa unei erori, atunci **error out** conţine aceleaşi informaţii ca şi **error in**; altfel descrie erorile apărute la execuţia funcţiei curente.

• **TCP Write** este funcţia care permite scrierea datelor la o conexiune TCP, având următorii parametri:
- **connection ID** este identificatorul conexiunii TCP.
- **data in** conţine datele care vor fi scrise la conexiunea TCP.
- **timeout ms** este timpul de aşteptare, în milisecunde. Dacă operaţia nu se termină în acest timp funcţia îşi încheie execuţia şi returnează un mesaj de eroare. Are valoarea implicită 25.000. Valoarea –1 defineşte un timp de aşteptare nelimitat.
- **error in** este conectorul de intrare pentru cluster-ul ce descrie erorile apărute în execuţia programului, până la intrarea în această funcţie. Valoarea implicită este "no error". Cluster-ul **error in** conţine următorii parametri:
- **status** are valoarea TRUE în cazul apariţiei unei erori. Dacă **status** are valoarea TRUE, funcţia nu se mai execută iar **error out** conţine aceeaşi informaţie ca şi **error in**.
- **code** este codul erorii. Valoarea 0 indică absenţa unei erori.
- **source** este sursa erorii. **source** are ca valoare numele funcţiei în care a apărut eroarea urmat de mesajul de eroare, sau în cazul execuţiei fără erori, numele ultimei funcţii executate, urmat de mesajul „no error".
- **connection ID out** are aceeaşi valoare ca şi **connection ID.**
- **bytes written** este numărul de octeţi trimişi la conexiunea TCP.
- **error out** conţine informaţii despre erorile apărute. Dacă **error in** indică prezenţa unei erori atunci **error out** conţine aceleaşi informaţii ca şi **error in**; altfel, descrie erorile apărute la execuţia funcţiei curente.

• **TCP Close Connection** este funcţia care închide o conexiune TCP, având următorii parametri:
- **connection ID** este identificatorul conexiunii TCP care va fi închisă.
- **abort** indică dacă conexiunea va fi închisă normal (varianta implicită) sau forţat. De obicei, acest parametru este ignorat.
- **error in** este conectorul de intrare pentru cluster-ul care descrie erorile apărute în execuţia programului, până la intrarea în această funcţie. Valoarea implicită este "no error". Cluster-ul **error in** conţine următorii parametri:
- **status** are valoarea TRUE în cazul apariţiei unei erori. Dacă **status** are valoarea TRUE, funcţia nu se mai execută iar **error out** conţine aceeaşi informaţie ca şi **error in**.
- **code** este codul erorii. Valoarea 0 indică absenţa unei erori.
- **source** este sursa erorii. **source** are ca valoare numele funcţiei în care a apărut eroarea urmat de mesajul de eroare sau, în cazul execuţiei fără erori, numele ultimei funcţii executate urmat de mesajul „no error".
- **connection ID out** are aceeaşi valoare ca şi **connection ID.**

- **error out** conţine informaţii despre erorile apărute. Dacă **error in** indică prezenţa unei erori atunci **error out** conţine aceleaşi informaţii ca şi **error in**; altfel, descrie erorile apărute la execuţia funcţiei curente.

14.2.2. Componente de comunicaţie la nivelul de sesiune, prezentare şi aplicaţie (OSI 5,6,7), bazate pe server-ul de comunicaţie DataSocket.

Serverul de comunicaţie **DataSocket** este o suită de componente care permit interfaţarea unor instrumente virtuale distribuite, prin înglobarea unor funcţii specifice nivelelor **OSI 5, 6, 7,** aşa cum se poate vedea în **Fig. 14.6**. Acest server permite eliminarea funcţiilor de formatare a datelor, filtrarea accesului clienţilor la servicii şi distribuirea informaţiilor între terminale.

Prezentare generală a tehnologiei DataSocket.

Folosind **LabVIEW** utilizatorii pot dezvolta uşor (chiar fără cunoştinţe de programare) aplicaţii care să permită afişarea panoului frontal al unui instrument virtual, sau ca o pagină Web.

Imaginile sunt uşor de transmis prin reţea, însă utilizatorii preferă soluţii interactive care să permită controlul de la distanţă şi îmbunătăţirea performanţelor. Aceasta înseamnă transferul valorii datelor neprocesate, în locul unor imagini de mari dimensiuni. Tehnologia **DataSocket** a apărut ca urmare a acestor cerinţe ale utilizatorilor.

DataSocket este o tehnologie de programare care simplifică schimbul de date, in timp real, între aplicaţii distribuite pe mai multe terminale, introdusă de compania National Instruments.

Folosind **DataSocket** programatorii pot transfera eficient date neprocesate prin reţea şi pot răspunde cererilor provenite de la mai mulţi utilizatori, fără creşterea complexităţii programării la nivelul **TCP**.

Fundamentarea tehnologiei.

Interconectarea unor componente hardware şi software a ridicat întotdeauna anumite probleme de compatibilitate. Pentru a rezolva aceste probleme, la nivel hardware, trebuie să se ţină cont de nivelul semnalelor, impedanţa circuitelor, etc. La nivelul componentelor *software* trebuie rezolvată problema schimbului de date între aplicaţii.

În faza iniţială, trebuie obţinute datele experimentale, folosind instrumente de achiziţie şi unelte din biblioteci de funcţii, iar pentru comunicaţia între aplicaţii se foloseşte un alt set de instrumente virtuale.

Unele aplicaţii salvează rezultatele într-un fişier, altele folosesc resurse de comunicaţie în reţea, de tipul suitei de protocoale **TCP/IP**, **DDE**, sau funcţii de control **ActiveX**. Fiecare mecanism de intrare-ieşire (I/O) are particularităţile sale, ceea ce impune cunoaşterea lor.

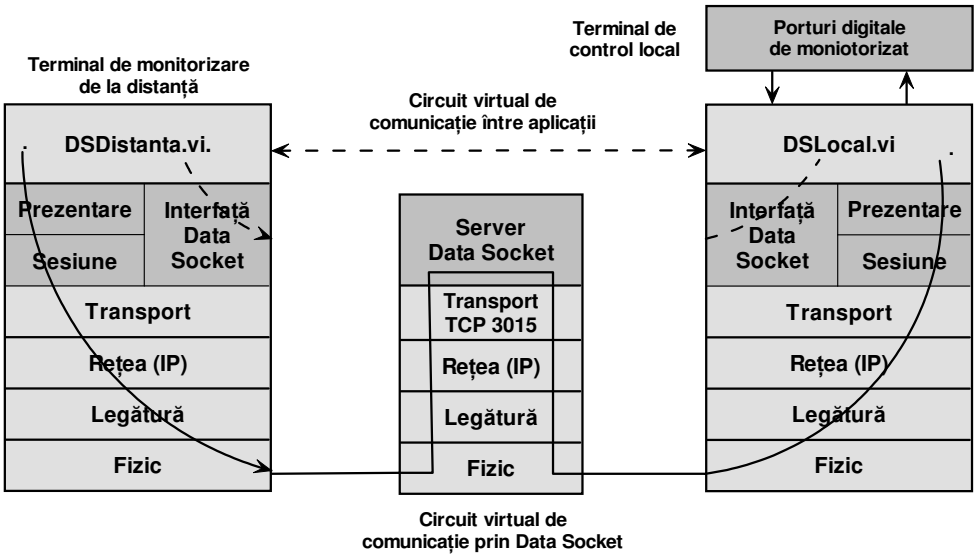

Fig. 14.6. Principiul comunicaţiei prin DataSocket şi poziţionarea componentelor în raport cu modelul de referinţă OSI.

Tehnologia DataSocket este o interfaţă de sine stătătoare, uşor de aplicat, care asigură accesul la mecanisme de I/O, fără a fi nevoie ca utilizatorul să folosească componente de interfaţare, formatare a datelor şi programare, de nivel redus. DataSocket înglobează tehnologii de comunicaţie, pentru măsurări şi automatizări, în acelaşi fel cum un browser web conţine diferite aplicaţii de reţea într-una singură, uşor de folosit.

Funcţii DataSocket.

• **Launch DS Server if Local URL** este funcţia care lansează automat serverul de comunicaţie DataSocket, de pe terminalul local, dacă adresa uniformă pentru localizarea resurselor (URL - *Uniform Resource Locator*) de intrare se referă la acesta (`dstp://localhost/wave`).

• **DataSocket Read** este funcţia care citeşte date de la conexiunea specificată în câmpul URL şi le returnează spre aplicaţie, având următorii parametri:
○ **URL** este identificatorul sursei de date.
○ **type (variant)** specifică tipul datelor care vor fi citite şi defineşte tipul de date al terminalului de ieşire, implicit este *Variant*, care poate fi oricare.
○ **ms timeout** specifică timpul de aşteptare pentru actualizarea datelor. Acest timp este ignorat dacă parametrul boolean **wait for updated value** are valoarea FALSE şi o valoare iniţială a fost recepţionată.
○ **error in** este un cluster care descrie starea erorilor înainte de execuţia acestei funcţii. Dacă **error in** indică apariţia unei erori în execuţia programului înainte de acest VI, acesta din urmă nu se mai execută şi informaţia despre eroare este trimisă la terminalul **error out**. Dacă n-a apărut nici o eroare, acest VI se execută normal şi pune propriile setări de eroare în **error out**. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
○ **status** are valoarea TRUE dacă a apărut o eroare înainte de apelarea acestui VI. Dacă **status** este TRUE valoarea parametrului **code** este diferită de zero. Dacă **status** este FALSE **code** poate avea valoarea zero sau valoarea unui cod de avertizare.
○ **code** este numărul de identificare al unei erori sau al unui avertisment. Dacă **status** este TRUE **code** are o valoare diferită de zero. Dacă **status** este FALSE **code** are valoarea 0 sau valoarea unui cod de avertizare. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
○ **source** este un şir care indică originea unei eventuale erori. De obicei se indică numele VI-ului în care a apărut eroarea.

o **wait for updated value**. Dacă acest parametru are valoarea TRUE funcţia aşteaptă actualizarea datelor. Dacă valorile au fost actualizate de la ultima citire funcţia îşi încheie execuţia. Altfel, aşteaptă timpul specificat de **ms timeout** pentru o actualizare. Dacă nu are loc o actualizare în acest timp este returnată valoarea curentă şi este setat parametrul de ieşire **timed out** la valoarea TRUE.
o Dacă **wait for updated value** este FALSE funcţia returnează valoarea curentă a datelor. Timpul setat de **ms timeout** este folosit numai dacă încă nu s-a primit nici o valoare.
o **duplicate URL** este URL-ul care specifică conexiunea DataSocket.
o **data** este rezultatul operaţiei de citire. Dacă expiră timpul **ms timeout** valoarea returnată la terminalul **data** este ultima valoare citită. Dacă expiră timpul **ms timeout** şi nu s-a citit nimic, sau tipul datelor este incompatibil se returnează valoarea 0 sau alt identificator echivalent.
o **timed out** returnează TRUE dacă timpul **ms timeout** a expirat, în aşteptarea unei actualizări sau a unei valori iniţiale.
o **error out** este un cluster care descrie starea erorii după execuţia VI-ului.
o **status** are valoarea TRUE dacă a apărut o eroare sau FALSE dacă execuţia a decurs fără erori. Dacă **status** este TRUE valoarea parametrului **code** este diferită de zero. Dacă **status** este FALSE **code** poate avea valoarea zero sau valoarea unui cod de avertizare.
o **code** este numărul de identificare al unei erori sau al unui avertisment. Dacă **status** este TRUE **code** are o valoare diferită de zero. Dacă **status** este FALSE **code** are valoarea 0 sau valoarea unui cod de avertizare. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
o **source** este un şir care indică originea unei eventuale erori. De obicei se indică numele VI-ului în care a apărut eroarea.

• **DataSocket Write** este funcţia care scrie date la conexiunea specificată prin URL, având următorii parametri:
o **URL** este identificatorul destinaţiei datelor.
o **data** reprezintă informaţiile care vor fi scrise la conexiunea DataSocket.
o **error in** este un cluster care descrie starea erorilor înainte de execuţia acestei funcţii. Dacă **error in** indică apariţia unei erori în execuţia programului înainte de acest VI, acesta nu se mai execută şi informaţia despre eroare este trimisă la terminalul **error out**. Dacă nu a apărut nici o eroare, acest VI se execută normal şi pune propriile setări de eroare în **error out**. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
o **status** are valoarea TRUE dacă a apărut o eroare înainte de apelarea acestui VI. Dacă **status** este TRUE valoarea parametrului **code** este diferită de zero. Dacă **status** este FALSE **code** poate avea valoarea zero sau valoarea unui cod de avertizare.
o **code** este numărul de identificare al unei erori sau al unui avertisment. Dacă **status** este TRUE, **code** are o valoare diferită de zero. Dacă **status** este FALSE, **code** are valoarea 0 sau valoarea unui cod de avertizare. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
o **source** este un şir care indică originea unei eventuale erori. De obicei se indică numele VI-ului în care a apărut eroarea.
o **duplicate URL** este URL-ul care specifică conexiunea DataSocket.
o **error out** este un cluster care descrie starea erorii după execuţia VI-ului.
o **status** are valoarea TRUE dacă a apărut o eroare, sau FALSE dacă execuţia a decurs fără erori. Dacă **status** este TRUE, valoarea parametrului **code** este diferită de zero. Dacă **status** este FALSE, **code** poate avea valoarea zero, sau valoarea unui cod de avertizare.
o **code** este numărul de identificare al unei erori, sau al unui avertisment. Dacă **status** este TRUE, **code** are o valoare diferită de zero. Dacă **status** este FALSE, **code** are valoarea 0 sau valoarea unui cod de avertizare. Se va folosi VI-ul de tratare a erorilor pentru interpretarea codului erorii şi afişarea mesajului corespunzător.
o **source** este un şir care indică originea unei eventuale erori. De obicei se indică numele VI-ului în care a apărut eroarea.

Exemplu de difuzare a datelor prin DataSocket.

Se consideră un sistem de măsurare şi se doreşte distribuirea rezultatelor măsurătorilor între mai multe sisteme de calcul. Un exemplu tipic este cazul unui laborator în care un calculator controlează experimentul, în timp ce câţiva studenţi fac analiza în timp real, pe staţii de lucru individuale.

Pentru a distribui datele folosind bibliotecile TCP trebuie parcurse următoarele etape :

- Se alege un număr de port TCP nefolosit de alte aplicaţii de tip server care operează pe acel sistem
- Se stabileşte o convenţie de transmitere a datelor,
- Se configurează aplicaţia server să asculte la portul selectat şi să accepte o conexiune, când primeşte o cerere de la o aplicaţie client,
- Se configurează aplicaţia server să formateze datele şi să răspundă la toate conexiunile cerute de aplicaţiile client,
- Se configurează aplicaţiile client să ceară stabilirea unei conexiuni la adresa şi portul destinaţie (*socket* destinaţie) al aplicaţiei server, să proceseze datele şi să le afişeze.

Dacă se operează anumite modificări ale aplicaţiei server (de exemplu, adăugarea unui parametru de transmis) trebuie modificate şi aplicaţiile client în vederea compatibilizării formatului şi a structurii datelor.

Folosind bibliotecile de funcţii DataSocket, acelaşi lucru se realizează în două etape :

- Deschiderea unei conexiuni DataSocket, folosind un nume de resursă pentru a identifica datele;
- Scrierea datelor la blocul de interfaţare cu conexiunea, în urma procesării ultimelor valori achiziţionate.

În acest caz, programarea la nivel TCP/IP a fost deja făcută prin utilizarea componentelor de interfaţare DataSocket.

Tehnologia DataSocket a fost concepută pentru a răspunde nevoilor inginerilor automatişti şi metrologi, de a transmite date de măsurare de-a lungul unor reţele complexe. De exemplu, folosind TCP/IP programatorul trebuie să scrie, în aplicaţia server, o procedură pentru a converti datele măsurate într-o formă nestructurată, adică un şir continuu *(stream)* de octeţi, care va fi apoi transmis cu ajutorul blocurilor oferite de biblioteca **TCP Connection** şi, în mod asemănător în aplicaţiile client, o procedură care să analizeze şirul de octeţi primit pe care să-l transforme în structura de date originală.

DataSocket transferă datele folosind componente care transformă, diferite formate de structuri de date (şir, scalar, boolean, forme de unda, etc), într-un format propriu de descriere. Operaţiile de citire şi scriere fac conversia datelor în şi din forma nestructurată de şir continuu, transferat apoi în reţea, într-un mod transparent pentru utilizator.

Se elimină astfel necesitatea de a scrie proceduri complicate de conversie. În plus, folosind formatul propriu DataSocket se pot asocia, datelor transmise, anumite atribute definite de utilizator.

De exemplu, valorilor achiziţionate de la un senzor de temperatură, se poate ataşa o marcă de timp, sau unui şir de valori discrete, o rata de eşantionare. Acestea pot fi foarte utile pentru refacerea la destinaţie a informaţiei achiziţionate.

Asocierea unei surse de date la un URL.

În general, o bibliotecă de I/O are o funcţie „**Open**" care primeşte ca parametru de intrare un nume, sau un număr, ca identificator al resursei (sau al dispozitivului) către care, sau de la care se face transferul datelor. Dacă se lucrează cu fişiere, numele resursei este calea fişierului, iar pentru conexiunile care folosesc TCP/IP, numele resursei are două părţi: numele terminalului şi numărul portului care identifică aplicaţia.

Acesta, aşa cum s-a putut vedea în capitolul 4, se numeşte *socket*. DataSocket foloseşte ca identificator al resursei un **URL** *(Uniform Resource Locator)* asemănător adreselor web folosite de un browser web. De exemplu, URL-ul `http://www.natinst.com/datasocket` îi spune browser-ului să folosească protocolul de transfer HTTP *(Hyper Text Transfer Protocol),* asociat suitei de protocoale TCP/IP, pentru a se conecta la terminalul cu numele simbolic `www.natinst.com`.

După aceasta, va cere aplicaţiei server web să transfere pagina web implicită din directorul numit `datasocket`.

În adresa resursei se defineşte, la început, protocolul de transfer (`file://` pentru citirea unui fişier local, `http://` pentru *Hyper Text Transfer Protocol,* `ftp://` pentru *File Transfer Protocol* etc.). Deoarece DataSocket foloseşte propriul protocol de transfer pentru datele de măsurare, acesta va fi identificat prin `dstp://`. De exemplu, în cazul accesării unor date între terminalele din laboratorul prezentat mai sus se poate folosi următoarea adresă: `dstp://serverdatasocket/wave1`.

Protocolul de transfer este în acest caz `dstp`, iar adresa se interpretează astfel: „deschide o conexiune DataSocket cu terminalul al cărui nume simbolic este `serverdatasocket` şi transferă informaţia corespunzătoare semnalului `wave1`.

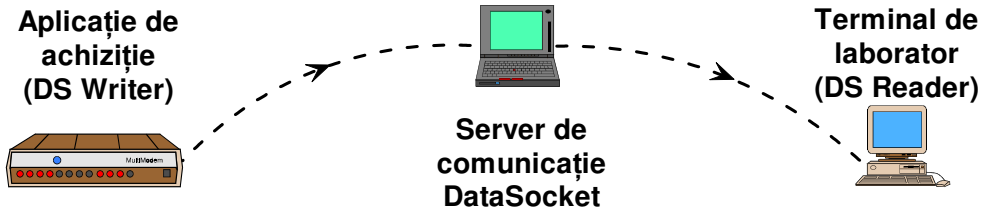

Fig. 14.7. Componentele unui sistem de comunicaţie prin DataSocket.

Componentele tehnologiei DataSocket.

DataSocket are două categorii de componente: **DataSocket API** (aplicaţii client DataSocket Writer şi DataSocket Reader) şi **DataSocket Server**, aşa cum se poate vedea în **figura 14.7.**

DataSocket **API** *(Application Programming Interface)* prezintă o singură interfaţă standard pentru comunicaţia cu mai multe tipuri de date, utilizată de mai multe limbaje de programare. DataSocket Server simplifică comunicaţia în reţele complexe (publice, private, sau mixte), acoperă nivelele **OSI 5, 6, 7** şi foloseşte pentru transport şi reţea suita de protocoale TCP/IP.

- **DataSocket API.**

O interfaţă API conţine un set de funcţii pe care programele le pot folosi pentru a apela rutine ale unei alte platforme software, de exemplu ale sistemului de operare. Astfel, un program poate realiza funcţiuni complexe trimiţând doar o singură instrucţiune unei interfeţe API.

DataSocket conţine o singură interfaţă API, bazată pe adresare URL, independentă de protocolul de transfer, limbaj, sau sistem de operare. Această interfaţă a fost proiectată pentru a simplifica distribuţia datelor binare şi este implementată sub formă de control ActiveX, biblioteci LabWindows/CVI, C sau ca un set de instrumente virtuale LabVIEW şi poate fi folosită în orice mediu de programare.

Interfaţa **API-DataSocket** converteşte automat datele de măsurare într-un şir continuu de octeţi, care este trimis apoi prin reţea, prin intermediul conexiunilor TCP. Aplicaţia **DataSocket**, care primeşte datele, converteşte automat şirul de octeţi la forma originală.

Acest proces automat de conversie scuteşte utilizatorul de necesitatea scrierii unor secvenţe de cod complicate.

API-ul DataSocket conţine patru operaţii de bază (deschidere, citire, scriere şi închidere) similare apelurilor standard de intrare-ieşire. Această interfaţă se poate folosi pentru a citi date de la următoarele aplicaţii:

- Fişiere locale
- Servere HTTP şi FTP
- Servere OPC (OLE for Process Control, OLE - Object Linking and Embedding);
- Servere DSTP

- **DataSocket Server.**

Server-ul DataSocket este o componentă de sine stătătoare folosită de programele care apelează interfaţa **API-DataSocket** pentru a transmite simultan, la viteze mari, date de măsurare între mai multe aplicaţii client. Conexiunile dintre clienţi, realizate de către aplicaţia server DataSocket, sunt iniţiate în mod automat şi gestionate individual în funcţie de tipul datelor şi scopul transferului.

În acest proces, de transmitere a datelor, sunt implicate trei componente: aplicaţia sursă, Server-ul de comunicaţie **DataSocket** şi aplicaţia destinaţie, **Fig. 14.8**.. Aplicaţia sursă foloseşte **API-DataSocket** pentru a scrie date la server; iar aplicaţia destinaţie foloseşte interfaţa pentru a citi date de la server. Atât aplicaţia sursă *(DataSocket Writer)* cât şi cea destinaţie *(DataSocket Reader)* sunt aplicaţii client ai server-ului **DataSocket**.

Cele trei componente pot opera pe acelaşi terminal, sau pe terminale diferite, alegerea variantelor depinzând de posibilitatea rutării pachetelor de date între aplicaţiile client şi server. Astfel, conexiunea dintre aplicaţiile client şi aplicaţia server trebuie să poată fi stabilită şi în situaţiile în care avem de-a face cu combinaţii de reţele public-private.

În cazul în care clienţii şi server-ul de comunicaţie sunt în aceeaşi reţea, acesta din urmă poate să opereze pe oricare dintre cele două terminale pe care operează aplicaţiile client, sau pe un terminal de sine stătător. În cazul în care aplicaţiile client operează pe terminale din reţele private separate între ele (fără rutare reciprocă a pachetelor de reţea), cu conectare la reţeaua publică prin **NAT** *(Network Address Translation)*, aplicaţia server trebuie să opereze pe un terminal din reţeaua publică, accesibil din reţelele private.

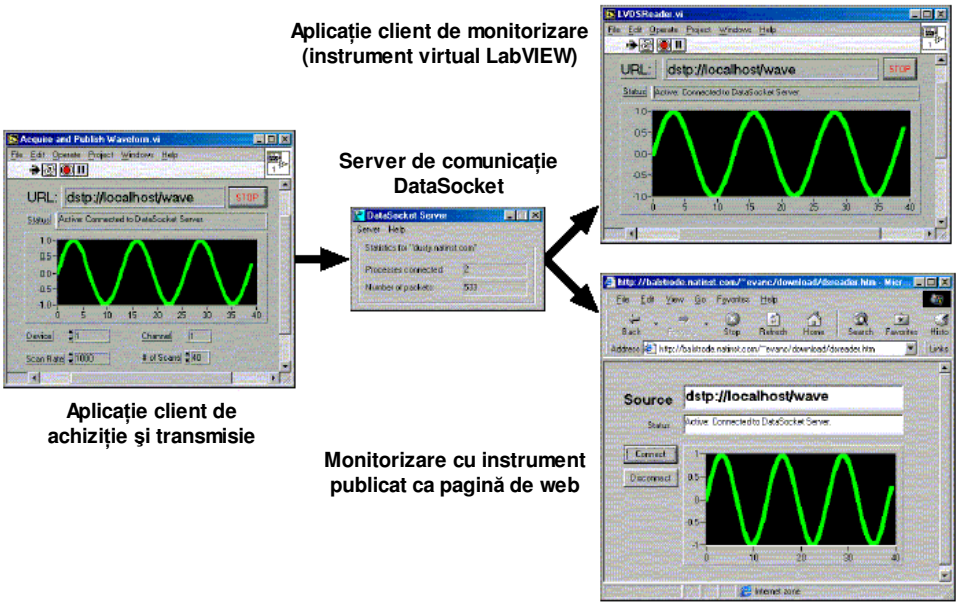

Fig. 14.8. Transmiterea datelor folosind server-ul DataSocket.

Server-ul DataSocket poate restricţiona accesul aplicaţiilor client la date, administrând drepturi şi permisiuni de iniţiere, scriere sau citire, în funcţie de adresa terminalelor. Aceste liste de control al accesului *(***ACL** - *Access Control Lists),* alături de alte opţiuni ale aplicaţiei, se pot configura cu ajutorul unei aplicaţii numită **DataSocket Server Manager.**

- **Aplicaţii bazate pe DataSocket.**

În continuare se prezintă, prin câteva exemple, modul de folosire a acestei tehnologii.

- **Monitorizarea variabilelor de proces folosind DataSocket.**

Se consideră cazul unei fabrici care realizează mai multe tipuri de produse. Fiecare tip de produs are propria linie de fabricaţie şi fiecare linie are un calculator care monitorizează variabilele de proces.

Se cere realizarea unei aplicaţii în **LabVIEW** care să monitorizeze, în mod continuu fiecare variabilă de proces. Aplicaţia trebuie să trimită datele prin reţeaua locală, în timpul cel mai scurt, către biroul central. Un calculator din biroul central adună informaţiile şi afişează situaţia curentă a variabilelor de proces, pentru fiecare linie de producţie, în parte.

Folosind componentele de comunicaţie din biblioteca **TCP/IP** ar trebui făcută câte o pereche aplicaţii client/server care să achiziţioneze datele de proces, să le transforme într-un şir continuu de octeţi, şi să le trimită spre terminalul din biroul central, pentru fiecare linie de fabricaţie în parte. Aplicaţia de pe terminalul din biroul central poate să conţină, sub forma unui set de instrumente virtuale, mai multe blocuri de tip client sau server.

Tehnologia DataSocket permite rezolvarea mai uşoară a problemei comunicaţiei între secţia de producţie şi biroul central.

În imaginea diagramei bloc, prezentată în **figura 14.8**, este ilustrat modul în care se pot citi şi scrie datele de proces, folosind interfaţa **DataSocket API**. Pentru că datele sunt trimise la server-ul **DataSocket**, aplicaţia care operează pe terminalul din biroul central nu trebuie să gestioneze conexiuni multiple, ci doar trebuie să citească informaţiile asociate fiecărei linii de fabricaţie.

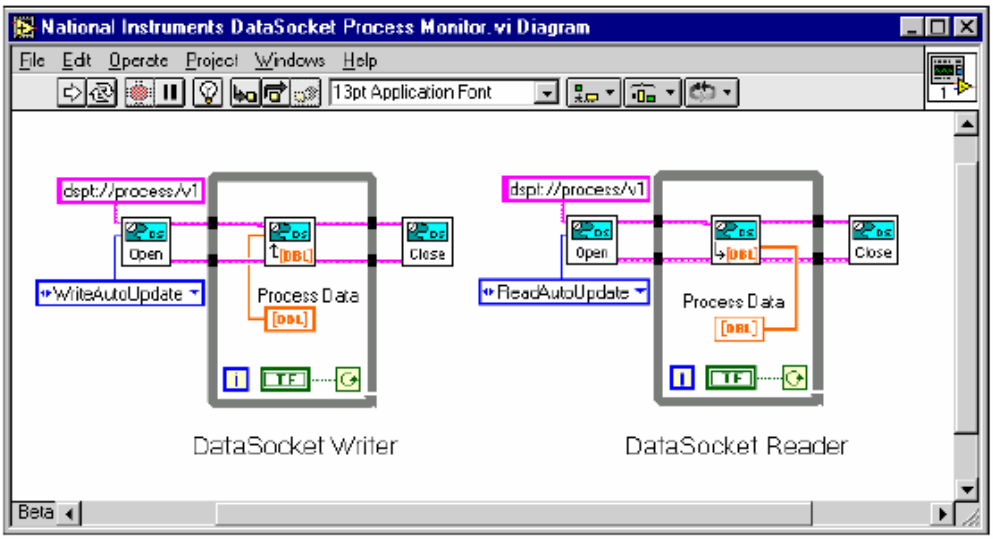

Fig. 14.9. Citirea şi scrierea datelor de proces folosind interfaţa DataSocket API.

Pornind de la acest exemplu, se pot imagina şi soluţii de control, dinspre terminalul din biroul central, a linilor de fabricaţie. În acest caz, aplicaţiile care operează pe terminalele din liniile de producţie, pot să conţină combinaţii de instrumente virtuale de tip client/server

14.3. Pagină de web interactivă folosind DataSocket.

Se consideră un laborator de procesarea semnalelor care este inclus într-un proces de învăţare la distanţă, prin Internet. Laboratorul are terminale de lucru (staţii) şi un terminal server, cu funcţiuni multiple, care deserveşte reţeaua locală a laboratorului. Acesta din urmă este conectat la reţeaua publică şi poate fi accesat de către cursanţii care utilizează terminale aflate în reţeaua publică.

Se doreşte prezentarea şi analizarea, pe aceste terminale, a semnalelor achiziţionate pe unul dintre terminalele din laborator (Aplicaţia de achiziţie DS Writer). Pe terminalele (staţiile) de lucru se folosesc aplicaţii de navigare (Internet Explorer, Netscape etc.) cu ajutorul cărora se pot transfera pagini de web interactive, conţinând componente DataSocket care se conectează la server şi citesc datele furnizate de aplicaţiile client DataSocket Writer, **Fig. 14.10**.

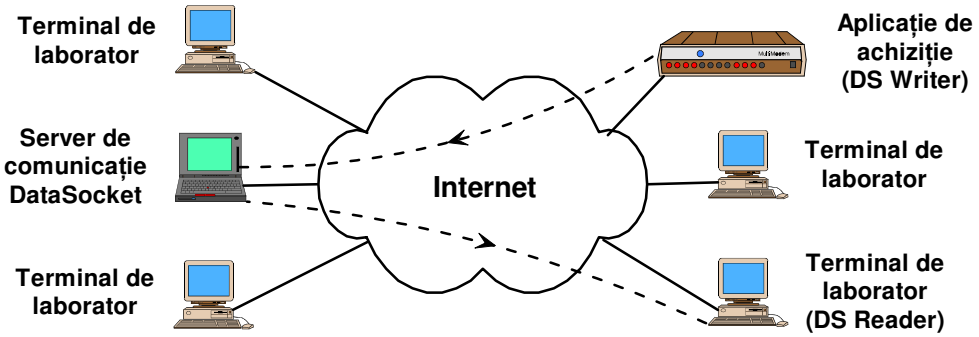

Fig. 14.10. Dezvoltarea unui laborator interactiv folosind tehnologia DataSocket.

Se presupune că există deja aplicaţia sursă, care achiziţionează datele şi le publică, folosind instrumente virtuale **DataSocket**. Se pune problema distribuirii datelor către cursanţii aflaţi în faţa terminalelor distribuite în reţea. Pentru a realiza pagina web interactivă se pot folosi diferite componente auxiliare de programare distribuită, de exemplu Visual Basic, pentru a crea interfaţa cu utilizatorul. Secvenţa de program *(script)*, proprie acestui mediu de programare, este inclusă în fişierul sursă HTML, care va fi apoi interpretat de către aplicaţia *browser*.

Citirea datelor de la server-ul **DataSocket** se face în trei etape:

1. deschiderea unei sesiuni de comunicaţie cu server-ul, folosind o componentă de control **ActiveX**,
2. citirea datelor de la server, furnizate acestuia de aplicaţia de scriere,
3. închiderea conexiunii cu server-ul **DataSocket**.

Pentru a putea parcurge aceste etape, este nevoie să se folosească următoarele componente software auxiliare, inserate în pagina web: Component-Works DataSocket şi Microsoft Visual Basic. Această pagină va fi apoi folosită pentru a citi, scrie, sau distribui date, spre alte aplicaţii din reţea.

În cazul acesta se poate opri sau reporni procesul de achiziţie, sau calculatorul pe care operează instrumentul virtual, fără a fi nevoie de a restabili conexiunile cu terminalele (staţiile de lucru). Acestea rămân conectate la server-ul de comunicaţie şi vor primi din nou date, în momentul în care procesul de achiziţie reîncepe şi vor fi trimise date spre server.

În continuare se prezintă realizarea componentei de citire DataSocket, şi modul în care se include secvenţa de program într-o pagină web (fişierul sursă HTML).

DataSocket este o unealtă de programare care permite scrierea, citirea, şi distribuirea datelor între aplicaţii cu diferite surse şi destinaţii şi poate accesa informaţii din fişiere locale, de pe servere HTTP şi FTP. În cazul utilizării funcţiilor de intrare/ieşire (I/O) de uz general, a funcţiilor de comunicaţie directă folosind suita TCP/IP, sau a cererilor de transfer prin protocoale FTP/HTTP, trebuie scrisă câte o secvenţă de cod separată, pentru fiecare protocol în parte. DataSocket oferă, pentru aceste protocoale de comunicaţie, o interfaţă API unificată.

Pentru conectarea la sursa de date trebuie specificată adresa resursei (URL), care se poate referi la mai multe tipuri de surse de date, după cum se specifică în prefix. Acest prefix se numeşte *URL scheme*. Alături de protocoalele de transfer mai sus menţionate, DataSocket are în plus propriul protocol *(scheme)* pentru distribuirea datelor în timp real, **dstp** (DataSocket Transfer Protocol).

Descrierea pachetelor auxiliare.

Pachetul ComponentWorks DataSocket include trei componente:

1. **DataSocket ActiveX Control** – componentă care conectează aplicaţiile la sursele de date şi distribuie între ele datele. Pentru că **DataSocket** este o componentă care foloseşte control ActiveX, se poate folosi în conjuncţie cu alte componente de acest tip (containere ActiveX), cum ar fi Visual Basic, Visual C++, sau Borland Delphi.
2. **DataSocket Server** este un program executabil care comunică şi schimbă date între două aplicaţii, folosind protocolul dstp. În exemplul prezentat aici, **DataSocket Server** operează pe server-ul din laborator, accesibil de către terminalele din reţeaua publică, sau privată.

3. **DataSocket Server Manager** este un utilitar de configurare pentru aplicaţia **DataSocket Server**. Acesta trebuie utilizat pentru configurarea parametrilor de funcţionare a aplicaţiei **DataSocket Server**, înainte de lansarea în execuţie a acestuia. **DataSocket Server Manager** permite, printre altele, stabilirea drepturilor de administrare de la distanţă, accesul, crearea resurselor, citirea sau scrierea datelor, a terminalelor din reţea.

În fişierul care constituie pagina web, se va crea mai întâi o componentă **DataSocket** care va avea rolul de a stabili conexiunea cu server-ul de comunicaţie, de a citi şi afişa datele şi de a închide conexiunea.

În acest exemplu se va crea această componentă folosind elemente de control **ActiveX** şi secvenţe de cod **Visual Basic**, care se va salva ca un element de control **ActiveX** (fişier cu extensia .OCX), care va fi inserat apoi într-o pagină web.

Realizarea componentei de citire presupune parcurgerea a patru etape:

1. Crearea componentei **DSReader** folosind elemente **ActiveX** în Visual Basic – se va folosi ComponentWorks DataSocket şi Visual Basic 5.0 Control Creation Edition. Apoi această componentă va fi inclusă apoi în fişierul HTML care reprezintă sursa pentru pagina web. Ea are rolul de a citi şi afişa datele de la server-ul **DataSocket**, primite de acesta de la aplicaţia care face achiziţia.
2. Crearea fişierului HTML – se foloseşte Visual Basic Application Setup Wizard pentru a dezvolta pagina web **DSReader**.
3. Afişarea datelor primite în timp real în pagina web **DSReader** – se deschide pagina web **DSReader** în Internet Explorer. După stabilirea conexiunii cu server-ul **DataSocket**, se citesc şi se afişează semnalele trimise de aplicaţia **DSWriter**.
4. Interacţionarea cu datele afişate – modificarea componentei **DSReader** astfel încât să permită interacţiunea cu datele.

Se vor folosi următoarele utilitare care sunt furnizate în mod gratuit: ComponentWorks 2.0, sau o versiune a mediului Visual Basic 5.0 Control Creation Edition şi Internet Explorer.

Crearea componentei DSReader folosind obiecte ActiveX în Visual Basic.

Această componentă permite conectarea la DataSocket Server, citirea datelor furnizate acestuia de către aplicaţia de achiziţie şi afişarea lor.

Etapele care trebuie urmate pentru deschiderea unui nou proiect şi încărcarea elementelor de control ActiveX, sunt următoarele:

1. Se lansează Visual Basic 5.0 Control Creation Edition;
2. În caseta de dialog New Project se selectează ActiveX Control;
3. Se selectează Views»Toolbox. Cu butonul din dreapta, se apasă pe Toolbox şi se selectează Components;
4. Se selectează National Instruments CW DataSocket şi National Instruments CW UI. Dacă elementele de control ComponentWorks nu apar în lista Controls, se foloseşte butonul Browse şi se selectează cwds.ocx şi cwui.ocx din directorul System al sistemului de operare Windows.

Figura 14.11 prezintă şablonul *(form)* componentei **DSReader**. Şablonul este fereastra gri în care se dispun elementele de control şi indicatoarele, pentru a crea interfaţa cu utilizatorul. Se vor parcurge următoarele etape:

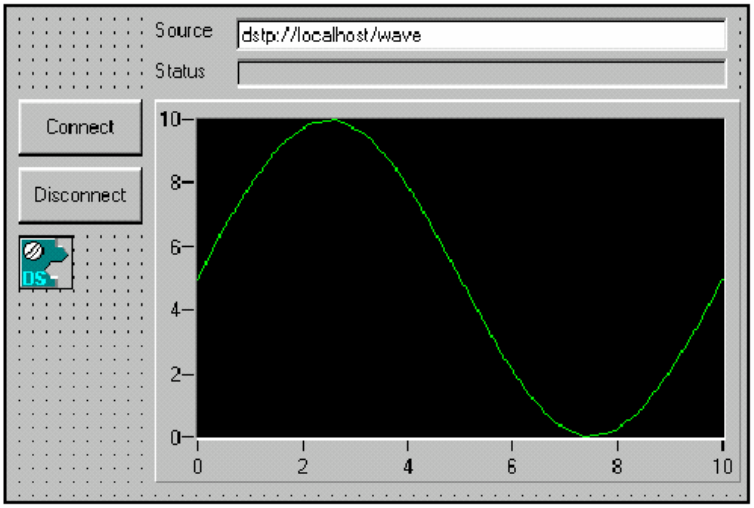

Fig. 14.11. Şablonul componentei DSReader.

1.  Se plasează un obiect de tip CommandButton pe şablon; se scrie în câmpul Name textul ConnectButton, iar în câmpul Caption, Connect.

2. Se procedează la fel pentru obţinerea butonului Disconnect.

3.  Se plasează pe şablon un obiect de tip TextBox pentru adresa sursei (URL). Se păstrează setările implicite şi se adaugă o etichetă pentru a identifica sursa de date.

4.  Se inserează un obiect de tip Label pentru afişarea stării conexiunii; se scrie în câmpul Name textul StatusLbl, se şterge valoarea implicită pentru câmpul Caption, se setează pentru Border Style opţiunea Fixed Single şi se adaugă o etichetă.

5.  Se plasează pe şablon un obiect de tip control CWGraph cu setările implicite.

6.  Se plasează pe şablon un obiect de tip control DataSocket. Acesta nu va fi vizibil în timpul rulării.

Scrierea secvenţei de cod în Visual Basic.

După inserarea elementelor de control pe şablon, pentru a trata evenimentele, trebuie scrisă secvenţa de cod Visual Basic. Un eveniment poate fi o acţiune, cum ar fi mişcarea mouse-ului, sau o schimbare de stare, cum ar fi încheierea achiziţiei.

Pentru a scrie procedura de tratare a evenimentelor în Visual Basic, pentru un element de control ActiveX, pentru a deschide editorul, se apasă de două ori pe el. Acesta generează automat o structură implicită a procedurii. Această structură conţine numele elementului de control, evenimentul implicit, şi parametrii formali.

1. În timpul rulării, la apăsarea butonului Connect, elementul de control DataSocket trebuie să stabilească conexiunea cu server-ul, să citească şi să actualizeze datele în mod automat. Pentru aceasta se apasă de două ori pe butonul Connect şi se adăugă următoarele linii de cod:

```
Private Sub ConnectButton_Click()
CWDataSocket1.ConnectTo Text1.Text, cwdsReadAutoUpdate
End Sub
```

2. Când elementul de control DataSocket primeşte date, acestea trebuie afişate de către ecranul grafic. Pentru aceasta, se apasă de două ori pe elementul de control DataSocket şi se adăugă următoarele linii de cod:

```
Private Sub CWDataSocket1_OnDataUpdated(ByVal Data As    CWDSLib.CWData)
' For extra error checking. Only plot the value if it's an array.
If IsArray(Data.Value) Then
CWGraph1.PlotY Data.Value
End If
End Sub
```

3. Pentru ca în câmpul de stare să se afişeze starea conexiunii, se apasă de două ori pe elementul de control DataSocket, se selectează OnStatusUpdated din lista de evenimente şi se adăugă următoarele linii de cod:

```
Private Sub CWDataSocket1_OnStatusUpdated(ByVal Status As Long, ByVal Error
As Long, ByVal Message As String)
StatusLbl.Caption = Message
End Sub
```

4. La apăsarea butonului Disconnect elementul de control DataSocket trebuie să închidă conexiunea şi să păstreze ultimele valori primite, pentru a permite utilizatorului să le acceseze şi după închiderea conexiunii. Se apasă de două ori pe butonul Disconnect, şi se adăugă următoarele linii de cod:

```
Private Sub DisconnectButton_Click()
CWDataSocket1.Disconnect
End Sub
```

Generarea obiectului ActiveX.

Pentru generarea obiectului ActiveX trebuie parcurse următoarele etape:

1. Se deschide Project Explorer, iar din meniul acestuia View»Project Explorer;
2. Se selectează Project1 în Project Explorer, iar în fereastra Properties se scrie Reader în câmpul Name;
3. Se apasă de două ori pe şablon pentru a-l selecta, iar în fereastra Properties, în câmpul Name scriem DSReader.

Salvarea proiectului.

Deoarece numele fişierului nu este identic cu numele obiectului ActiveX, ele trebuie salvate separat.

1. Se selectează comanda File » Save Project As;
2. Se salvează elementul de control ca DSReader.ctl;
3. Se salvează proiectul ca DSReader.vbp.

Generarea fişierului de control ActiveX.

1. Se selectează File » Make DSReader.ocx. Visual Basic crează obiectul ActiveX şi îl salvează în locaţia în care se află proiectul.
2. Se selectează Project » Reader Properties » Component, şi se setează Version Compatibility, la Binary Compatibility.

Observaţie: Fişierul HTML foloseşte identificatorul **GUID** pentru a adresa elementul de control. În modul **binary compatibility**, Visual Basic nu modifică identificatorul GUID (Globally Unique Identifier) la fiecare salvare a fişierului .OCX

4. Se salvează proiectul File » Save.

Crearea fişierului HTML.

Folosind Application Setup Wizard (în Visual Basic 5) sau Package and Deployment Wizard (în Visual Basic 6) se va genera un pachet de distribuţie prin Internet, care conţine un fişier arhivă (cu extensia .cab) şi un fişier HTML care conţine componenta DataSocket Reader. Fişierul arhivă conţine informaţii despre componenta DSReader, şi este descărcat, expandat şi instalat automat de Internet Explorer, pentru ca utilizatorii celorlalte calculatoare să poată vedea componenta DSReader, folosind o aplicaţie de tip *browser web*.

Observaţie: dacă se modifică componenta DSReader în Visual Basic, trebuie refacut fişierul .OCX şi pachetul de distribuţie prin Internet (fişierele .cab şi .htm).

Deşi această pagină web este simplă, ea afişând numai componenta DSReader, folosind un editor HTML, se pot adăuga căsuţe text, câmpuri în culori, imagini, sau alte obiecte.

Afişarea datelor în timp real folosind pagina web DSReader.

Funcţionarea laboratorului se poate simula folosind următoarea procedură:

1. Se lansează DataSocket Server din meniul Start Programs » National Instruments ComponentWorks » DataSocket Server.
2. Se deschide, în Internet Explorer, fişierul DSReader.htm.
3. Pentru stabilirea conexiunii se apasă butonul Connect din pagina web DSReader.htm. Se poate observa că DataSocket Server indică faptul că există un proces conectat, iar în pagina web, în câmpul de afişare a stării, se confirmă stabilirea conexiunii. Deoarece încă nu se trimit date la server-ul DataSocket, nu se afişează nimic nici pe graficul DSReader.

Observaţie: pentru ca un terminal din reţea să poată publica date folosind DataSocket Server, trebuie să aibă drept de creare a obiectului care reprezintă semnalul de transmis. Aceste drepturi se stabilesc folosind DataSocket Server Manager, unde se introduce adresa IP a terminalului, în lista grupului Creators permission.

4. Se lansează programul executabil DSWriter.exe aflat în directorul ...\ComponentWorks\samples\VisualBasic\DataSocket\.
5. După introducerea adresei resursei `dstp://localhost/wave` în câmpul URL al aplicaţiei DSWriter şi apăsarea butonului Connect (AutoUpdate) DSReader începe să primească date de la DSWriter, prin intermediul server-ului DataSocket.

Observaţie: în cazul în care server-ul de comunicaţie se află pe alt terminal decât cel pe care operează aplicaţia DSReader, trebuie înlocuit numele simbolic, al terminalului local, `localhost` cu numele simbolic, sau adresa IP, a terminalului pe care se află server-ul de comunicaţie. Numele simbolice pot fi folosite doar în cazul în care acestea sunt ţinute în evidenţa unor server-e de nume (DNS), iar terminalele din reţea au acces la acestea.

Modificarea paginii web pentru a permite interacţiunea cu datele.

Dacă se doreşte o pagină de web interactivă, în care datele iniţiale să fie analizate sau transformate, după primirea semnalul folosind pagina web **DSReader**, aceasta se poate modifica, sau completa. În acest exemplu se va genera un semnal sinusoidal, care să fie afişat pe grafic alături de semnalul primit de la **DSWriter**. Parametrii semnalului generat vor putea fi modificaţi, folosind un „potenţiometru" inserat pe interfaţa grafică iar parametrii semnalului primit vor pute fi modificaţi din **DSWriter**. De asemenea, cele două semnale se vor însuma, rezultatul fiind afişat pe un alt grafic.

Modificarea elementului de control.

Se deschide, în Visual Basic, proiectul DSReader pentru a face modificările. În **figura 14.12** se prezintă varianta finală a componentei DSReader, conectată la server-ul de comunicaţie DataSocket, afişându-se apoi semnalul primit, semnalul generat şi rezultatul însumării lor.

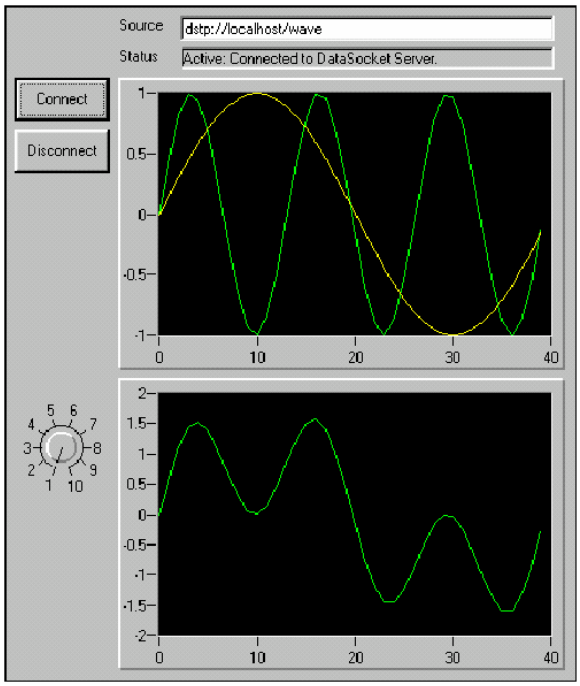

Fig. 14.12. Varianta finală a componentei DSReader conectată la DataSocket Server.

1. Se adăugă, pe graficul CWGraph1, încă un ecran folosit pentru afişarea unui semnal *(plot)*. Pentru aceasta, se apasă cu butonul din dreapta pe grafic şi se selectează Properties. În pagina Plots, se adăugă un semnal folosind butonul Add şi se schimbă culoarea pentru Plot-2, în galben.

2.  Se inserează, pe şablon, un element CWGraph şi se păstrează setările implicite.

3.  Se inserează, pe şablon, un element de control CWKnob. Pentru a modifica scala „potenţiometrului", se apasă cu butonul din dreapta pe el, selectând Properties, iar pe pagina Numeric se înlocuieşte Minimum cu 1.

Modificarea secvenţelor de cod.

1. Se adăugă următoarea funcţie, în fereastra cu codul. Această funcţie generează un semnal sinusoidal, folosind valoarea potenţiometrului, însumează semnalul generat cu cel primit de la DSWriter şi afişează rezultatul. Acesta va fi afişat pe cel de-al doilea grafic, iar semnalul generat pe primul.

```
Sub AddSinWaves()
Dim i
Dim ResultWave
Dim MySinWave(0 To 39)
ResultWave = CWDataSocket1.Data.Value
For i = 0 To 39
  MySinWave(i) = Sin(2*3.14*i*(CWKnob1.Value/40))
  ResultWave(i) = ResultWave(i) + MySinWave(i)
Next i
CWGraph1.Plots(2).PlotY MySinWave
CWGraph2.Plots(1).PlotY ResultWave
End Sub
```

2. Funcţia `AddSinWaves` trebuie apelată când se primesc date de la DSWriter. Pentru aceasta se adaugă linia `AddSinWaves` la procedura de tratare a evenimentului, creată mai înainte:

```
Private Sub CWDataSocket1_OnDataUpdated(ByVal Data As   CWDSLib.CWData)
' For extra error checking. Only plot the value if it's an array.
If IsArray(Data.Value) Then
CWGraph1.PlotY Data.Value
AddSinWaves
End If
End Sub
```

3. Dacă utilizatorul modifică valoarea potenţiometrului trebuie, de asemenea, apelată funcţia `AddSinWaves`. Pentru aceasta se apasă de două ori pe potenţiometru şi se adaugă următoarea procedură de tratare a evenimentului:

```
Private Sub CWKnob1_PointerValueChanged(ByVal Pointer As Long, Value As
Variant)
AddSinWaves
End Sub
```

În continuare se salvează proiectul, într-un fişier DSReader.ocx nou, se şterg fişierele HTML şi .cab şi se refac folosind Application Setup Wizard. Apoi se deschide pagina web **DSReader** în Internet Explorer, iar după stabilirea conexiunii se testează funcţionarea acesteia, modificând din potenţiometru parametrii undei generate şi a semnalului primit din **DSWriter**.

În acest exemplu se simulează funcţionarea aplicaţiilor, pe un singur terminal, dar se prezintă, în continuare, paşii de urmat pentru a permite vizualizarea paginii web de pe un calculator din reţea.

Accesarea paginii web DSReader folosind protocolul HTTP.

Pentru a vizualiza pagina web DSReader, se pun fişierele HTML şi arhiva .cab pe un server HTTP. Fişierul .cab conţine informaţii despre toate elementele incluse în componenta DSReader şi, pentru că există o referinţă pentru el în fişierul HTML (tag-ul `<CODEBASE>`), acesta va fi descărcat, expandat şi instalat automat de aplicaţia Internet Explorer.

Utilizând setările de securitate implicite, Internet Explorer descarcă numai elementele de control ActiveX cu semnătură digitală, care provin de la producătorii de aplicaţii software înregistraţi. Componenta DSReader, care a fost dezvoltată nu are semnătură digitală, de aceea trebuie modificate setările de securitate din Internet Explorer, astfel:

1. În aplicaţia Internet Explorer se selectează Tools » Internet Options;
2. În fereastra Security, în lista Zone se selectează Trusted site zones şi se stabileşte nivelul de securitate *low*. (Observaţie: acest nivel se referă la site-urile din lista *Trusted sites zones* şi nu modifică nivelul de securitate pentru celelalte site-uri);
3. Folosind butonul Add Sites se introduce numele site-ului**;**
4. Se deselectează opţiunea Require server verification (https://) for all sites in this zone.

Observaţii: La încărcarea paginii DSReader se va primi un mesaj de avertizare privind încărcarea unui control ActiveX, fără semnătură digitală. Dacă fişierele sunt puse pe un server Unix trebuie făcută o modificare a fişierului HTML. În linia următoare de cod HTML se înlocuieşte CAB scris cu majuscule, cu cab cu litere mici:

`CODEBASE="reader.CAB#version=1,0,0,0"`

În mod asemănător se poate dezvolta şi o componentă de tip DSWriter care să fie inclusă într-o pagină web DSWriter.

TESTE DE AUTOEVALUARE

I. Alegeţi răspunsul corect:

1. O pereche de aplicaţii de tip „Client/Server” operează la nivelul OSI ... şi comunică printr-un circuit virtual care operează la nivelul OSI...:

   a) 7-client, 6-server;
   b) 7-aplicaţie şi 3-reţea;
   c) doar 7-aplicaţie;
   d) 7-aplicaţie, 4-transport;
   e) 7-aplicaţie, 4-transport şi 3-reţea.

2. Instrumentele virtuale bazate pe TCP-Write/TCP-Read se pot folosi în următoarea combinaţie de reţele publice-private:
   a) Client în reţea publică, server în reţea privată;
   b) Client în reţea privată, server în reţea publică;
   c) Client în reţea publică, server în reţea publică;
   d) Client în reţea privată, server în aceeaşi reţea privată;
   e) Client în reţea privată, server în altă reţea privată.

3. Instrumentele virtuale bazate pe componentele DataSocket se pot utiliza în următoarele situaţii:
   a) Client DS în reţea privată, server DS în reţea publică;
   b) Client DS în reţea privată, server DS în aceeasi reţea privată;
   c) Client DS în reţea publică, server DS în reţea publică;
   d) Client DS în reţea publică, server DS în reţea privată;
   e) Client DS în reţea privată, server DS în altă reţea privată.

II. Alege□i răspunsul corect:

1. Două aplicaţii DSWrite şi DSRead se pot interconecta prin DSServer doar dacă acesta se află în reţeaua ăublică.

   □ adevărat □ fals

2. TCPWrite este funcţia care permite scrierea datelor printr-o conexiune TCP, doar dacă pe terminalul destinaţie operează componenta TCPRead:.

   □ adevărat □ fals

3. Instrumentul DSWrite, operând pe un terminal aflat într-o reţea privată, permite transferul datelor spre un instrument DSRead, care operează pe un terminal aflat în altă reţea privată, conectat la acelaşi server DS.

   □ adevărat □ fals

RĂSPUNSURI

I. Varianta de răspuns corectă:

1. d
2. b, c, d
3. a, b, c

II. Alege□i răspunsul corect:
1. adevărat
2. fals
3. adevărat

Bibliografie

1. **Bulăceanu, Cl.,** *Reţele locale de calculatoare*, Ed. Tehnică, Bucureşti, 1995
2. **Lowe, D.,** *Tehnologia client/server pentru toţi*, Ed. Teora, Bucureşti, 1996
3. **Ranch, D. A.**, „*Linux IP Masquerade HOWTO*”, v2.00.110903, Distribuţie Linux, 2003
4. * * * Manuale LabVIEW