Int. J. of Computers, Communications & Control, ISSN 1841-9836, E-ISSN 1841-9844
Vol. VI (2011), No. 2 (June), pp. 297-304

The Communication in Distributed Client - Server Systems Used for Management of Flexible Manufacturing Systems

V. Lupu, D.E. Tiliute

Valeriu Lupu, Doru E. Tiliute
"Stefan cel Mare" University of Suceava
Department of Informatics, Faculty of
Economics and Public Administration
E-mail: valeriul@seap.usv.ro, dtiliute@seap.usv.ro

Abstract: Labour productivity growth is a necessary condition for social and economic progress, in general, and to overcome the economic crisis facing most of the world, in special.
Applying innovative solutions, based on the ITC, is one of the straight ways for achieving that objective, both important and necessary. This paper presents a software solution applicable to industrial production based on numerically controlled machines. It involves a distributed client - server communication system, combined with MLP neural networks for the recognition of the 2D industrial objects, viewed from any angle. The information on prismatic and rotational parts to be processed by numerically controlled machine, are stored on a database server together with the corresponding processing programs. The client applications run on the numerically controlled machines and on the robots serving groups of machines. While the machines are fixed, the robots are mobile and can move from a machine to another. As a novelty of the proposed solution, in some well defined situations, the clients are allowed to change messages among them, in order to avoid the server overload. The neural networks are used to help robots to recognize the parts before and during manipulation.
Keywords: client, server, web, Java, image.

1 Introduction in domain

The flexible production system consists in a number of fixed numerically controlled machines which process the parts, several mobile robots [1] whose main task is to load/unload the machines and storing parts in technologic stocks. The production system is driven by a complex distributed informatics system containing two servers, an application server and a database server, and several clients. The application server provides those applications required by the coordination of the robots and controlling the processing [2] stage of each part on each machine. It also has the communication protocols with the clients and with database server. Two video cams equip each robot and machine in order to ensure a 3D vision of parts to be processed or handled. The position of each robot is monitored continuously by the application server using an appropriate number of video cams fixed on the production hall ceiling.

An overview of the communication system is shown in figure 1.

For each part, the following geometrical features are stored and available in database server: aria, perimeter, scale factor, minimum radius, maximum radius, average radius, standard radius deflection, minimum embedded rectangle dimensions, maximum inertia momentum, minimum inertia momentum, elongation factor, average of the gray levels, Pentland ratio, $R_{min}(x)/R_{max}(x)$ ratio, $A_{elipsa}(x)/B_{elipsa}(x)$ ratio, form characterization using momentum, etc.).

Functional features achieving an accurate enough description of the image objects using analytically developments easy to utilize. This category comprise Fourier coefficients of the inherent contour curve function, inertia momentum of different orders calculated on contour curve and the invariant computed using inertia momentum.

Topological features - conveying objects proprieties non depending on the distortions who aren't affecting the surface. These features are free from distances. We specify some of them:

C - the connected components number;

H - the cavities number;

E = C-H Euler's number.

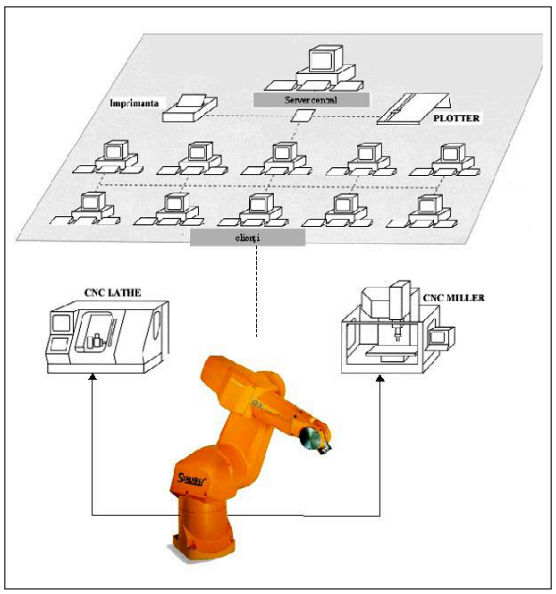

Figure 1: An example of distributed client-server system

Also, the relations among different areas are forming a good descriptor. These relations can be represented as a graph of connected components.

The features mentioned above are needed in order to recognize the parts about to be processed using numerically controlled machine. Images of the parts are captured by two video cameras placed on each robot in perpendicular planes, resulting in a 3D view [3]. For recognition it is used a neuronal architecture of multilayer perceptron type.

The solution presented in this paper employs neural networks for image processing and for the recognition of the model-based industrial objects [5]. In this order, a software application was created, giving to the robots the ability of recognizing the parts to be processed on the numerical-control machines . The software provides also a human interface that was used to verify correct functionality of neural network pattern recognition and communications, as described bellow.

The neural network [4] training is ensured by a software component that implements the well-known backpropagation algorithm. For the classification of the vectors bearing the characteristics of the shapes in the input space, the model of the multilayer perceptron was used. The vectors of the training set present themselves to the network with the class they belong to. In other words, the training inputs are organized as association lists in which the associated vectors correspond to a classification of the key vectors. Consequently, an element of the list is a pair key vector - corresponding class. If sufficient objects have been memorized from each class, an internal representation can be made of each class by means of the connection weights of the network. The model of the multilayer perceptron used in this software solves the problem of the shape

classification. The network inputs are the vectors that are to be classified, while its outputs are the corresponding classes.

2 The client-side program

The "client" program contains three classes: "client", "network" and "writer". Initially, the program imports three packages: "java.io.*" (used for input/output actions; io = Input / Output), "java.awt.*" (used for graphical objects; awt = Abstract Window Toolkit) and "sun.net.*" (used to establish a connection between client and server) [6], [7].

In order to run the client program, the user will type the following command:

java client $< IPoraddress >< ClientName >$;

where:

→ *java* - the name of the application that launch the classes;

→ *client* - the name of the class that will be executed;

→ *IP or address* - IP address or server name;

→ *ClientName* - the name of the client who wants to connect to the server.

The following sections explain the three classes of the client application.

The "client" class.

The "client" class extends the Frame class, which has a method called add. Using this method we can insert in the middle of the frame an object of type network.

The class contains two methods: main and client.

In the main method it is verified if the number of arguments is smaller than two. The arguments meaning is:

- args[0] - the IP or the server name;
- args[1] - the name of the client who wants to connect to the server.

This function is the first executed when the client class starts.

With this method, an instance of the client class is created, having as parameters the server IP address and the client's name.

The client method receives as parameters the name or the IP address of the server (String host) and the user name (String username). Inside the frame we insert a new object, which is actually a new instance of the network class, having as parameters the same values received by the function. Then, the frame size is set to (500,500). The method show() displays the frame on the main screen.

The "network" class.

This class extends the Panel class. The meaning of the variables is the following:

• public static boolean afis - if it is true, then the messages that are received from server are shown in the green box from the client's main window;

• NetworkClient network_client - using this method, we can declare a client. We use the specifications from the sun.net.* package;

• DataInputStream net_input - using this variable, the client can receive messages from server;

• PrintStream net_output - using this variable, the client can send messages to the server;

• static int modif=0 - if it is 1, then the image received from server is shown in the client's main window. This image is called "imagine.gif". If it is 0, then this image isn't shown in the window.

• String username - user name;

• boolean connected=false - if it is true, this means that the client is connected to the server;

• writer w - using this variable, we can create a new instance of the writer class, which handles with the read of characters or strings received from server;

• String typed_line="" - the message that's typed by client.

Bellow are the methods of this class:

The class constructor. The constructor receives two parameters, which represents the name or the IP address of the server and the user name. Each time a client tries to connect to the server, the connect method is called. If the connection succeeded, the connected will be set on true and also a new instance of the writer class will be created. If the connection doesn't succeed, an error message is returned to the client (it may also be viewed on the human interface client) and the execution of the application stops.

The **connect** method. This method tries to connect to the server on a specific port (we chose port No. 1111 but it can be any unassociated port). The connection is achieved by creating a new instance of the class NetworkClient, which can be found in the sun.net.* package. The arguments of this constructor are the name or IP address of the server and the port number. After establishing the connection the net_input and net_output variables are initialized with instances of the classes that correspond to their definition. The serverOutput variable is declared inside the NetworkClient class and it is initialized when the constructor of the NetworkClient class is called.

The **read_net_input** method. This method reads a line from server and returns the string received or null in case that an input/output error occurs. It also returns null if there is nothing available from the server.

The **write_net_output** method. This method writes a string to the server.

The **close_server** method. This method tries to close the communication between client and server.

The **keyDown** method. This method is executed when a key is pressed in the client's window (in human interface client version). Also, it calls the repaint method in order to make a quick repaint of the client's window.

The **paint** method. This method paints the client's frame. If writing messages from server is allowed, a welcome message is displayed: "Hello, <UserName> !", then two colored boxes are drawn, one red and the other one green, under the welcome message. Each box displays different messages; the red one displays the messages sent by client and in the green one are displayed the messages received from server. If the modif variable is set, then the image "imagine.gif" will be shown. This image is received from the server.

The "writer" class.

This class handles with the reading of data from server. This method extends the Thread class, which is specific for repeated actions. If the keyword "RECV" is received from server, the variable primesc becomes true. The string specifies that a data transaction between server and client is about to begin. The transmission consists in the content of a ".gif" image, whose size is specified by "SIZE nnnn" string, which precedes the actual content. Data are received until the "STOP" string is detected. In this case the modif variable from network class will become 1 and the repaint method from the same class will be called.

3 The server-side program

The server application contains two classes [6], [7].

The server class extends the NetworkServer class, which is a part of "sun.net" package. In this class we consider that the maximum number of users is 1,000. The meaning of the variables from the server class is given bellow:

String user[] - this array will contains the name of the users that are currently connected to server;

String sir[] - in this array will be stored the last message from each user;

DataInputStream net_input[] - by using this array, the server can receive data from clients;

PrintStream net_output[] - by using this array the server can send information and data to cliens;

reader r[] - in this vector will be stored the addresses of the "reader" class instances;

static int client_counter = 0 - the current number of connected clients.

The main method creates a new instance of the server class.

The constructor of this class initializes the arrays presented above.

When idle, the server console displays Waiting for users...

The most important method in this class is serviceRequest, which is a function of the NetworkServer class. When an event occurs at the server, this function is automatically called and creates an instance of the reader class. This thing happens each time when a client connects to the server.

The method read_net_input reads data from clients.

The method write_net_output writes data to clients.

Then the server is started, by calling the "startServer(1111)" instruction. This instruction starts the server, specifying that the port on which the server will receive message is 1111. This port number can be changed, but if is changed in the server class it also has to be changed in client class.

If the server was started successfully, a message is printed on the server console:

Waiting for users ...

If it's not possible to open the server on the port 1111, then an error message will be shown and the execution of this application will be canceled.

The most important method in this class is serviceRequest, which is a function of the NetworkServer class. When an event occurs at the server, this function is automatically called. In this function it is created an instance of the reader class. This thing happens each time when a client is connecting to the server.

The method read_net_input reads data from clients.

The method write_net_output writes data to clients.

The *reader* class

This class extends the Thread class. The most important method of this class is run. This method is called each time when an event occurs (connection of a client, the send of a message from a client, etc.). When the end of sentence is detected (point, !, ?), it checks if the message represents the standard syntax for sending data (images). This syntax is:

SEND $< ImageName > TO < user >$ where ImageName is the name of the image stored on the server and the user is the name of the user that has to receive the image. Examples:

SEND image1 TO machine1.

Send image2 TO machine2.

The syntax is "case-insensitive".

The application server has the possibility to search the image in database. Using the methods of the package "java.sql", queries may be sent to database and the results are fetched. Snapshots of server and clients with human interface are depicted in figure 2. Samples of pictures of parts to be processed are given in figure 3.

4 Case study - MLP type neural network for recognizing objects from any angle

The network has three layers. At the input, each neuron is connected to the outputs of all the neurons from the previous layer, its output being connected to the inputs of all the neurons of the following layer. Within the same layer, the neurons are not connected between them. The dimension of the input layer is given by the dimension of the five vectors describing each shape (sphericity, Pentland's ratio, radii ratio, ratio between the ellipses on the two axes, shape characterisation by means of moments). The dimension of the output layer is equal to the number of classes into which the input data will be separated, that is four (corresponding to the classes: disk, square, triangle, cross).

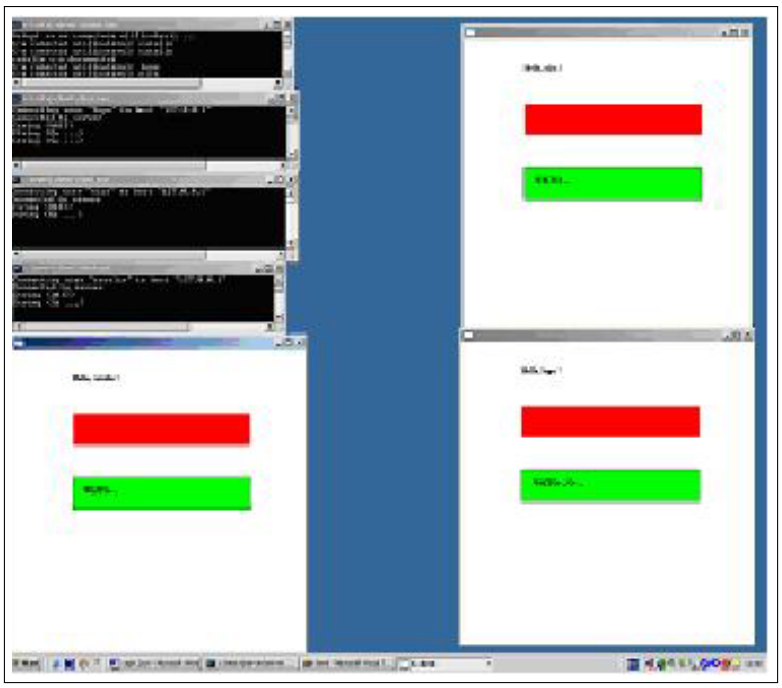

Figure 2: a) windows screen

The activation function used for the output of the neurons is:

$$f(a) = 1/(1 + exp(-a)).$$

As a result, the outputs of the network (y) will be in the domain $[0, 1]^4$. The representative of the class where the data set presented at the input is placed will have the output 1, while the others, 0.

For other types of pieces, the theoretical values cannot be determined directly, requiring measurements and experiments specific to each class.

The values obtained, following the implementation of the parameters calculation are given in table 2 (through medium values):

Table 2 Medium Values of the Shapes' Parameters

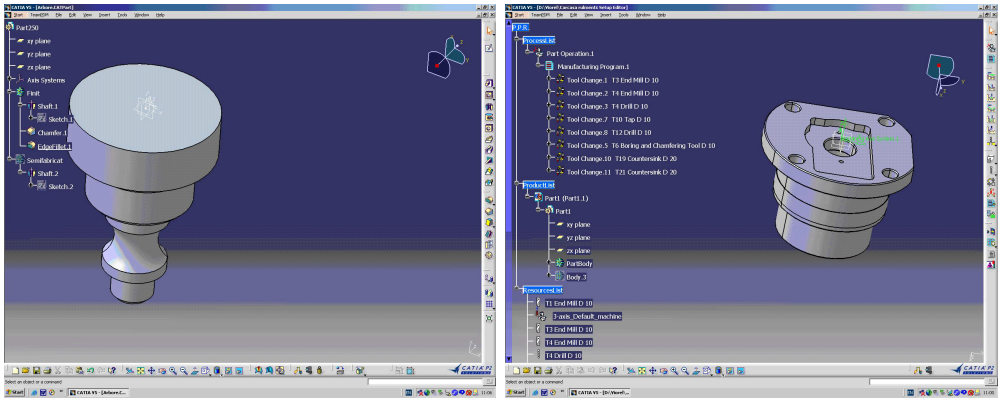

Figure 3: b) picture 1 c) picture 2

	C(X)	P(X)	R_{min}(X)/ R_{max}(X)	$A_{eclipse}$(X)/ $B_{eclipse}$(X)	F(X)
	1.014	1.120	0.921	1.042	0.040
	0.841	0.531	0.673	2.512	0.125
	0.691	0.512	0.481	3.824	0.230
	0.265	0.401	0.213	14.412	0.612

Where: **1. sphericity** C(X) = 4 π A(X) / $(P(X))^2$.
2. Pentland's ratio P(X) = 4 A(X) / $(\pi FMD(X))^2$.
3. radii ratio R_{max}(X)=maxdi / i=1..N;
$R_{min}(X)$=min di / i=1..N.
4. ratio between the ellipses on the two axes.
A_{elipsa}(X) = ˝ (P(X) / +$((P(X)/)^2$ - 4 $A(X)/)^{1/2}$);
B_{elipsa}(X) = ˝ (P(X) / - $((P(X)/)^2$ - 4 $A(X)/)^{1/2}$).

5. shape characterisation by means of moments

$$F(X) = \frac{\sqrt{1/N \sum_1^N (d_i - M_1)^2}}{1/N \sum_1^N d_i}$$

Structure of the file data.txt
N patterns - number of input vectors for the network training
$X_1, X_2, X_3, X_4, \ldots, X_{np} Y_i$ (i=1, nc, where nc = number of classes) the components of the input vector (shapes' parameters), where np = number of shapes' parameters.

The parameters were introduced as follows (according to the table corresponding to each shape):

class			
1-square	0.867 0.660 2.145 0.124	0.583 0.583 2.302 0.164	0.738 0.738 1.968 0.102
2-disk	1.018 0.867 1.000 1.000	1.021 0.857 1.000 0.039	0.994 0.867 1.168 0.051
3-cross	0.160 0.063 22.924 0.732	0.292 0.250 11.615 0.462	0.169 0.143 21.596 0.647
4-triangle	0.670 0.462 3.696 0.229	0.712 0.423 3.313 0.231	0.708 0.514 3.347 0.211

5 Conclusions

The functionality of the proposed system was proved by a dedicated application software that responds to the following main requirements:

- provides the recognition of the 2D and 3D industrial objects viewed from any angle;
- provides border touching, partial overlapping and clipping of the objects in the image;
- it is noise tolerant.

The comparative study of the existent solutions, in correlation with the above-mentioned requirements, leads to the conclusion that 3D object-centered models are suitable in order to ensure the deduction of the model views from any angle.

Bibliography

[1] C.V. Kifor, C. Oprean, D.D.M. Banciu - Intelligent Systems for Assisting Decisions in Advanced Product and Process Planning and Design, *Studies in Informatics and Control*, SIC Vol.18, No.3, 2009

[2] Mircea Ivanescu, Mihaela Cecilia Florescu, Nirvana Popescu - The control of the Hyper - redundant Manipulators by Frequency Criteria, *Studies in Informatics and Control*, SIC Vol.18, No.3, 2009

[3] Lupu Valeriu - *Contributions to the management of the flexible manufacturing systems* (PhD Thesis), 2004

[4] Mohammad Reza Soltanpour, Seyed Ehsan Shafie - Design and Stability Analysis of a Robust Impendance Control System for a Robot Manipulator, *Studies in Informatics and Control*, SIC Vol.19, No.1, 2010

[5] O.Khatib, K.Yokoi, K.Chang, D.Ruspini, R.Holmberg, A.Casal - Coordination and Decentralized Cooperation of Multiple Mobile Manipulators, *Journal of Robotic Systems*, Volume 13, No.11, 1996, ISSN: 0741 - 2223

[6] D.Logofatu - *Algoritmi fundamentali in Java. Aplicatii*, ISBN: 978 - 973 - 46 - 0815 - 7, Editura Polirom, 2007

[7] Java on-line documentation (http://java.sun.com)