The Microsoft .NET Remoting Framework

Distributed Systems Fall 2002
Uppsala University, Sweden

September 30, 2002

Submitted by:

Michele Berlingerio 801070-7852 <berling@cli.di.unipi.it>
Richard Heggbrenna 770125-6997 <rihe8139@student.uu.se>
Tony Johansson 730225-1637 <tojo3423@student.uu.se>

Abstract:

In a distributed system where communication between clients and remote resources exists, applications can access resources that reside on different computers. This requires a technology that offers the necessary features that makes it possible for applications to gain access to remote resources. Under the Microsoft .NET Framework, the .NET Remoting Framework is used to achieve this. The aim with this study is to examine how the .NET Remoting Framework works. This document is a literature study performed using public documents and books. The study resulted in an overview that can be used as an introduction of the .NET framework.

Keywords: Microsoft .NET Remoting Framework, distributed system.
Table of Contents

INTRODUCTION ......................................................................................................................................1

OBJECTIVES ....................................................................................................................................2

METHOD .......................................................................................................................................2

EXPECTED RESULT ...............................................................................................................................2

DELIMITATION ...................................................................................................................................2

THE MICROSOFT .NET FRAMEWORK .............................................................................................3

.NET REMOTING OVERVIEW ............................................................................................................3

.NET REMOTING COMMUNICATION ..................................................................................................4

THE REMOTING INFRASTRUCTURE .................................................................................................5

CHANNELS AND FORMATTERS ........................................................................................................5

REMOTE OBJECTS ...............................................................................................................................6

THE REMOTING PROCESS ..................................................................................................................7

DISCUSSION ....................................................................................................................................9

CHOICE OF METHOD .........................................................................................................................9

QUALITY OF THE SOURCES ............................................................................................................9

THE EXPECTED VERSUS THE ACHIEVED RESULT ............................................................................9

FURTHER READING ............................................................................................................................9

CONCLUSIONS ................................................................................................................................9

List of Tables and Figures

Figure 1 The .NET framework [2]. ......................................................................................................3
Figure 2 The Remoting process [5]. .....................................................................................................7
Table 1 Remoting Scenarios [6]. ........................................................................................................8
Introduction

There are a lot of different definitions of a distributed system in the literature, as you can read in [1]. According to Tannenbaum and van Steen a sufficient definition of a distributed system is [1, p2]:

"[…] a collection of independent computers that appears to its users as a single coherent system".

Let us introduce you to the idea of a distributed system. Imagine that you want to access resources such as processors, remote objects, database entries and so on, without having them on your machine. Imagine that you do not want to take care of problems like connections, security and atomicity of your operations; then what you need for your purposes is a distributed system, which can provide you with features such as transparency, openness and scalability.

The implementation of these features depends on many factors such as the type of service the systems should offer to the users; for example, a remote printing system does not fulfill the goal of the location transparency since the user knows where the printer is physically located. Scalability interests the system developers, not the user. Developers should make the system modifiable in the future in terms of dimension.

The openness feature refers to the fact that a distributed system should offer a service following some rules known to users; a description of those rules can be found for example in some RFC (Request For Comments).

Now, imagine you are on a MAC machine and you want to access a resource located on a PC running Linux. How can you access this resource? First of all, is it really possible? Sometimes different machines are using different representations of data, or different ways to communicate. In this case a distributed system should have a layer called middleware, which interfaces the operating system with the distributed application. More generally, it is possible to say that a middleware can solve problems due to the heterogeneous nature of the machines.

There are different kinds of middleware models. These are sometimes called paradigms [1], which are used for describing different communications and distribution types. Examples for those paradigms are distributed file system model and Remote Procedure Call (RPC) model. A few years after the birth of RPC, the idea of remote procedure calls was applied to objects, making it possible to invoke methods to remote objects. This technology is called Remote Method Invocation (RMI).

Remote object invocation permits you to use any public method and property belonging to a remote object, making it possible to use it like a local resource. Different ways to pass the object are used depending on what you want to do with it. If you would like to modify something in it then is not enough to have a local copy of the object.
A distributed database is a good example for distributed systems. In fact, it introduces problems like security, fault tolerance and simultaneous accesses to shared objects when performing read or write operations.

Now it is easy for the reader to understand that developing distributed applications requires hard work, which could made easier using a powerful programming platform which provides high-level libraries to the programmer.

**Objectives**

The aim of this paper is ¹:

- To extend the writers' knowledge in a selected area of distributed systems through research.
- To become familiar with distributed systems research resources.
- To develop skills in integrating information from multiple resources
- To become familiar with doing research within a group.

**Method**

The method used in this paper is a literature study. The majority of the documents used as sources, have been acquired from the MSDN² online library. One document has been acquired from the ACM³ online catalogue.

**Expected result**

The expected result is an overview of the Microsoft .Net Remoting Framework.

**Delimitation**

The study focuses on Microsoft .NET Remoting features in general, it will not go into details about all the different scenarios. The aim is not to provide any sample code or any information about the details of the implementation.

---

¹ [http://user.it.uu.se/~hsander/Courses/DistributedSystems/assignment1.htm](http://user.it.uu.se/~hsander/Courses/DistributedSystems/assignment1.htm)
² [http://msdn.microsoft.com](http://msdn.microsoft.com)
³ [http://www.acm.org](http://www.acm.org)
The Microsoft .Net Framework

The .NET framework is a platform from Microsoft Corporation that is intended to simplify application development in a distributed environment [2]. As explained by Aaron Weiss [3], the core concept of the .NET is a proposal for a distributed operating system. Unfortunately, due to Microsoft's effort to promote brand consistency and applying the .NET brand as a suffix to all their new products and services, it has become hard for the public to understand what .NET is really about.

The .NET framework has two main components: the Common Language Runtime (CLR) and the .NET framework class library. The CLR works as a middleware [1, p.36] to the underlying hardware. All compilers that support the .NET framework compile the source code into an intermediate language called Microsoft Intermediate Language (MSIL). When the programs that are compiled to target the .NET framework are executed, the CLR compiles the MSIL code using a feature called Just-In-Time (JIT) to the native code that runs on the hardware. The code that targets the CLR is called "managed code" and the code that does not target the CLR is known as "unmanaged code" [2]. The .NET framework class library is a set of object-oriented classes that make it much easier to learn and use new features of the .NET framework.

Figure 1 The .NET framework [2].

---

4 Some papers suggest that the .NET Framework has got 3 main components: The CLR, the class library and ASP.NET.
**NET Remoting Overview**

The .NET Remoting framework offers a simple yet powerful programming model and runtime environment for making distributed applications. All the services required for communication between applications developed using the .NET platform are provided in the .NET Remoting architecture. Such applications can reside on the same computer in different application domains, work on different computers in the same Local Area Network (LAN), and even be scattered across the world in connected networks. In such a case the connected hosts must support and use the CLR. [4]

The .NET Remoting infrastructure is an abstract approach to interprocess communication. Much of the system operates without drawing attention to itself, appearing to be a local system when it is in fact a distributed system. For example, distributed applications use remote objects automatically across different application domains. Either the object is passed as a copy to the client, or referenced remotely on the server. The application domain can exist within a single process located on a single machine or be located in different processes on different machines. There can also be several application domains in the same process. [5]

The real strength of the .NET Remoting system, however, resides in its ability to enable communication between objects in different application domains or processes using different transportation protocols, serialization formats, object lifetime schemes, and modes of object creation. In addition, if you need to intervene in almost any stage of the communication process, for any reason, remoting makes this possible. [5]

Whether you have implemented a number of distributed applications or are simply interested in moving components to other computers to increase the scalability of your program, it is easiest to understand the .NET remoting system as a generic system of interprocess communication with some default implementations that easily handle most scenarios. The following discussion begins with the basics of interprocess communication using .NET remoting. [5]

**.NET Remoting Communication**

Communication between different processes requires a server object whose service is available to outside callers, a client that makes remote calls on the server object, and a transportation mechanism channeling calls and responses from one end to the other. A server object is available to a client in one of two ways; as a copy or as a reference. When a server object is copied, in its entirety, and sent to the client process by value its functionality is directly available to the client since it's residing in the client's process. The object will now reside in the client's memory and be executed in the client's process. This method of using the server object could be used when the client repeatedly needs the provided service. Performance and use of bandwidth could be improved since fewer remote calls and responses are needed between the client- and server object. However, many server objects cannot, nor should they, be copied or moved to some other process for execution. Large objects containing many methods and complex
structures are most likely not needed by a client. Often, only a few of the remote object's methods are used to retrieve information. The other methods contain information that is not usable. For example, a file handle held by the remote object is only valid in the server’s context not the clients. In addition, memory and execution time is preserved when no unnecessary executable structures are added to the client. There is also a potential security leak when objects are passed since these objects can be used to gain information regarding internal data.

In the cases where a copy of the remotable object might not be a good solution, a reference to the server object could be passed to the client instead. This reference is located in the client process but is not executed. Instead, the remoting system takes care of the client call and locates the right server object and executes it and then responds with information to the calling client. This preserves bandwidth, supports low coupling and information hiding among other things.

The Remoting Infrastructure

The key feature of the remoting infrastructure is the ability to handle communication between client and server objects. The system enables the client to call and execute methods on the remote object, creating an illusion as if they were executing in the same process, when they are in fact separated in different processes. The remoting infrastructure uses proxy objects to achieve this impression to the client. Proxies are stand-in objects that represent themselves as some other object. When a client calls for the creation of a remote object, the remoting system creates a proxy object called a TransparentProxy that, to the client looks exactly like the remote object, and places it in the client's process. Under the hood there is also another proxy created called a RealProxy. The RealProxy takes care of forwarding of the message across a configured channel to the receiving channel. All methods of the remote object are now callable through the use of the proxy. In order for this cross-process communication to work, means of setting up connections, the use of protocols, parameter passing and sending of bytes is provided with a channel.

Channels and formatters

Channels are used to physically move bytes in form of messages from and to remote objects. A channel can be either simplex or duplex, depending on the purpose of it. In the .Net Framework there are two channels predefined: TcpChannel and HttpChannel. The two predefined channels are using one of the native formatters in the .NET runtime. The TCP channel is using a binary formatter to serialize the message and uses raw sockets to transmit data across the network. This formatter is normally used if the remote object is located on the same LAN as the client and gives the advantage of a low overhead. When a remote object resides outside the LAN, the HTTP channel is used since it makes it possible to access objects behind firewalls. The HTTP5 channel is using the SOAP6 formatter. [4]

5 Hypertext Transport Protocol <http://www.w3.org/Protocols/>
The server is responsible for creating and registering a channel with the remoting framework. The channels must be registered before the remote object is registered. When a channel is created and registered it is set to listen to one or more ports. A channel is application domain specific and the domain can have several channels. Channels cannot be set to listen to the same port, even if they belong to different application domains. A channel does not belong to any particular remote object so it can service any number of remote objects. A channel is only available when the process that created them is alive. [5]

**Remote Objects**

In the .NET Remoting Framework some objects are called remote objects. They are instances of classes that work well in distributed systems and their functionality is public to clients. There are two types of remote objects in particular:

- Marshal-by-value (MBV) objects, which are copies passed to other application domains.
- Marshal-by-reference (MBR) objects, for which a proxy is created and used by a client to access the object remotely.

In addition to the types, all remote objects contain a state that is somehow related to the client. On an abstract level they can either have or not have a state. This is usually represented as state full or stateless remote objects. A state full object can contain and preserve state between different method calls made by a client while a stateless object cannot. Under the .NET Remoting Framework, objects can be configured in three possible ways to serve as remote objects:

- Single Call
  This is a one to one mapping between a client and a remote object. These objects do not hold state. An object configured this way can be used in a load-balancing way.

- Singleton Objects
  This is a remote object created in only one instance that can serve many clients at the same time. They can store states, which is shared among the clients. They are useful in cases when data needs to be shared explicitly between clients and also when overhead of remote object creation and maintenance is of importance. A client creates the object if there is not already an existing instance that the client can use.

- Client-Activated Objects
  Are objects in a one to one mapping between clients and remote objects. These objects are used when client state is important and needs to be preserved between method calls

---

on the remote object. Every time a client creates an object of this kind a new instance of the remote object is created. [6]

**The Remoting Process**

The illustration below shows how an application running on one computer can use services provided by a type that is stored somewhere else, perhaps on another computer.

A reference to the created proxy is given to the client by the remoting infrastructure. When a client calls a method on the TransparentProxy the CLR intercepts the call and examines whether the object is on the same computer or a remote computer. If the object that is being called resides on the same computer as the client, the CLR simply makes a conventional method call. If the object resides on a remote computer, the CLR marshals the parameters into a message and then sends the created message over to the server process through a registered channel, using the RealProxy. On the server side, a listening channel receives the message and forwards it to the server remoting system, which calls the object. If there is a response by the server any return values are packed into a new message and then sent back to the client side the reverse way.

If the remote object is Marshaled By Value, there are no proxies created on the client side.
Remoting Scenarios in .NET

The following table simply shows a list of possible client-server combinations, including the payload and the protocol used by default. It is also possible to write own channels and formatters. [6]

Table 1 Remoting Scenarios [6].

<table>
<thead>
<tr>
<th>CLIENT</th>
<th>SERVER</th>
<th>PAYLOAD</th>
<th>PROTOCOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Managed / Unmanaged</td>
<td>.Net Web Services</td>
<td>SOAP/XML</td>
<td>http</td>
</tr>
<tr>
<td>.Net Component</td>
<td>Unmanaged Classic COM Component</td>
<td>NDR (Network Data Representation)</td>
<td>DCOM</td>
</tr>
<tr>
<td>Unmanaged Classic COM Component</td>
<td>.Net Component</td>
<td>NDR</td>
<td>DCOM</td>
</tr>
</tbody>
</table>
Discussion

Choice of method

The method chosen is very appropriate for these types of study. A different approach could have been to write applications that use the .NET Remoting Framework, and from the empirical results draw conclusions. However, that would probably take much more time than the literature study, although we feel that it would be more interesting.

Quality of the sources

Unfortunately, except for one article, we have not been able to find any objective criticism concerning the .NET Remoting Framework. However, for the aim of this study the articles acquired from the MSDN Online Library can be considered sufficient.

The expected versus the achieved result

The expected result was an overview of the Microsoft .Net Remoting Framework. This goal has been achieved.

Further reading

We suggest that searching the MSDN Online Library for further information can be a starting point for the interested reader.

Conclusions

The Microsoft .NET Remoting Framework provides a technology that can be used for developing distributed applications that can access resources remotely in a simple way.
References


