An Overview of Distributed Programming Techniques

M. Golub, D. Jakobović

Department of Electronics, Microelectronics, Computer and Intelligent Systems
Faculty of Electrical Engineering and Computing, University of Zagreb
Unska 3, HR-10000 Zagreb, Croatia
Phone: (+385-1)6129 967, E-mail: marin.golub@fer.hr, domagoj.jakobovic@fer.hr

I. Janeš

Technical support department
HEP - Transmission Ltd.
Ulica grada Vukovara 37, Zagreb, Croatia
Tel.: +385 1 6322 019 Fax: +385 1 6171 179, E-mail: ivan.janes@hep.hr

Abstract – In this paper we investigate the utilization of several parallel programming paradigms for use in a distributed programming environment. The implementations presented here are Remote Procedure Call mechanism (RPC), Message Passing Interface (MPI), Common object request broker architecture (CORBA), Java Remote Method Invocation (JAVA RMI), Distributed Component Object Model (DCOM) and .NET Remoting. A distributed application is implemented using each of the mentioned methods and their efficiency is compared. We address the issues of stability, portability and the amount of work needed for implementation. Particular attention is paid to security issues involved in distributed computing environment and the ability of presented methods to support the development of secure applications.

I. INTRODUCTION

Distributed computing has widened the object oriented and component paradigms. Now, it is possible for objects and components to exist on physically separated computers or platforms and communicate with each other through heterogeneous networks. The most important paradigms, that have marked the distributed computing era, are Open Network Computing Remote Procedure Call (ONC RPC), Message Passing Interface (MPI), Common Object Request Broker Architecture (CORBA), Java/Remote Method Invocation (Java/RMI), Distributed Component Object Model (DCOM) and successor to DCOM, .NET Remoting.

Distributed programming paradigms can roughly be divided into three models: remote procedure calls (ONC RPC), message passing model (MPI) and distributed objects (CORBA, Java/RMI, DCOM, .NET Remoting), with each model suitable for its own domain.

Distributed application is such that broadens its area of execution to more that a single computer. Goal of the distributed application architecture is mainly the improvement of performance and scalability. Ideally, distributed application can be broadened to service thousands of simultaneous clients by simply adding new computers.

Furthermore, there are other reasons for utilizing distributed architecture, such as:

- Code integration that is executing in different environments, on different operating systems and platforms.
- Providing synchronization and real-time communication between numerous clients (e.g. chat server). The implementation of such design as a traditional server would involve tremendous amount of database usage and frequent polling which would deny the possibility of serving a great number of users.
- Supporting thin clients (e.g. software on embedded devices) that do not posses enough processing power to accommodate for their data requirements.

This paper starts with a brief description of the evaluated distributed programming techniques in Section II. The distributed application used for the evaluation of the different techniques is introduced in Section III. Efficiency comparisons among the evaluated techniques are presented in Section IV, while the conclusions are drawn in Section V.

II. DISTRIBUTED PROGRAMMING TECHNIQUES

A. ONC RPC

Remote Procedure Call represents client/server infrastructure which increases interoperability, portability and flexibility of applications and thus enabling application to be distributed over several heterogeneous platforms. RPC decreases the complexity of application development which includes several operating systems and network protocols by isolating the application developer from the details relevant to different operating systems and network interfaces.

The concept of RPC is discussed in literature since 1976, while complete implementations have emerged in the late 1970s and early 1980s, with ONC RPC being among the most important ones.

RPC protocol enables users to work with remote procedures in same way as with local procedures. Remote procedure calls are defined through the routines contained inside RPC protocol. Each call message is associated with a corresponding reply message. RPC protocol is a message exchange protocol that also supports callback procedures on the server side.

With RPC, each server provides a program that represents a set of remote procedures. A combination of server address, program number and procedure number precisely specifies a particular remote procedure. Inside RPC model, client calls the procedure for sending the data packet to the server. When the packet arrives, server calls the dispatching routine, carries out the request and sends the response back to the client. The procedure call then returns the result to the client process.

RPC interface is usually used for the communication between processes located on different computers on the network. However, RPC functions equally successful between different processes on the same computer. [1, 2]

B. MPI

MPI [9] is a standard which defines subroutines for sending and receiving messages and performing collective operations. Due to its widespread usage in the scientific community, it has been recognized as a *de facto* standard for message-passing programming paradigm (other examples being PVM, p4, Express, etc). MPI's advantage over older message passing libraries is that it is both portable, because MPI has been implemented for almost every distributed memory architecture, and fast, because each implementation is optimized for the hardware it runs on.

In the MPI programming model, a computation comprises of one or more processes that communicate by calling library routines to send and receive messages to other processes. The number of processes participating in a computation is fixed during the run, i.e. the standard, in its original version, did not define methods for spawning new processes. The newer version of the standard (MPI-2) now allows dynamic process creation. The default programming model for MPI programs is SPMD (single program, multiple data), although there is support for more general MIMD model.

The standard itself does not preclude creation of interfaces or remote procedure access, so MPI cannot be used to implement a dynamic client/server infrastructure, i.e. the one where clients are executed independently of the server. However, most of the developed applications employ some form of client/server model in a constrained and dedicated environment. Individual clients do not represent computer users, but rather participants in a global computation process.

C. CORBA

CORBA represents middleware that provides integration, standardization and interoperability necessary in today's heterogeneous world. Modern enterprise applications are typically distributed in heterogeneous environments which involve different hardware platforms, operating systems, databases and network protocols. They consist of components written in different programming languages and often have to integrate many legacy applications that would be too expensive to rewrite or port. The only way to bypass these differences is to rely on the standardized concepts. CORBA supports the software development for these environments introducing the standard concept of distributed objects, and separating the implementation of these objects from their interfaces in a clear way by using a well defined Interface Definition Language (IDL).

CORBA is a standard for object method call through the network, and was developed by Object Management Group (OMG), a large consortium of companies determined to improve the aspects of remote object method calls. From the beginning, CORBA was developed with the goal of supporting a number of networks, operating systems and programming languages.

While CORBA on its own is not a language, it introduces a new language. CORBA services are described with a scheme which represents a template for the methods that an object exposes. Such schemes are expressed using IDL language. Programming languages such as Java, which support CORBA, can implement an IDL schema and in that way enable other software to call methods. IDL is language neutral which enables its use in every programming language for which IDL mapping exists. [3, 4]

D. Java/RMI

Java RMI is a robust and effective solution for developing distributed applications in which all included programs are written in Java. For that reason, RMI represents surprisingly simple and easy framework for utilization.

Although RMI is relatively easy to use, it constitutes a remarkably powerful technology. The primary objective for RMI designers was to allow programmers a development of distributed Java programs with the same syntax and semantics used for the non-distributed programs. To achieve this, they had to carefully map the Java class and object model of the single Java Virtual Machine (JVM) into new model in the distributed environment (multiple JVMs). As RMI functions in a homogeneous environment, there is no need for the use of the standardized paradigms such as IDL. [5]

E. DCOM

Microsoft's Distributed COM extends the Component Object Model (COM) to support the communication between objects situated on different computers on the LAN, WAN or the Internet. As DCOM is an unnoticed evolution of COM, it is possible to reuse the existing investment into COM-based applications, components, tools and knowledge for the move into the world of distributed computing based on standards.

DCOM is a high level network protocol which takes over the job, from the user, of writing network code for the control of the communication required for the interaction of distributed components over network. DCOM is not a programming language but a specification and service built using (and on top of) COM, and uses COM object oriented technology for providing its services.

By publishing DCOM, Microsoft has introduced a new set of call interfaces at the low level called Object Remote Procedure Call (ORPC). ORPC is located on top of the standard Distributed Computing Environment RPC (DCE RPC) environment and expands the procedural programming model to accommodate distributed objects. [6]

F. .NET Remoting

.NET Remoting provides a framework that enables interaction between objects over the application domains. The framework ensures many services, including a support for the activation and object lifecycle, as well as communication channels responsible for the delivery of messages to remote applications and vice versa. Formatters are used for encoding and decoding messages before their transfer over the channel. In the situation where the performance is of a critical nature, applications can use binary encoding, while in the situation where the interoperability with other distributed technologies is essential, XML encoding will be adequate. XML encoding uses Simple Object Access Protocol (SOAP) for the transport of messages from one application domain into another. .NET Remoting is designed with security in mind, so there exists a number of ways in which channel sinks can access the messages and serialized data stream before this stream is transported through the channel.

Lifecycle management of remote objects without the support of the inherent framework is often very difficult. .NET Remoting provides several activation models to be chosen from. These models belong to the following two categories:

- Client Activated Objects (CAOs)
- Server Activated Objects (SAOs)

Client activated objects are under control of a lifecycle manager based on leases, which ensures that an object is destroyed when its lease expires. In the case of server activated objects, developers can choose either *single call* or *singleton* model. The lifecycle of a singleton object is also controlled by a lease. [7, 8]

III. DISTRIBUTED APPLICATION EXAMPLE

A. Problem definition

Certain aspects of distributed programming techniques presented in this paper will be compared on a simple example which involves a simplified model of Internet Relay Chat (IRC) client/server program system.

The main method which server implements is `send_message()` that is used by client for sending its textual message. At the moment when server receives a message from client, server uses a callback mechanism and notifies all registered clients with the received message by calling a remote method `message_callback()` implemented by each client. For that purpose, some technologies (RPC, CORBA and Java/RMI) require server to implement methods `register_callback()` and `unregister_callback()`. DCOM provides an indirect support for events via Active Template Library (ATL), while .NET Remoting provides a direct support for events with which a two-way communication problem is solved. In the MPI implementation no interface or remote procedures are defined, so the IRC program is run as a simulation of the chat environment.

B. Tools used

- RPC – A *trial* version of RPC protocol implementation in .NET environment was used, called Distinct ONC RPC / XDR for .NET together with the Microsoft Visual Studio .NET 2003 development environment.
- MPI – An MPI library MPICH [10] was used, which is a free and portable implementation of the standard for both UNIX/Linux and Windows platforms.
- CORBA – An ORB implementation in Java 2 Standard Edition was used together with the Sun ONE Studio 5 Standard Edition development environment.
- Java RMI – Sun ONE Studio 5 Standard Edition development environment was used.
- DCOM – Server was developed in Microsoft Visual C++ 6.0, while the client was developed in Microsoft Visual Studio .NET 2003 development environment.
- .NET Remoting – Microsoft Visual Studio .NET 2003 development environment was used.

Microsoft Windows XP Professional was used as a platform.

IV. COMPARISONS

A. Stability

Stability, i.e. maturity of a technology can effectively be measured by a time period that a particular technology has been an active participant of the market.

TABLE I
Stability of each distributed technique

Technique	**Year of appearance**
RPC	1988
CORBA	1991
MPI	1994
Java/RMI	1996
DCOM	1996
.NET Remoting	2002

B. Portability

The portability of a programming technique reflects the amount of work needed to transport an application from one programming language or computing platform to another. Some properties of the described methods regarding portability are given in Table II.

TABLE II
Portability of each distributed technique

Technique	Portability
RPC	RPC represents a specification that is referred in several RFCs which means that it can execute on each platform for which exists an RPC support, but it is mainly confined to UNIX platforms. It is possible to use any programming language for which a development version of RPC protocol exists.
MPI	MPI is a standard that defines communication subroutines and as such can be used with virtually any platform and programming language. Current implementations support C/C++ and Fortran languages on UNIX/Linux or Windows operating systems.
CORBA	As CORBA represents a specification, it can be used on any platform for which an ORB implementation is developed. The same is valid also for the choice of programming language since this choice is dependent on the existence of ORB libraries for that particular language.
Java/RMI	It can be executed on every platform for which a Java Virtual Machine exists. Since it greatly uses Java Object Serialization, only Java programming language can be used.
DCOM	It can be executed on each platform for which an implementation of COM run-time environment exists. Since it is a specification on a binary level, a whole array of programming languages can be used.
.NET Remoting	Since .NET Remoting requires the existence of .NET Framework on the platform on which it is executing, currently only Microsoft Windows operating system supports it. It is possible to use any programming language capable of translating to Common Intermediate Language (CIL).

C. Amount of work for implementation

In Table III we show the minimal amount of code needed for each of the techniques to implement a simple communication mechanism.

TABLE III
Comparison of implementation code

Technique	Implementation code

RPC	***Interface definition***
	
	program RPCIRC_PROG {
	  version RPCIRC_VERS {
	    void register_callback(int)=1;
	    void send_message(string,string)=2;
	    void unregister_callback(void)=3;
	  }=1;
	}=0x20021016;
	
	Instantiating remote object
	
	server=new rpcirc(System.Net.Dns.Resolve(serverTB.Text)
	  .AddressList[0],true);
	
	Registering callback procedure
	
	int progNo=Pmap.getTransient(1,0,false);
	callbackServer=new RPCIRCCallback(progNo);
	server. register _callback_1(progNo);
	
	Sending message to server
	
	server.send_message_1(nickTB.Text,
	  messageTB.Text);

MPI	***Identifying processes*** MPI_Comm_size(MPI_COMM_WORLD, &numprocs); MPI_Comm_rank(MPI_COMM_WORLD, &myrank); ***Sending message to server*** MPI_Send(message, LEN, MPI_CHAR, n, 99, MPI_COMM_WORLD); ***Sending message to clients*** MPI_Bcast(message, 255, MPI_CHAR, 0, MPI_COMM_WORLD);

CORBA	***Interface definition***
	
	module CORBAIRC {
	  interface CORBAIRCCallback {
	    void message_callback(in string message);
	  };
	  interface CORBAIRCServerI {
	    void register_callback(in CORBAIRCCallback
	      callbackClient);
	    void send_message(in string message);
	    void unregister_callback(in CORBAIRCCallback
	      callbackClient);
	  };
	};
	
	Instantiating remote object
	
	java.util.Properties props=new java.util.Properties();
	props.put("org.omg.CORBA.ORBInitialPort","900");
	props.put("org.omg.CORBA.ORBInitialHost",
	  serverTF.getText());
	orb=ORB.init(new String[] {},props);
	POA rootpoa=POAHelper.narrow
	  (orb.resolve_initial_references("RootPOA"));
	NamingContextExt
	  root=NamingContextExtHelper.narrow
	    (orb.resolve_initial_references("NameService"));
	NameComponent[] name=new NameComponent[1];
	name[0]=new
	  NameComponent("CORBAIRCServer","");
	server=CORBAIRC.CORBAIRCServerHelper.narrow
	  (root.resolve(name));

	Registering callback procedure
	
	callbackServer=new
	CORBAIRCCallbackImpl(this,orb);
	rootpoa.activate_object(callbackServer);
	callbackServerRef=
	CORBAIRC.CORBAIRCCallbackHelper.
	narrow(rootpoa.servant_to_reference(
	callbackServer));
	server.register_callback(callbackServerRef);
	rootpoa.the_POAManager().activate();
	Thread callbackServerThread=new
	Thread(callbackServer);
	callbackServerThread.start();
	
	Sending message to server
	
	server.send_message(nickTF.getText()+":
	"+messageTA.getText());

Java/RMI	***Interface definition***
	
	Server
	
	package JRMIIRC;
	import java.rmi.*;
	
	public interface JRMIIRCServerI extends
	java.rmi.Remote {
	void register_callback(JRMIIRCCallback
	callbackClient) throws RemoteException;
	void send_message(String message) throws
	RemoteException;
	void unregister_callback(JRMIIRCCallback
	callbackClient) throws RemoteException;
	}
	
	Client
	
	package JRMIIRC;
	import java.rmi.*;
	
	public interface JRMIIRCCallback extends
	java.rmi.Remote {
	void message_callback(String message) throws
	RemoteException;
	}
	
	Instantiating remote object
	
	server=(JRMIIRC.JRMIIRCServerI)
	Naming.lookup("rmi://"+serverTF.getText()+
	"/JRMIIRCServer");
	
	Registering callback procedure
	
	callbackServer=(JRMIIRC.JRMIIRCCallback) new
	JRMIIRCCallbackImpl("JRMIIRCCallback",this);
	server.registriraj_callback(callbackServer);
	
	Sending message to server
	
	server.send_message(nickTF.getText()+":
	"+messageTA.getText());

DCOM	***Interface definition***
	
	import "oaidl.idl";
	import "ocidl.idl";
	[
	object,
	uuid(4ED9E6AD-AB0F-4F93-B911-
	515A0EB19609),
	dual,
	helpstring("IDCOMIRCServerImpl
	Interface"),
	pointer_default(unique)
	]
	interface IDCOMIRCServerImpl : IDispatch
	{
	[id(1), helpstring("method send_message")]
	HRESULT send_message([string] wchar_t
	*message);
	};
	[
	uuid(43CCE165-2922-4583-B502-
	D042391552F7),
	version(1.0),
	helpstring("DCOMIRCServer 1.0 Type
	Library")
	]
	library DCOMIRCSERVERLib
	{
	importlib("stdole32.tlb");
	importlib("stdole2.tlb");
	[
	uuid(3A3A8AA6-8DBF-4AF1-8E0F-
	CAA645D545F0),
	helpstring("_IDCOMIRCServerImplEvents
	Interface")
	]
	dispinterface _IDCOMIRCServerImplEvents
	{
	properties:
	methods:
	[id(1), helpstring("method message_callback")]
	HRESULT message_callback([string] wchar_t
	*message);
	};
	[
	uuid(9A64C80B-C8A9-461D-BA75-
	7507D3528565),
	helpstring("DCOMIRCServerImpl Class")
	]
	coclass DCOMIRCServerImpl
	{
	[default] interface IDCOMIRCServerImpl;
	[default, source] dispinterface
	_IDCOMIRCServerImplEvents;
	};
	};
	
	Instantiating remote object
	
	server=new DCOMIRCSERVERLib.
	DCOMIRCServerImplClass();
	
	Registering callback procedure
	
	server.message_callback+=new

	DCOMIRCSERVERLib.
	_IDCOMIRCServerImplEvents_message_
	
	callbackEventHandler(server_message_callback);
	Sending message to server
	
	server.send_message(nickTB.Text+": "+messageTB.Text);

.NET Remoting	***Interface definition***
	
	namespace General
	{
	public delegate void messageEventDelegate (string message);
	public interface IREMOTIRCServer
	{
	event messageEventDelegate messageEvent;
	void send_message(string message);
	}
	}
	
	Instantiating remote object
	
	RemotingConfiguration.Configure ("REMOTIRCClient.exe.config");
	RemoteHelper remoteHelper=new RemoteHelper();
	server=(IREMOTIRCServer) remoteHelper.getObject(typeof(IREMOTIRCServer));
	
	Registering callback procedure
	
	server.messageEvent+=new messageEventDelegate(new MessageEvent(new messageEventDelegate(server_messageEvent)). server_messageEvent);
	
	Sending message to server
	
	server.send_message(nickTB.Text+": "+messageTB.Text);

D. Security issues

RPC protocol defines only authentication. Authorization has to be solved by user.

MPI standard does not define any authorization methods, so security issues are left to the authors of a particular implementation to resolve. Since MPI applications are meant to execute in a closed and dedicated environment, there is usually no ready-made support for secure communication.

CORBA Security Service provides a complete framework for the security of distributed objects. It supports authentication, authorization and non-repudiation.

Java Authentication and Authorization Service (JAAS) provides a powerful mechanism for authentication and authorization that supports many security systems such as Windows NT, UNIX, Kerberos and Keystore.

DCOM provides one of the most advanced and complex security models applied in distributed systems. DCOM security is tightly coupled with Windows NT security which offers many advantages over other operating systems since NT security is a fundamental part of the operating system. Authentication and authorization are, of course, supported.

With .NET Remoting, authentication and authorization are indirectly solved when using IIS as activation agent, while in other cases there is a need for an implementation of custom sinks and sink providers with adequate security functionality.

V. CONCLUSION

We derive our conclusions based on implementations of the example application described in Section III. All of the programming techniques can be used on all the computing platforms where matching implementation is available, except for .NET which is currently supported only for Windows. Considering that .NET technology is the newest one, we can expect its portability to improve over the next few years.

The 'easiest to implement' techniques were shown to be Java RMI and MPI, while only the former is capable of creating open server/client infrastructures. Java RMI and DCOM incorporate the most sophisticated secure application development tools, whereas MPI has none of those features. CORBA, as a specification, does not define secure methods - rather, it offers CORBA Security Service, which is as yet available in a smaller number of implementations.

REFERENCES

[1] Cory Vondrak, *Remote Procedure Call*, December 2004, (http://www.sei.cmu.edu/str/descriptions/rpc.html)

[2] *AIX Version 4.3 Communications Programming Concepts, Chapter 8 Remote Procedure Call*, December 2004, (http://www.unet.univie.ac.at/aix/aixprggd/progcomc/toc.htm)

[3] Suhail Ahmed, *CORBA Programming Unleashed*, Macmillan Computer Publishing, 1998

[4] Gerald Brose, Andreas Vogel, Keith Duddy, *Java Programming with CORBA – Advanced Techniques for Building Distributed Applications, Third Edition*, Wiley Computer Publishing, 2001

[5] David Reilly, Michael Reilly, *Java Network Programming and Distributed Computing*, Addison Wesley, 2002

[6] Brian Hoang, *DCOM: Overview and Architecture*, December 2004, (http://sern.ucalgary.ca/Courses/CPSC/547/F98/Slides/Hoang/DCOM.html)

[7] Ingo Rammer, *Advanced .NET Remoting (C# Edition)*, Apress, 2002

[8] Scott McLean, James Naftel, Kim Williams, *Microsoft .NET Remoting*, Microsoft Press, 2003

[9] MPI Standard (http://www.mpi-forum.org/)

[10] MPICH (http://www.mcs.anl.gov/mpi/mpich/)