Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 49(63), Fascicola 2, 2004

# IeL Com – an integrated module for communication in E-learning

B.Orza[1], M. Gîvan[1], A. Vlad[1], A. Olah[1], A.Vlaicu[1]

**Abstract – IeL Com is a module of the integrated environment for educational activities management IeL. This module, developed by us, permits students, professors, tutors and administrators to communicate each other using a chat-whiteboard application. The application permits a group of persons to draw together, on a shared table, using the client application that runs on a computer connected in a network. Application puts at command users a set of graphic and textual objects that could be used to realize graphic objects. Participants can communicate through text box (chat) using a dedicated zone in the window.**

**Users can be students and tutors involved in the virtual university managed by IeL environment. There is necessary the authentication of the users, this will have access only to those chat rooms in which he is involved. He can also have more than one chat or whiteboard open window, one for each user involved in the session.**

**IeL Com is a client-server application developed using the C# technology. Due the utilization of the technology specialized for the development distributed applications (.Net Remoting), that permits appeal of the object methods located on server as these objects find out on local client, we could use OOP (Object Oriented Programming) concepts and architectures like polymorphism, listener concept or view-control architecture.**

**Keywords: e-learning, remoting, whiteboard, .net**

## I. INTRODUCTION

The application will allow a group of people to draw together, on a joint whiteboard, each of the members of the group being in front of a pc connected to the network. The application allows the user to draw using different graphic objects and different ways of manipulating them. The participants can communicate by writing text on a special design part of the screen.

## II. TEHNOLOGY

.NET Remoting is for web services, what ASP was to CGI programming. .Net gives us a large array of tools and facilities, e.g. allows the work with objects that keep their state after being called. It also allows different transfer mechanisms (HTTP and TCP), coding mechanisms (SOAP) and security mechanisms (IIS, SSL).

One of the major advantages of this technology is the easy way in witch one can create distributed applications. There are no intermediary steps in the case of compiling the proxy/stub like in the case of Java RMI. One doesn't have to define interfaces in special programming languages like in the case of CORBA and DCOM. By changing only a word in the configuration file we have the possibility to select the coding format used starting from the binary format to the SOAP format.

### A. EXTENDED ARCHITECTURE

The technology offers to the developers and administrators a great variety of protocols and formats. Every time a client application gets a reference to an object on the server, the object will be represented through a proxy, thus "masking" the destination object. The methods of the object will be called through the proxy. Every time a call will reach the proxy, the call will be converted into a message, and the message will pass through numerous layers. The message will be passed to the serialization layer that will convert it into a special format (SOAP, binary). The serialized message will then reach the transport channel, where it will be transferred to the server through a protocol like HTTP or TCP. On the server side, the message cross inversely the formatting layer, the serialized message being brought to the original form and then sent to the dispatcher. Finally the dispatcher calls the method of the object and sends back the answer through the same layers.

By changing only a few parameters in the configuration file, we can change through different

[1] Technical University of Cluj Napoca, 26-28 G. Baritiu street, Cluj Napoca, 0264-401309, fax. 0264-591689, Bogdan.Orza@com.utcluj.ro

types of layer implementations without writing any code writing. This way an application that uses TCP can be very easily modified so that it will use HTTP as a transport channel, thus having a better scalability.

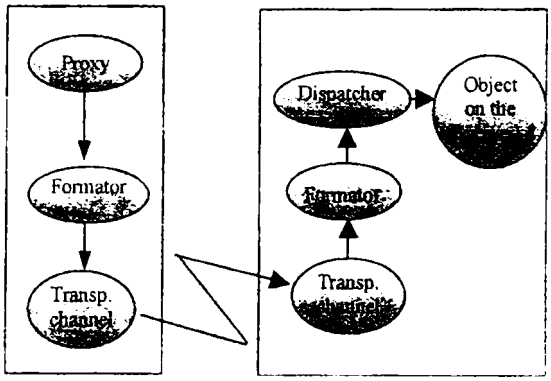


Fig. 1. Simplified architecture of .NET Remoting

## B. INTERFACE DEFINITION

Many distributed systems like DCE/RPC, RMI and J2EE, need to manually create the so called proxy/stub objects. The proxy encapsulates the connection to the remote object and sends calls to the object on the server. In many of those systems (CORBA, DCE/RPC and DCOM) "the source code" that generates these objects must be written in IDL (Interface Definition Language) and precompiled in order to generate the header files for some programming languages.

In contrast with this traditional approach, .Net Remoting uses a generic proxy for all this kind of objects. This is possible because .Net was conceived from the beginning as a distributed applications platform, this facilities being added lately to the other technologies.

## C. DATA SERIALIZATION

All the frameworks used for distributed applications support the automatic coding of objects in any of the following formats binary, XML or SOAP. The problem arises when we want to transfer a copy from the server to the client, but COM+ doesn't offer this facility as Java RMI and EJB do. In this case we use ActiveX objects for the transfer, but their use means sending a big amount of data through the network.

In .Net it is sufficient to mark the object with the Serializable attribute or to implement the Iserializable interface and the platform will take care of the rest. We also may transfer data via XML.

## D. OBJECT LIFE SPAN MANAGEMENT

There are three ways to control the life span of objects in distributed applications. The first one consists of a connection (e.g. a TCP connection between the client and the server). When this connection is closed, the object/objects on the server will be destroyed. DCOM uses another method; it combines the pinging mechanism and reference counter. In this case the server receives the messages from the client at predefined intervals of time. When it stops receiving messages, the server will release the resources.

In the Internet era, we still don't know too much of the clients at the other end of the line, we cannot relay on the possibility to create a direct TCP connection between the clients and server. The user may be behind a firewall which allows only HTTP traffic to pass through. The same router can block pings sent by the server to the user. Taking into account all this, the .Net Remoting object life span management can be customized for every application. First to an object will be assigned a certain life span, and at every client call that life span will be increased. Also a so called sponsor register to the object on the server may exist. The sponsor is contacted before the life span expires, and if exists the object life span will be increased.

# III. APPLICATION CLASSES

Iel Com is a client server application. We will further describe the server side architecture and the architecture of the client-server/server-client communication module.

Because of the use of a specialized technology for calling the remote objects (.Net Remoting), we wore able to use OOP specific concepts and architectures like polymorphism, listeners, or model view controller architecture.

The next figure shows the simplified architecture of the classes used for this application as the way they communicate with the IeL platform.

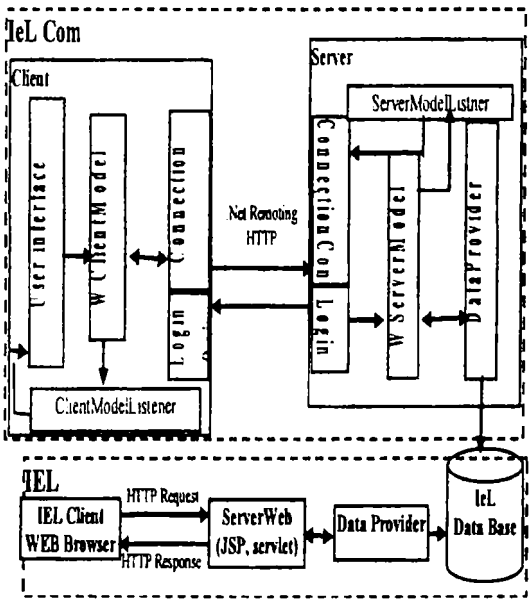


Fig. 2. Simplified architecture of the application

Classes WClientModel and WServerModel correspond to the models from Model View Controller, WConnection to the controller, WhiteboardGui and ChatGui to the interface.

## IV. COMMUNICATION MECHANISM BETWEEN THE CLIENT AND THE SERVER

### A. COMMUNICATION PROTOCOL

In the first phase the client sends to the server an authentication request. The server takes the request and interrogates the database. If the name and password are correct, the server will return to the client a connection object that contains some information for the user (the user type, the name, the database id etc.). If the authentication is not made the server will return NULL, this is interpreted by the client interface by displaying an error message. If the authentication is made the client can communicate with the server by sending and receiving WObject objects or derivate objects form WObject. The communication is made between users subscribed to a certain course, they send objects to the server, and the server sends those objects to the rest of the users on the same course. At the server side and client side the objects are stockated, thus allowing operations like save and undo, if this will be done only at the server side the traffic will be much higher if one would need to save or undo.

For instance if we want to obtain information regarding the number of students on every course, the number of online users at a certain moment of time, the courses that are running now, the client calls the getUserInfs (WIntMess mess) method of the WConnection object. By doing this the user does not interrogates directly the database, but communicates with it through objects on the server, thus increasing the application level of security.

We use WIntMess objects to manage the application, although the objects are derived from WObject (to assure transparency at the transport layer) they will be not stored in the lists of the corresponding course on the server and the client, but they will be used only to transmit data regarding different events that are occurring, like: a user enters or exits, someone logs on another machine, someone makes an undo, etc.). In all this cases the client or the server will create such objects, set their desired message attribute and they will send them in the network to let know the users of the events described earlier.

At the client side, the objects received from the server will be sent to the graphic interfaces of the chat and whiteboard in order to be displayed. For this we have used the listener pattern, which means that we add to a list all the WClientModeListner objects corresponding to the graphical interface and then we run through the list every time we receive a new object that will be finally sent to the graphical interface.

### B. CLASSES AND OBJECTS USED FOR COMMUNICATION

The communication between the client and the server is made using the WObject objects. From this generic type we have derived all the other objects which are used for drawing or text display.

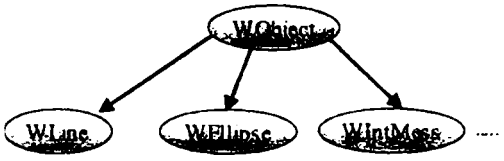


Fig. 3. Inheritance relationship between the objects used for drawing

So at the client we will display drawings in graphical format not as bitmaps. The advantage of this representation is that we keep the semantic content of the drawings, and thus one can very easily modify or change the drawings. The use of this solution is based on the concept of polymorphism, so in the network we transmit only on object type, WObject, which can be instances of other objects derived from WObject (WEllipse, WLine, WIntMess, etc.) that will be recognized and treated as needed at the moment they reach the client graphical interface.

To transfer objects through the network we can use the following: value transfer or reference transfer. The value object transfer supposes the serialization of the objects, including also the referential objects from the class, in a persistent form from which they will be reconstructed at reception. An object can be serialized if it's marked with the [Serializable] attribute or if it implements the ISerializable interface. After the serialization we will have an XML document that will be sent to the server which will interpret and remake the original object.

It is important to outline that transfer by value does not imply the existence of remote objects. All the object methods will be locally executed in the same context as calling one. This means that the compiled classes need to be available also at the client side. Although objects, that are derived from the MarshalByRefObjects, will not imply that.

When an object that needs to be transferred by value has a reference to another object, this last one has to be derivate from MarchalByRefObjects or need to be marked with the [Serializable] attribute.

The other types of objects are the ones that run on the server and allow the client to call their methods. It is

mandatory that these objects inherit the MarshalByRefObjects class. Instead of transferring a value that points to such an object, in the network it will be transferred only one type of objects: ObjRef, objects contain the name of the server/ip and an identifier, indicating uniquely an object on the server.
In the case of IelCom we use both transfer types. The graphical objects and text are serialized in the XML format and they are sent from the client to the server where they are stored on the corresponding server list and then depending on the destination they have they are sent to the appropriate client. Every object has two addresses. The first one is the user name of the user (this name is unique in the data base) and the second one is a combination between identifiers representing the address of the client and the address of the graphical interface to which the object is sent. All the other identifiers are attributes of WObject class, in such a way that all the derived classes will inherit them. WLogin Class makes the authentication and creates the connection object (WConnection) for each client. This class inherits the MarshalByRefObjects, so the transfer is made through the interface. Such a method is WConnection doLogin (String user, String pass, String type) which checks if the user is in the database and if the answer is affirmative it will create the connection object.

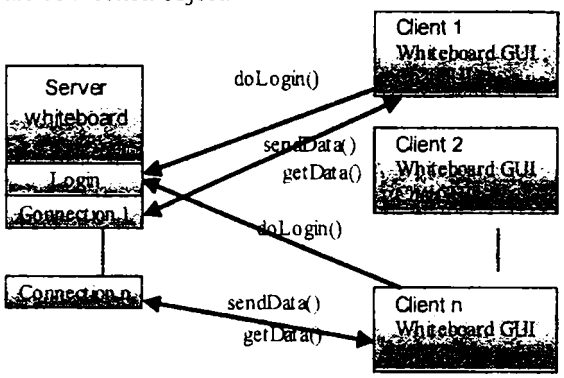


Fig. 4. The logging mechanism

So all the clients will communicate with the server through their communication object, thus the application is more secure because the client does not have direct access to any data on the server, this makes possible the implementation of a more complex security system that doesn't exist at this at this point in our application. The connections are object uniquely identified by two attributes: the client user name and a number generated by the server.
The connections are instances of WConnection class having two important methods: sendData (WObject obj) that allows transfer of objects from the client to the server, and addNewObj (WObject obj) through which a client receives objects from the server.

C. CLIENT AND SERVER SIDE OBJECT STORING

The object storing is made on the server side but also on the client side. We chose this method because we wanted to keep a low traffic when we want to save the drawings. The storing is made in lists of ArrayList type, where we can store WObject objects and WObject objects. The user subscribed at the virtual university can open a chat and/or a white board session for every course that is running, and he can communicate with any other user through a private chat.

For each of these communication methods there is a list where the text and objects of the drawings are stored. The server keeps the corresponding models of every opened course with active users; the client keeps only the ones where the user is active at that moment of time.

To distribute the objects from the server to the destination clients it was implemented an algorithm based in the listener concept. This concept presume that once a connection is made, it is added on a list on the server (WServerModel) and at the moment an object appears on the server, the list will be read and it will be sent to the appropriate connections.
From here through a thread the objects will be taken by an object corresponding to the WClientModel client model and the temporary list will be emptied. The following code sample is an example of the way objects are distributed on connections through the function wakeUpListners (WObject obj).

The listener concept is also used in the case of graphical interfaces (whiteboard and chat) which are registered as listeners to WClientModel. So the objects that come will be redirected depending on the destination application (the text for chat and the graphical objects for the whiteboard).

The WClientModel class as the WServerModel class implements the Singleton pattern, in such a way that there will be only one instance of every class on the whole application. To obtain the private class constructor is declared, such that we cannot instantiate the class outside. First is declared a static attribute of the class type, the value of this attribute will be set initially to null. It will be declared also a static method that will initiate the attribute just once. The uniqueness of the object corresponding to the models is very important because it is necessary to be able to obtain references to them from different points of the application, more then this they offer flexibility because we can add new modules without doing important changes of the application.

So any graphical interface, or any other module that needs the user identification data, can access them through the reference to the WClientModel provided by the method getInstance().

For a better management of the application we have defined the WIntMess class. Through objects of that class the messages are sent and received from the server. This objects are not stored on the server, they are used just for the management of the application. There are two attributes of the class: query and response. When the client wants to obtain some information from the server, for instance the number of student in a course, he will not point directly the data base because that can cause security problems. The solution is the creation of a WIntMess object with the query attribute set with the proper message, which is then sent to the server. The server will take the object and analyze the request and after that it will set the response attribute of the object with the object that holds the information desired by the client. The object will be then sent on the connection that came from. This type of object is also use to signal if a client connects on another machine, if he left the application, or if he wants to create a new drawing.

An interesting advantage of the application is that of undo. Although for stand-alone application this is quite a simple thing to implement, in the case of distributed applications this arise some problems. The first problem appears when we want to establish the way we want to make the undo. There are at least to possibilities: the first one is that the user is able to make undo only to the objects that he created, but this contrast the principle of shared whiteboard, because the users must be able to modify also the work of others. The second possibility is that the user can make undo on all the objects on the drawing, this is also what we choose for our implementation. So it was created a WIntMess object that sends the undo message to the server every time a user hits the Undo button from the graphical interface. This message reach the server, the server will update the model using the updateModels (WIntMess mess) method, which will eliminate the last added object. The message is also sent on the connections corresponding to the online users through the wakeUpListeners (WObject obj).

## V. CONCLUSIONS

Taking into account the continuous growing of the use of computers in the academic environment, such an application (chat and whiteboard) is a very useful tool that can be successfully used for distance education. At the moment the application doesn't need a large bandwidth for transferring the information from the client to the server, so it can be used even with poor internet connections (e.g. dial-up).

IeL Com is part of the integrated environment for distance education IeL, being a synchronous communication solution for students and teachers, administrators and tutors, creating a virtual space where the teachers or the tutors can teach their courses and the students can ask questions and receive their answers in real time.

The application was developed using .NetRemoting technology, because it is a good compromise between the bandwidth needed to communicate and the ease of implementation. Although it requires a larger bandwidth than the use of sockets, .NetRemoting provides the programmer an advanced implementation environment, which abstracts the transport layer from the OSI model, allowing the transmission of objects through the network and the calling of remote methods. In the case of .NetRemoting as in the case of Java RMI, due to actual security demands, the application configuring process is hard enough, because often there are added new security levels from one version of the frame work to the other, thus the need of adding new information in the configuration files.

The use of these technologies allows an easier implementation of the object oriented programming concepts (polymorphism, inheritance, etc.), it adds scalability plus to applications so that one can add more easily new modules and facilities. This is also the case of IeL Com, the developer can add new graphical objects, deriving the appropriate classes belonging to WObject, without worrying about the transmission through the network.

## Bibliography


[1] I. Rammer, *"Advanced .Net Remoting (C# edition)"*, APress 2002.

[2] A. Turtschi, *"C# .Net"*, Syngress Publishing, Inc.

[3] A. Vlaicu, V. Dobrotă, S. Iacob, "Tehnologii multimedia, sisteme, rețele și aplicații ", UTCN.

[4] B. Orza, M. Givan, S. Cristea, A. Vlaicu, *"IEL 2 - an Integrated Solution for Management, Evaluation and Communication in E-Learning"* , International Conference Advanced tools for E-learning in the Environmental Education, 12-13 February, Napoli, Italy

[5] B. Orza, M. Givan, S. Cristea, A. Vlaicu, *"Integrated solution for management, evaluation and communication in distance education systems "*, Optimization Of Electrical And Electronic Equipment Optim 04, May 20-22, 2004, Brasov, Romania