The DFN-RPC, a Remote Procedure Call Tool

Rolf Rabenseifner

25 February 1992

The DFN-RPC, a Remote Procedure Call Tool, has been developed to distribute scientific-technical application programs between a workstation and a Compute Server. An introduction of RPC-technology will be given by showing the specifications of DFN-RPC. With a demo application program, essential aspects of RPC-distribution will be discussed. All processes will run under user-account.

1 Introduction

The DFN Remote Procedure Call Tool (short: DFN-RPC) has been developed by the computing center of Stuttgart University on behalf of the German Research Network (DFN-society) in cooperation with four pilot users.[1]

For a scientist or engineer, DFN-RPC makes it possible to run programs distributed between his workstation and a remote supercomputer. Quick calculations e.g. for simulations on the Compute Server, graphical visualisation on the local workstation and realtime interaction opens the possibility of working in a new and better way and helps shorten develop and research cycles.

2 The DFN-RPC

R. Rabenseifner[2]

2.1 Basics

A Remote Procedure Call is a powerful, but also easy-to-use method to distribute programs between two computers.

Fig. 1 shows the distribution of simulation routines (in the lower part of the figure) to a remote computer. Parts of the Remote Procedure Call Tool are illustrated as grey blocks.

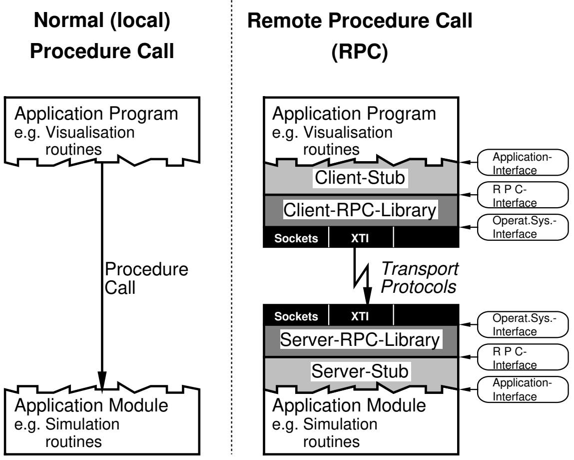

Figure 1: from a ”Local” to a ”Remote Procedure Call”

The ”stubs” supply client-side entries, which have the same procedure header as the specific simulation routines, but instead of its procedure body they have a short part of code which transmits the parameter list via network to the server ”stub”. The server gets a procedure number, all field lengths (to allocate the parameter list) and all input parameters, see fig. 2. With all these informations, the server ”stub” rebuilds its own parameter list and calls the real simulation subroutine. After the routine has ended, all output parameters are sent back to the client ”stub”. While sending and receiving, input/output parameters are converted automatically due to specific storage of numerical values on different computers. [NEL 81]

Normally, more routines that run via RPC and

---

[1] Alfred-Wegener-Institut, Bremerhaven (AWI); Institut für Kernenergetik und Energiesysteme, Stuttgart (IKE); Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen, Stuttgart (ISD); Lehr- und Forschungsbereich theoretische Astrophysik, Tübingen (TAT)

[2] email: rabenseifner@rus.uni-stuttgart.de

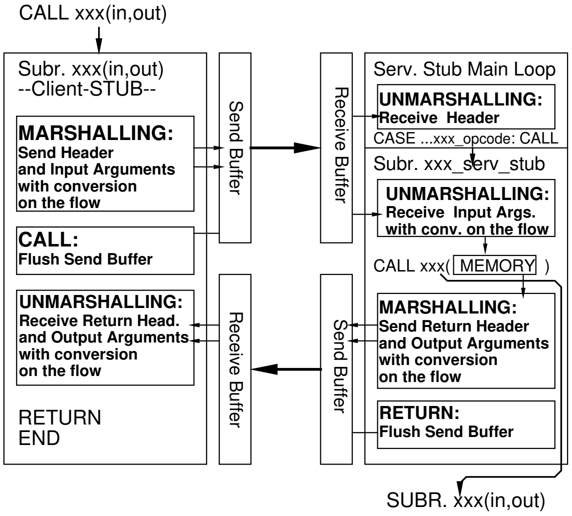

Figure 2: the "Marshalling", the function of the "stubs"

belong to the same context are concentrated in a single process.

2.2 Marshalling

Sending and receiving of parameters is called "Marshalling" and "Unmarshalling". Many computers store floating-point values internally different (byte ordering and format), therefore, while transferring the values, conversion is necessary. These are the main strategies for conversion:

1. Transmission in a fixed, predefined format, e.g. IEEE big-endian. This is the easiest solution, but if the two computers use a different, but internally same format (e.g. two VAX computers), data is converted senselessly twice. e.g. see also SUN-RPC [SUN 90].
2. The transfer format is freely choiceable.
3. Using the format of the sender. This solution has an disadvantage on a 64 bit Compute Server, because a floating-point value is transmitted unnecessarily with a length of 8 bytes to the workstation via network and there it is converted into a floating-point value with a length of 4 bytes.

   See also OSF/DCE [DCE 90].

DFN-RPC uses the second method, with *IEEE big-endian* as the predefined format.

2.3 Interface definition

The "stubs" are application-depended, but there is no need to program them manually. They can be produced automatically by using their interface definition which consists mainly of their procedure headers. DFN-RPC must substitute the call-by-reference with a data transmission (call-by-in-copy-and-out-copy), so additional informations are needed:

- is in-copy and/or out-copy really necessary or is it a workspace parameter?
- how can an array length be determined at runtime?
- is it possible to transmit a parameter with less precision?

DFN-RPC has been developed especially for Fortran applications, but it can also be used for C programs. Fig. 3 shows a short example with so called "RPC qualifiers" integrated.

```
C*[RPI_BEG]
      SUBROUTINE exchange (n,a,b,wrk)
      INTEGER  n      [IN]
      REAL     a(n)   [IN,OUT]
      REAL     b(n)   [IN,OUT]
      REAL     wrk(n) [DUMMY]
C*[RPI_NON]
      DO 1 i=1,n
         wrk(i) = a(i)
         a(i)   = b(i)
         b(i)   = wrk(i)
    1 CONTINUE
      END
C*[RPI_END]
```

Figure 3: example of an RPC interface definition

These qualifiers make it possible to have a single source, with no need to gather procedure headers into a interface definition file.

Fig. 4 shows some more qualifiers. MAX-INDEX defines the length of variable arrays, IN and OUT define the transfer regions and TRANSFER helps reducing the precision of transfer parameters.

```
C*[RPI_BEG]
      SUBROUTINE sss(n,a,b,c)
      INTEGER  n [IN]
      REAL  a(*) [MAXINDEX = n*(n+1),
     +            IN =     1 : n*n
     +            OUT= n*n+1 : n*n+n]
      REAL b(3,*)[MAXINDEX = n+1,
     +            IN = (1:3, 1:n),
     +            OUT= (1:3, n+1:n+1)]
      DOUBLEPRECISION c(n)[TRANSFER*4]
C*[RPI_END]
```

Figure 4: Complex qualifiers

In principle, it is possible to merge qualifiers into the source as a comment in special continuation lines:

```
      INTEGER n
C*RPC+          [IN]
```

Besides marshalling of normal data types, DFN-RPC supports also:

- alternate return jumps,
- "callback" (a server-routine which has been called via RPC can call also another routine on client side via RPC),
- entry statements with different parameter lists,
- assembling of server routines into a module,
- separate generation of stubs for different modules,
- synchronous, asynchronous (not-blocking) and asynchronous buffered RPCs.

See also the chapters to follow.

2.4 Stub generator

Fig. 5 shows the compilation scheme for single source programs. The stub generator splits the source into client and server part and produces the stubs.

It is recommended that the programmer mounts (e.g. with NFS) the disks which contain client/server source and the stubs to the client and server computer.

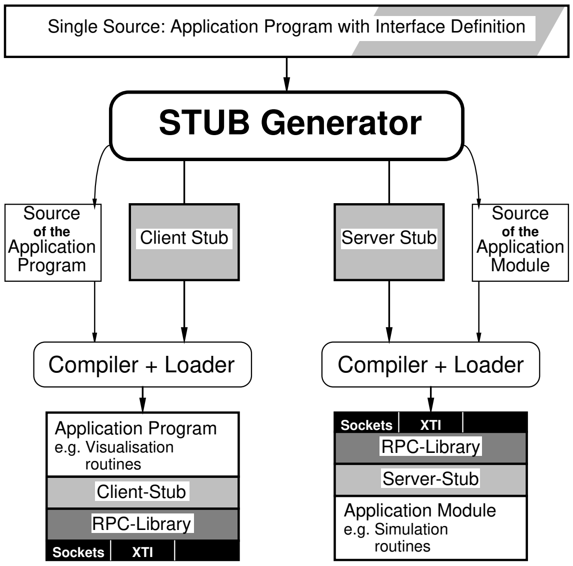

Figure 5: compilation scheme for single source programs

2.5 Separate stub-generating

It is possible to make interface definitions apart from application programs, then, procedure bodies aren't needed anymore, see fig. 4.

Separate stubs can be used

- with remote libraries, this means access to existent libraries via RPC from a remote computer; this can be a numerical library on a vector computer (Compute Server) or also a visualisation library on a workstation (client=supercomputer, server=workstation)
- with modified or manually written stubs. Stub-generating consists of three logical phases: After splitting the source into client and server part, the stubs are generated in a portable macro-code. Afterwards, this code is converted into compilable code. In special cases, if the interface definition language is not powerful enough, the programmer has the possibility to create or change the macro file himself.

Fig. 6 shows the scheme of stub-generating in general.

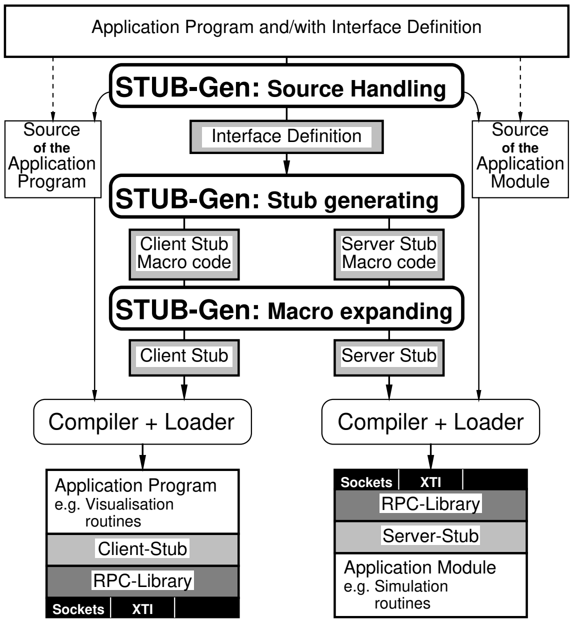

Figure 6: the generation of stubs in detail

2.6 Synchronizing

Normally, a RPC is fully synchronized. As shown in fig. 7, the server waits for the client to send a RPC order. The client waits after transmission of the input parameters until the server routine has ended and the output parameters have been sent back. This is equivalent to a normal subroutine call.

By using this *synchronous* RPC, you cannot exploit the parallelism of the processors involved.

Therefore, DFN-RPC allows two other forms of synchronizing: *Asynchronous* and *asynchronous buffered* RPCs. Output parameters aren't allowed in procedures using one of these two forms of synchronizing. In both forms, the client application program retakes control after marshalling of the input parameters. The order of calling the server routines is set by the RPCs of the client program. Using "flushcontrol" (the network can be seen as a pipe with a limited buffer-length), the client can be blocked until the server process has received all input data. A server routine which has been called via an asynchronous RPC can't use callback.

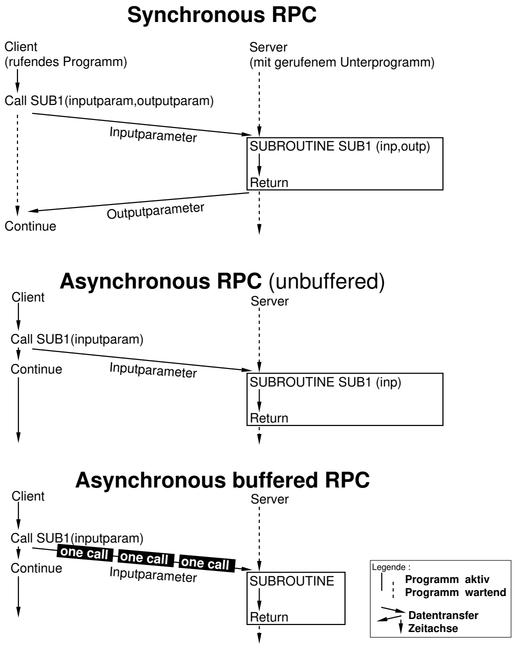

Figure 7: Synchronizing

If client and server processes don't share data (like memory, files on a file server, etc.), the asynchronous RPCs are semantically identical to normal procedure calls.

2.7 Starting of processes

A distributed application is composed of at least two processes: the client process containing the main program and the server process with the RPC-callable subroutines. Multiple server processes are also possible.

DFN-RPC makes it possible that all processes run under user-account.

All processes involved can be started manually, but also beginning with a process (call *Master*[3]), all other processes can be started automatically.

The DFN-RPC product includes some prototype examples concerning this matter. It is possible to use any existent starting-tool, like rsh or nqs. In principle, the transport connection needed for data transport is build from Slave to Master, see fig. 8. The Master-process opens a port (Transport Service Access Point, TSAP) and tells the slave the port-number

---

[3] The Master process can be the client process or a server process

when starting the process. Then, the Slave can build the connection to the Master.

If this method is used, no ”portmapper”[4], no globally known port number[5] or application number[6] are needed.

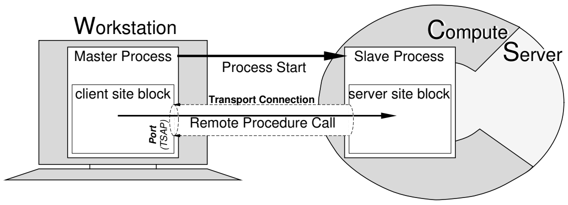

Figure 8: Starting and connecting of processes

2.8 Protocol and semantics

With DFN-RPC, you can use as transport protocol: TCP/IP, Ultra TP4 or ISO-OSI TP4, see fig. 9, [TAN 89].

The user chooses a protocol-stack simply by giving a specific address. The protocol-stacks themselves differ highly in interconnectivity, transfer-rate and ”round-trip-time”[7].

The connection-orientated transfer-protocols grant a so called ”exactly-once semantic”, this means all RPCs have completed successfully if the application gets no error message from DFN-RPC (from start to end of the connection), this is also valid for asynchronous RPCs.

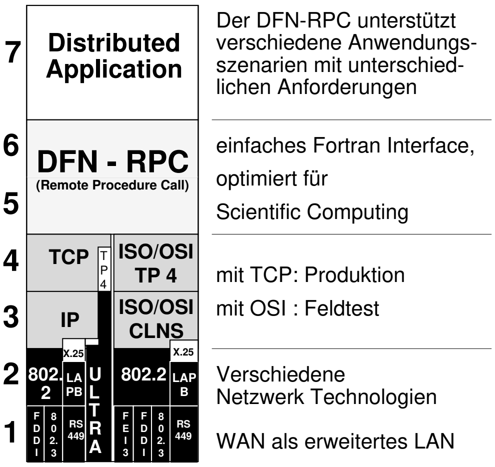

Figure 9: DFN-RPC in an ISO-7-Layer-Model

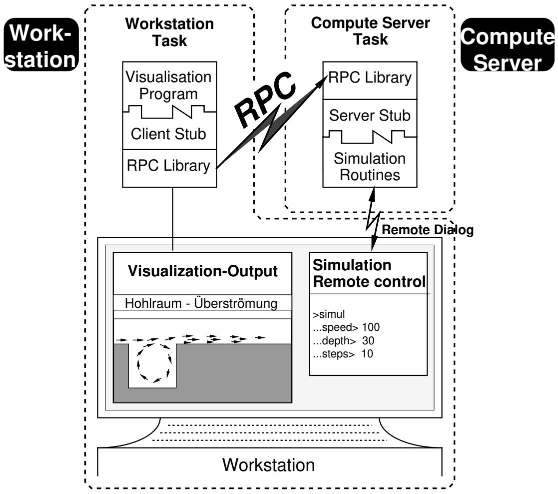

Figure 10: overflow characteristics of hollow rooms

3 Distributed applications

Fig. 11 shows some typical application cases how applications like scientific-technical simulations and visualisation between a workstation and a Compute Server can be distributed.

The program has been split into a compute-part, a visualisation-part and a graphics-library. In fig. 11, the compute-part contains the calculation of the simulation (then it’s a ”distributed simulation and visualisation”), if it contains a strong calculation-part of the visualisation, it’s a ”distributed visualisation”.

In case A, the main program runs on the workstation (the Client) initiating RPCs to the compute-part on the Compute Server.

The example in fig. 10 shows distributed visualisation. The integration of the flow- and stroke-lines to illustrate the overflow characteristics of hollow rooms is calculated on the Compute Server based on a time-variant 30 MByte flow-field. These integrations were put on the Compute Server, because the visualisation must run animated and, despite using 30 MByte, no Paging is allowed. A scientist has

---

[4] Deamon under Root-Account all ports from application programs are reported to, see OSF/DCE [DCE 90]

[5] see SUN-RPC [SUN 90]

[6] e.g. the Universal Unique Identifier UUID from OSF/DCE

[7] Time needed for an Remote Procedure Call with an empty server routine

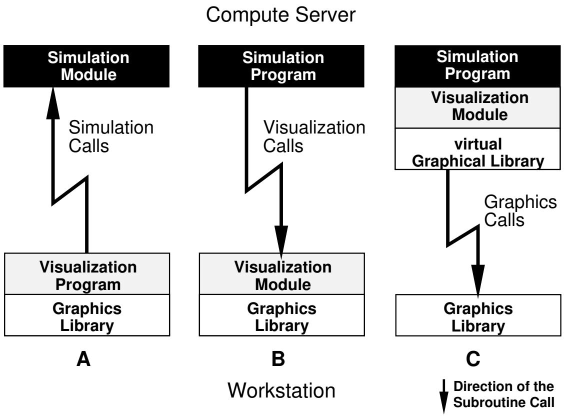

Figure 11: some application cases

now the possibility to set color-probes interactivly, he can study the simulated flow looking at stoke-lines it produces. Furthermore, he can select different kinds of visualisation via menue, he can e.g. put in "sawdust" to follow the resulting track-lines immediately. Thus it is possible to study graphical-interactivly the results of the flow calculation with help from a supercomputer.

In case B, the main program runs on the Compute Server (the Client), the workstation is now the Server (in RPC-logic).

To put the cases A and B into reality, an easy-to-use Remote Procedure Call Tool is needed. It is important that while designing the program, the RPC-interface is made in a way that a minimum of data and RPCs are sent via network afterwards.

In case C, the complete application program runs on the Compute Server, only the graphics-library is located on the workstation.

In most cases, you will use X Window, PEX[8] or DGL[9]. If another graphics library is needed, this can be realized using RPC tools.

Mostly all RPCs for this case can be made using asynchronous buffered calls.

The use of a distributed graphics library is very comfortable, especially for an application programmer, because he must only substitute the graphics library with the remote graphics library.

References

[ALL 92] Allrutz R., Rabenseifner R.: Der DFN-RPC, ein Remote Procedure Call Tool. Proceedings, S. 523-532, 15. DECUS M"unchen Symposium, 1992.

[DCE 90] Open Software Foundation (OSF): Remote Procedure Call in a Distributed Computing Environment. White Paper, OSF, Oct. 1990.

[ERT 91] T. Ertl, et al.: Visualisierung in der Astrophysik. Lehr- und Forschungsbereich Theoretische Astrophysik der Universität Tübingen. Beitrag zum Workshop Visualisierungstechniken, Rechenzentrum der Universität Stuttgart und Gesellschaft für Informatik, FG 4.1.2. Kurzfassungen der Beiträge. Stuttgart. Juni 1991.

[NEL 81] Bruce-Jay Nelson: Remote Procedure Call. Computer Science Department, Carnegie-Mellon University, Pittsburgh, Pennsilvania. 3. Mai 1981.

[RAB 91] Rolf Rabenseifner: Distributed Applications between Workstation and Supercomputer using ISO/OSI Protocols. Proceedings, Twenty-Seventh Semi-Annual Cray User Group Meeting, S. 80-84. London. 1991.

[SUN 90] SUN microsystems: Network Programming Guide. Part Number 800-3850-00, 1990.

[TAN 89] A.S. Tanenbaum: Computer Networks. Prentice-Hall International, 2.Ed. 1989.

This paper is the english translation of the first part of [ALL 92]. The second part of [ALL 92], written by R. Allrutz, describs distributed applications in theoretical astrophysics.

---

[8] PHIGS in X Windows

[9] Distributed (Iris) Graphics Language