INTER-PROCESS COMMUNICATION WITH .NET REMOTING

Lecturer Rocsana BUCEA-MANEA-ŢONIŞ
„Spiru Haret" University

Lecturer Radu BUCEA-MANEA-ŢONIŞ
„Spiru Haret" University

REZUMAT. Lucrarea se concentrează asupra diferitelor tehnologii care asigură comunicarea în interiorul programelor pentru calculator, precum şi între calculatoare. Se referă la paradigma UNIX asupra modulelor slab-cuplate, foarte importantă chiar şi astăzi, în special în aplicaţii web. Se porneşte de la elementele de bază privind comunicarea pipe-line, stiva de protocoale web, procedura de apeluri la distanţă, şi mesajele orientate-obiect. Apoi se analizează facilităţile oferite de .Net Remoting în paralel cu serviciile web moderne. În final este prezentat un studiu de caz axat pe comunicarea inter-proces în domeniul de cercetare principal al autorilor, şi anume sistemele distribuite de luare a deciziilor.

Cuvinte cheie: IPC, UPI, RPC, CORBA, SOAP.

ABSTRACT. This paper focuses on various technologies that ensure communication inside computer programs and between computers. It focuses on UNIX paradigm on loosely-coupled modules, very influential even today, especially on web applications. It starts from the basics with the pipe-line communication, web stack of protocols, remote procedure calls, and object-oriented messaging. Then it explores the facilities of .Net Remoting in parallel with the web services upon the modern web seemed firmly based. It ends with a case study focused on inter-process communication from the main research field of the authors, distributed decision making systems.

Key words: IPC, UPI, RPC, CORBA, SOAP.

1. BRIEF HISTORY OF INTER-PROCESS COMMUNICATION (IPC) SYSTEMS

Communication between applications is actually communication between processes executed in the same memory space or in different memory spaces.

For the first case, it is widely known UNIX - pipeline communication implementation. This consists of using descriptors of mounted files as parameters for the read/write operations. PIPE() method by [Stevens, 2005] takes one descriptor like parameter for each operation, the result of writing descriptor serving as input for reading descriptor. These operations will be initiated by different processes, each with a unique identifier (PID).

In the case of processes running in different memory spaces, it is recommended to use socket interface. The machines involved in communication will play an alternative client/server role, and specified ports will have to remain open. The necessary steps to establish a communication via socket after [Buraga, 2001] are:

1) creating the socket device - SOCKET();

2) link the device with the network address of the machine - BIND();

3) listening and accepting connection requests from clients - LISTEN() and ACCEPT();

4) establishing a connection - CONNECT().

A computer network can be viewed as a graph of protocols that manages the messages and participants'addresses. After [Peterson, 2001], the supported classes by the networking kernel for UNIX systems are protocols and sessions. A protocol is the point at which IP or TCP datagram headers are attached or removed from messages. After the same author, a session is an object that exports a couple of operations - one for sending and another to receive messages - representing the end of a communication channel that implements the code necessary to interpret messages.

The uniform protocol interface (UPI) defines the following methods:

- xOPEN (where x stands for IP or TCP) – returns a session object,

- xPUSH() / xPOP()– attached/retrieve the header for the corresponding protocol,

- xCLOSE() – closes the session object and the channel of communication as well.

Common Object Request Broker Architecture (CORBA) uses the standard protocol IIOP to communicate using objects. For each object type, it must be defined an interface in OMG IDL. The interface is the syntactic part of the method that server instance exposes to customers in order to be invoked. Interface definition is then used to encapsulate the result of the request. Each

instance has its own unique reference to an object, for unequivocal identification matters. Clients use object references for their requests, identifying for Object Request Broker (ORB) exactly the instance they want to invoke.

Communication through Remote Procedure Call (RPC) is based on executing a program in the memory of a distant computer using a stored procedure. In addition to efficiency achieved by the developer, the portability is considerably enhanced by using the XDR language (XML-Data Reduced) for converting data from both server and client. This is achieved by two components with the role of proxy stub automatically generated with RPCGEN utility from Sun. It takes one Portmaper resident application on the server so that client request to be solved.

2. IPC WITH .NET REMOTING

Communication between applications is achieved by a process of services' integration at the functional level. At this level, integration can be achieved in three ways [Lungu, 2007]:

✓ By distributed objects - extends the object oriented calculus; distributed applications objects interact as if they were items of the same application;

✓ By message-oriented middleware - the systems are connected through asynchronous message queues;

✓ By services - applications communicate by using Web services, sharing common methods through the .Net platform.

Web Services are a customization of .NET Remoting, but they have a more simplified programming model and are dedicated for a wider audience. Web services allow applications the back-and-forth transmission of messages independently of platform, object model and programming language. But Web services do not know anything about the client making the request. Clients communicate by transferring messages to a specific format known as Simple Object Access Protocol (SOAP).

.NET Remoting allows the integration of applications on the same computer, on computers in the same network or on computers in different networks.

The differences between NET Remoting and Web Services are presented below [Lambert, 2010]:

✓ ASP.NET Web Services can be accessed via HTTP, while. NET Remoting can be accessed by any protocol.

✓ Web Services creates one new object for each request. .NET Remoting can correlate multiple calls from the same client and facilitates communication in both directions.

✓ Web services streamlines objects contained in SOAP messages through XML and can therefore only work with items that can be fully expressed in XML. .NET Remoting relies on the existence of the CRL assembly that contains information about data types.

✓ Web services enable interoperability between platforms for the heterogeneous environments, too. .NET Remoting can be built using. NET, or any other platform that supports a homogeneous medium.

In .Net Remoting remote objects are accessed by two types of channels: HttpChannel and TcpChannel, using the HTTP and TCP protocols.

Fig. 1. The class diagram associated with the dynamic library RemotableObject .

3. CASE STUDY

We developed an inter-process communication module using a dynamic library (see Fig.1) [www1], accessed by the decision module, in client mode, for interface module (see Fig.2), in server mode. Our Peer-to-Peer application makes use of the .NET Remoting technology.

In order to access the software library on both client and server side, using TCP for machines that are running in the same namespace, we have to follow next steps:

Step 1: `TcpChannel chan = new TcpChannel();` opens a TCP channel through which the stream of bytes is transmitted between client and server; the server listens to requests from the remote object

Step 2: `ChannelServices.RegisterChannel(chan);` the customer is required to register the communication channel before accessing the server

Step 3: `remoteObject = (MyRemotableObject) Activator.GetObject (typeof(MyRemotableObject), "tcp://localhost:8080/Object");` creates an instance of the object accessed remotely.

It is good to notice that the service must be connected to an available port, for example 8080, which is free on the working computer. To see a list of used ports on the computer we have to open a command window and run the command 'netstat - o'. Any object can be accessed remotely by the process of serialization and/or publication (Marshaling), that expose the object attributes to the customer. Object serialization can be binary or in XML format. When transmitting the object by value, serialization is the first step and delivery is the next one. When transmitting the object by reference only the address is delivered.

Step 4: **`remoteObject.SetMessage(lines);`** the SetMessage method is invoked by the client and allows the server to access the message

Step 5: **`RemotingConfiguration.RegisterWellKnownServiceType (typeof(MyRemotableObject),"Object",WellKnownObjectMode.Singleton);`** the object type that is the subject of communications is recorded on the server; the property *SingleCall* indicate that the object manages a single message from the client.

The service can be registered as *WellKnown Object Mode.SingleCall*, leading to a new instance of the object for each client, or *WellKnownObject Mode.Singleton*, leading to an instance of the object used for all customers.

Step 6: After successfully creating and compiling the project, it can be tested. After opening the interface, the server is ready to listen. The client has to make the request. The client execution should lead to display a decision tree in the interface. The client window will be closed and the server remains opened and available. It can be tested on another computer, too, but we have to change the localhost reference in the URL to indicate server location.

Step 7: **`RemotableObjects.Cache.Attach(this);`** the object inherits the address of the server active instance, and the *Notify* method of the *Observer* interface takes the string from that address and copy it into the multi-line textbox in the interface.

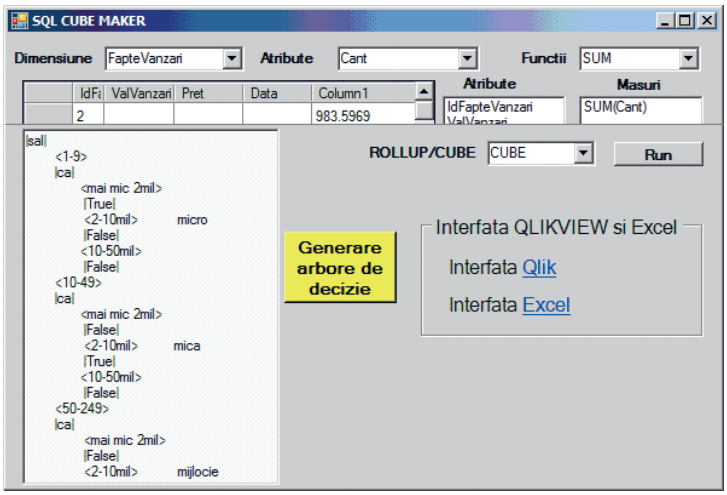

Fig. 2. SQL Cube Maker Interface.

Application modules behave alternatively as client and server by means of transmission (marshalling) asynchronous messages in both directions. Each module is linked to the Remoting Objects library and implements the *Observer* interface for the notification (*Notify* method) of the next module.

4. CONCLUSIONS

Applications must be developed based on interconnected modules, each module acting on the principle of UNIX filtering programs (pipe-line communication), where each command can accept and pass forward parameters, otherwise signaling an error. Inheriting the speed and robustness of non-proprietary systems, modern applications should focus on inter-process communication based on .Net Remoting technology.

BIBLIOGRAPHY

[Buraga, 2001] **Buraga S., Ciobanu G.**, *Atelier de programare în rețele de calculatoare*, Ed. Polirom, București, 2001, ISBN: 973-683-755-6;

[Lambert, 2010] **Lambert M. Surhone, Miriam T. Timpledon, Susan F Marseken,** *Net Remoting*, Verlag Dr. Mueller AG & Co. Kg, 2010, ISBN: 978-613-221-161-3

[Lungu, 2007] **Lungu I., Bologa A., Diaconiță V., Bâra A., Botha I.**, *Integrarea Sistemelor Informatice*, Ed. A.S.E., București, 2007, ISBN: 973-594-975-4;

[Peterson, 2001] **Peterson L., Davie B.,** *Rețele de calculatoare - o abordare sistemică*, Ed. All, București, 2001, ISBN: 973-684-389-0;

[Stevens, 2005] **Stevens W.R., Rago S.A.,** *Advanced Programming in the Unix Environment*, Second Edition, Addison-Wesley, 2005, ISBN: 0-201-43307-9;

[www1] http://www.codeproject.com/KB/IP/Net_Remoting.aspx