Design and Implementation of NetSolve using DCOM as the Remoting Layer

Ganapathy Raman

Department of Computer Science
University of Tennessee
Knoxville, TN 37996
raman@cs.utk.edu

Jack Dongarra

Department of Computer Science
University of Tennessee
Knoxville, TN 37996
dongarra@cs.utk.edu

Technical report UT-CS-00-440

University of Tennessee

May 2000

Abstract

NetSolve is a RPC based client/agent/server system that allows one to remotely access both hardware and software components. The Distributed Component Object Model protocol (DCOM) is an application-level protocol for object-oriented remote procedure calls supported on Microsoft Windows Platforms. NetSolve currently has its own custom remoting layer. This report documents the design and implementation of an experimental NetSolve version that leverages the facilities provided by DCOM to simplify and enhance the remoting layer.

This work was supported by Microsoft Research and a grant from the National Science Foundation #ACI-9876895

1. Introduction

NetSolve is a software infrastructure that enables its users to realize the vision of Grid Computing. A NetSolve enabled system consists of 3 major components

- *NetSolve Clients*
- *NetSolve Agents*
- *NetSolve Servers*

End users (clients) interact with the NetSolve grid infrastructure through an appropriate NetSolve Client Interface. The strengths of NetSolve include simplicity of its client interfaces and the wide variety of language support. NetSolve client interfaces can be accessed from C, Fortran, Matlab, Mathemetica, Java and Excel. NetSolve client-users gain access to aggregate resources without the users needing to know anything about computer networking or distributed computing. In fact, the user does not even have to know remote resources are involved.

The NetSolve agent maintains a database of NetSolve servers along with their capabilities (hardware performance and allocated software) and dynamic usage statistics. It uses this information to allocate server resources for client requests. The agent finds servers that will service requests the quickest, balances the load amongst its servers and keeps track of failed ones.

The NetSolve server is a daemon process that awaits client requests. The server can run on single workstations, clusters of workstations, symmetric multi-processors or machines with massively parallel processors. A key component of the NetSolve server is a source code generator that parses a NetSolve problem description file (PDF). This PDF contains information that allows the NetSolve system to create new modules and incorporate new functionality's. In essence, the PDF is the interface between NetSolve and the various service providers.

The (hidden) semantics of a NetSolve request are:

1. *Client contacts the agent for a list of capable servers.*
2. *Client contacts server and sends input parameters.*
3. *Server runs appropriate service.*
4. *Server returns output parameters or error status to client.*

Steps 2 through 4 in the NetSolve protocol are accomplished by using a remoting layer to simulate the RPC. The current version of NetSolve has its own custom RPC implementation. This has the advantage that NetSolve is not tied to any particular RPC implementation. At the same time RPC has been around a long time and most operating systems have a mature RPC implementation supported by default. In particular all Microsoft platforms support an object oriented RPC mechanism called DCOM. Leveraging a standard RPC implementation has the following potential benefits

- *The remoting layer in NetSolve is simplified*
- *Can support more sophisticated remoting scenarios*
- *Helps separate NetSolve functionality from RPC details*

This report documents the design and implementation of an experimental version of NetSolve that uses DCOM as the remoting layer.

2. DCOM

DCOM is the remoting layer of the Microsoft Component Object Model (COM). The Component Object Model (COM) is a component software architecture that allows applications and systems to be built from components supplied by different software vendors. COM can be summarized as providing the following services

- *Defines a binary standard for component interoperability*
- *Is programming language independent*

- *Enables components to evolve gracefully*
- *Is extensible*
- *Is Location Transparent*
- *Supports rich error reporting*
- *Allows for dynamic loading and usage of components*
- *Is Secure*

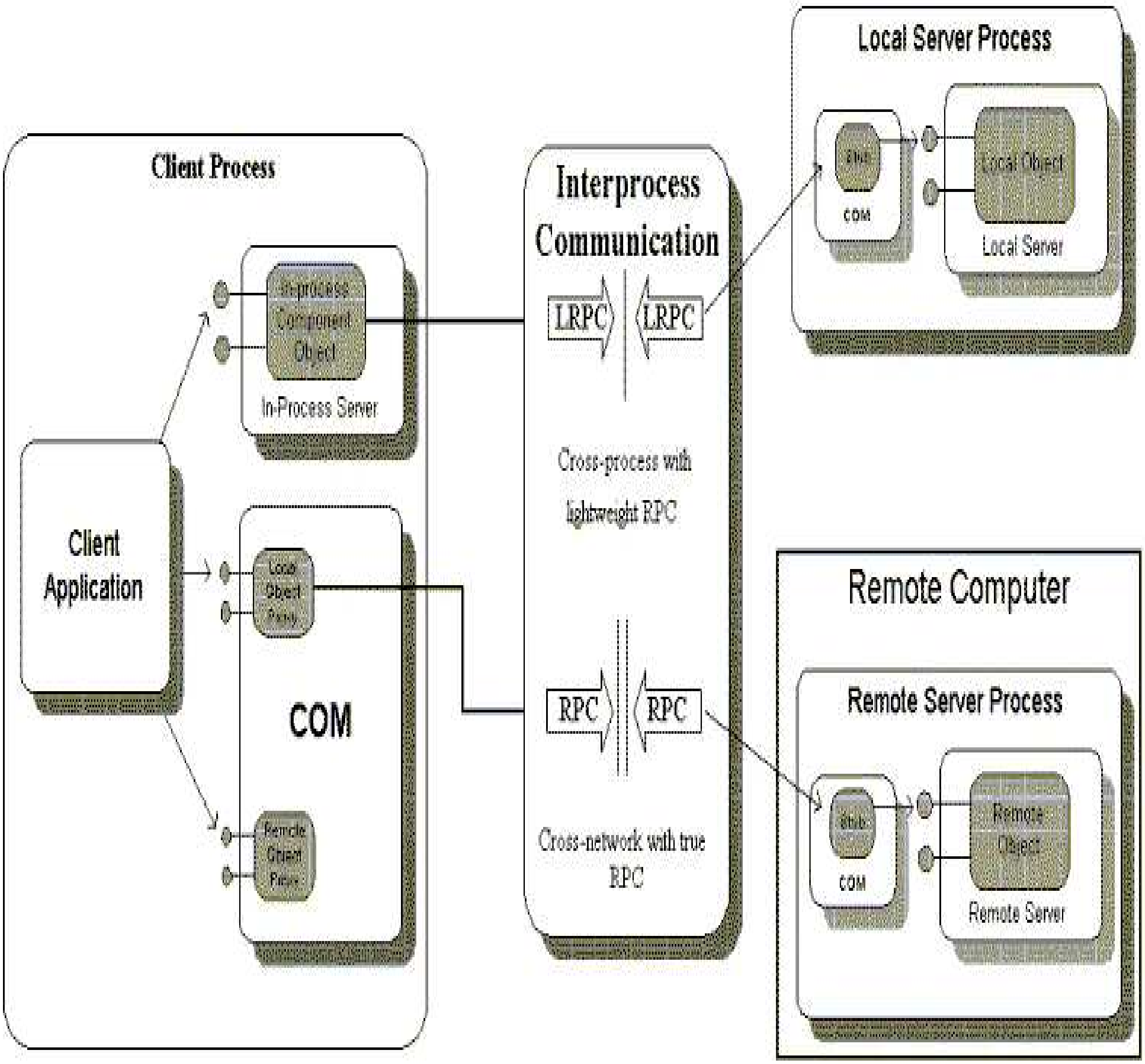
The features that primarily interest us are location transparency, dynamic usage of components and security. The following sections look into the above in detail.

2.1. Location transparency

COM is designed to allow clients to transparently communicate with components regardless of where those components are running, be it the same process, the same machine, or a different machine. What this means is that there is a single programming model for all types of component objects for not only clients of those component object, but also for the servers of those component objects.

From a client's point of view, all component objects are accessed through interface pointers. A pointer must be in process, and in fact, any call to an interface function always reaches some piece of in-process code first. If the component object is in process, the call reaches it directly. If the component object is out-of-process, then the call first reaches what is called a "proxy" object provided by COM itself, which generates the appropriate remote procedure call to the other process or the other machine. Note that the client from the start should be programmed to handle RPC exceptions; then it can transparently connect to an object that is in process, cross process, or remote.

From a server's point of view, all calls to a component object's interface functions are made through a pointer to that interface. Again, a pointer only has context in a single process, and so the caller must always be some piece of in-process code. If the component object is in process, the caller is the client itself. Otherwise, the caller is a "stub" object provided by COM that picks up the remote procedure call from the "proxy" in the client process and turns it into an interface call to the server component object.



2.2. Dynamic Usage of Components

Most RPC implementations require the client application to know the function signatures of the remote method calls at compile time. But in the case of NetSolve new problem solvers can be added at run time and hence it is not possible for the NetSolve client library to meet the above requirement. COM helps us solve this problem elegantly through a feature called Automation. Automation defines the following

- ***Type information***

This is usually in the form of a *.tlb file that contains the function signature information in a standardized format.

- ***Standard Invocation format***

It defines a minimal function signature that both the client and server should agree upon. The function is of the form

Invoke (id of method, array of arguments)

When a client calls an automation enabled server through the Invoke call, the COM library uses the type information stored in the *.tlb file to effect the RPC call. There is of course a cost associated with using Automation but it is one that cannot be avoided in the case of NetSolve.

2.3. Security

DCOM gives both callers and objects a range of choices to determine how the data on the connection is to be secured. The overhead in terms of machine and network resources tends to grow with the level of security. DCOM therefore lets applications dynamically choose the level of security they require.

The process of authenticating the caller can be relatively expensive, depending on the security provider and the security protocol it implements. DCOM lets applications choose if and how often this authentication occurs. DCOM uses the security framework provided by Windows NT. The Windows NT security architecture supports multiple security providers, including:

- *Windows NT NTLM authentication protocol*
- *The Kerberos Version 5 authentication protocol*
- *Distributed password authentication (DPA)*
- *Secure channel security services (SSL)*
- *Public key based security*

DCOM currently offers two fundamental choices with regard to data protection: integrity and privacy. Clients or objects can request that data be transferred with additional information that ensures data integrity. If any portion of the data is altered on its way between the client and the object, DCOM will detect this and automatically reject the call. Data integrity implies that each and every data packet contains authentication information. However, data integrity does not imply that no one can intercept and read the data being transferred. Clients or objects can request that all data be encrypted. Encryption implies integrity check as well as per-packet authentication information

3. Design and Implementation

The design is fundamentally similar to that of NetSolve with one exciting addition. This version supports ‘dynamic code download’. The original version of NetSolve tied servers with problem solvers at compile time. This meant that every time a new problem solver had to be added to a server, it had to be stopped and re-compiled. In the current design the problem solvers are stored in a separate ‘Repository’. When a server gets a request to solve a particular problem, it contacts an appropriate repository and downloads the solver component at run time. This has the following potential advantages

- ***Real-time adjustment to demand***

Servers are not pre-loaded with all possible solvers. Instead as and when requests arrive solvers are downloaded. Also if a problem is not accessed for a period of time, the server can discard it without affecting its operation

- ***Easier maintenance***

The service provider doesn’t have to install problems on new servers. All he has to do is bring the server up and point to the appropriate repository.

- ***Flexibility***

To improve performance the service provider can opt to pre-load servers with frequently used problems.

- ***Better code sharing and version control***

The entities that provide hardware resources and those who write the problem solvers can be in different administrative domains. Also no single repository need have all the information, since they can form a distributed network of their own. New versions can be added to repository and some kind of ’cache invalidation’ can used to make sure that servers have up to date solver components.

3.1. Architectural Components

In the NetSolve world one can identify 4 different entities

- *Clients*
- *Service providers*
- *Software providers*
- *Infrastructure providers*

One can obtain a better understanding of NetSolve by looking at the roles played by the above entities and their interactions with the grid infrastructure.

Clients

These are the entities that consume ‘power’ from the grid and use it in their applications. They interact with the NetSolve grid infrastructure thro a client API. The API is designed to be simple and is usually supported on a wide variety of languages.

Service Providers

These are the entities that provide the ‘power generators’ that feed ‘power’ into the grid. They set up NetSolve servers that can run solver components on behalf of clients.

Infrastructure Providers

These entities are responsible for maintaining the entry points into the NetSolve grid. They maintain NetSolve Agents and NetSolve Repositories. They also enforce grid usage policies and accounting.

Software Providers

They are responsible for writing the problem solver components. The process to create a new solver component is as follows

1. Write the problem description

The problem description file (.pdf) contains information such as

- *ID of problem*
- *Function signatures*
- *Compile/Build information*
- *Meta Data information*

2. Generate the component

The problem description file passed thro a 'Component Generator' that generates 3 files

a. *The component exe (.exe)*

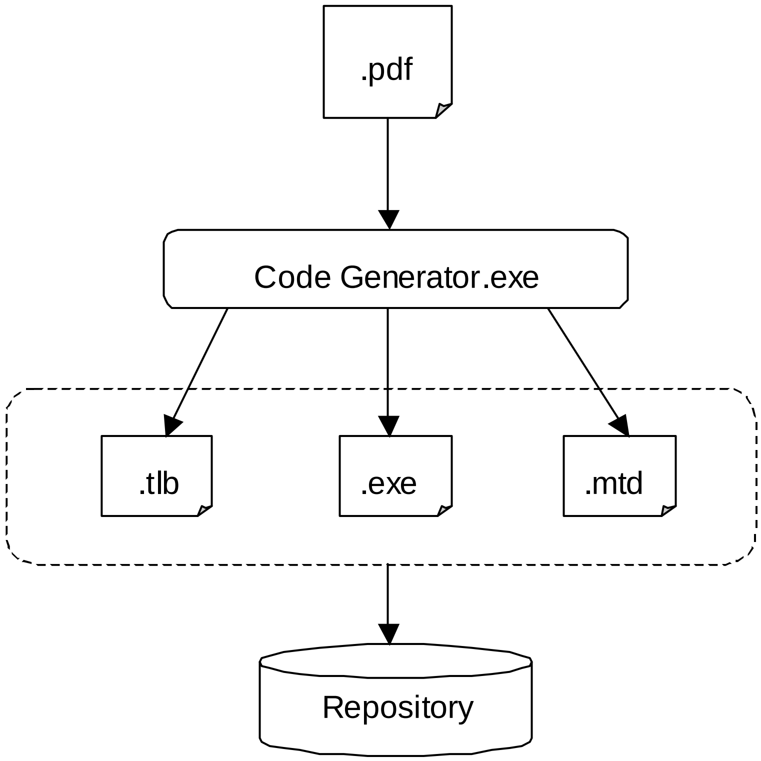
b. The component type info (.tlb)

c. Associated Meta Data (.mdata)

3. Publish to repositories

The 3 generated files are stored in NetSolve repositories for download by NetSolve servers. The repositories inform the NetSolve agents about the new solver components.

The following figure illustrates the process



3.2. Software Components

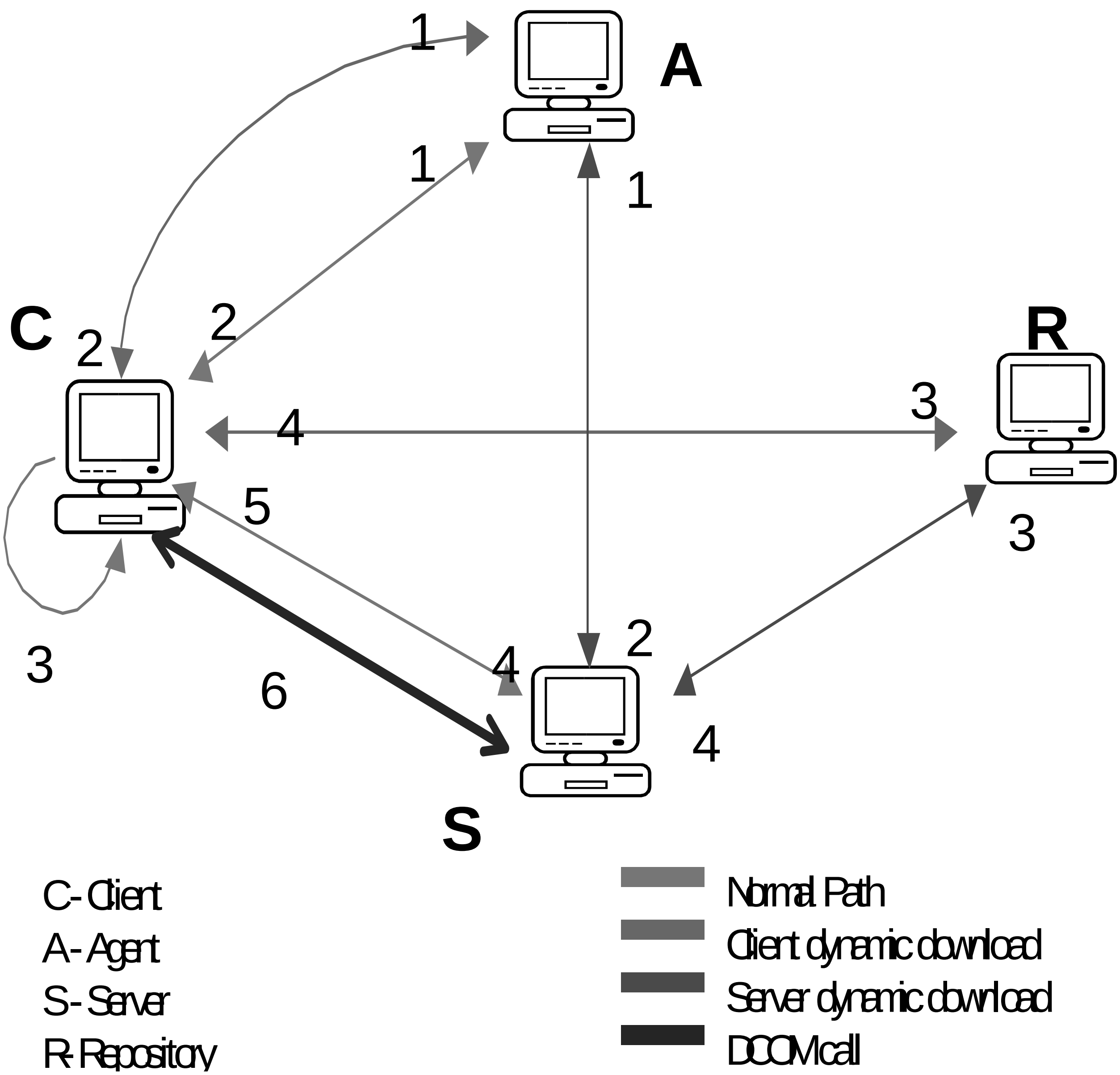
From a software engineering viewpoint a NetSolve system consists of 4 active and 1 passive component. The passive component is the 'Code Generator' that is used to generate problem solver components. The active components are

- *Agent*
- *Repository*
- *Server*
- *Client Library*

The Agent, Repository and Server are multi-threaded servers supporting the TCP/IP Internet protocol. The actual type of the Client Library will depend upon the language used by a user to interact with the NetSolve system. Currently only a static C library is supported.

3.3. Sample Scenario

Looking at the steps that occur when the user solves a problem can help us attain a better understanding of the NetSolve internal protocol. This also illustrates the power of dynamic code download.



Step 1: Client contacts the agent to get details on the problem

Step 2: Agent returns problem details and list of servers

Step 3: Client uses the details to determine if it needs to download the components (*.tlb file) associated with the problem. If it does not have the necessary components, it initiates dynamic download procedure

Client dynamic download

Step 1: Client contacts the agent for list of repositories

*Step 2***:** Agent returns list of repositories

Step 3: Client contacts repository for problem components (*.tlb file)

Step 4: Repository returns the appropriate components (*.tlb file)

Now client has the capability to solve the problem

Step 4: Client informs the server to get ready

Server uses the details sent by the client to determine if it needs to download the components (*.tlb and *.exe files) associated with the problem. If it does not have the necessary components, it initiates dynamic download procedure

Server dynamic download

Step 1: Server contacts the agent for list of repositories

Step 2: Agent returns list of repositories

Step 3: Server contacts repository for problem components (*.tlb, *.exe files)

Step 4: Repository returns the appropriate components (*.tlb, *.exe files)

Now server has the capability to solve the problem

Step 5: Server informs the client that it is ready

Step 6: The actual DCOM call is made and the problem is solved

4. Conclusions

The goal of the project was to evaluate the effects of replacing the NetSolve custom RPC protocol with a standard RPC implementation. The experience overall has been a good one. The dynamic download capability has proven itself to be a useful one and its implementation would be much harder if it were not for the facilities provided by DCOM. But on the negative side we lose portability since DCOM is supported only on windows platforms. One of the criticisms of NetSolve has been that it is just RPC and nothing more. This pilot shows conclusively that NetSolve provides much more services than just RPC. DCOM or RPC plays only a small role in the overall services provided by the NetSolve grid infrastructure. NetSolve is clearly a front-runner in helping end users realize the vision of grid computing.

5. Acknowledgments

I am most fortunate to have as my adviser Dr. Jack Dongarra. I thank him for giving me the freedom in pursing my thoughts. Thanks to Dr. Terry Moore for his encouraging comments and advice. Thanks to Dorian Arnold for his help with NetSolve. And finally thanks to Dr. Wolski for his many thought stimulating seminar classes.

6. References

1. D.Box, *Essential COM*, Addison-Wesley, 1998.

2. H.Casanova and J.Dongarra, *NetSolve: A network server for solving computational science problems*, The International Journal of Supercomputer Applications and High Performance Computing, 11(3):212-213, 1997.

3. G.Eddon and H.Eddon, *Inside Distributed COM*, Microsoft Press, 1998.

4. Microsoft, *Component Object Model Specification*, version 0.9, October 1995.

5. Microsoft, *Distributed Component Object Model Protocol*, version 1.0, January 1998.

6. Microsoft, *Microsoft Developers Network (MSDN)*, January 2000.