Intercomputing Interactions in Re-engineering Technology for Distributed Computation

A.A. Sapozhnikov, A.P. Sapozhnikov, T.F. Sapozhnikova
Laboratory of Information Technologies, JINR

Abstract

A new technology is proposed for integration the old standalone Fortran - written computational programs into more large distributed computer systems. This is a re-engineering technology, because the main developer's tool becomes a F2F program for source modules converting. This converter provides the maintenance of all rules, needed for transformed program, created initially for monoprocessor computer systems, into Computational Server, working in distributed network area. The principles of F2F converter and details of communications between Client and Server are discussed.

Starting from the early 90-s of XX century, the object-oriented programming technologies, based on C++ and Pascal languages, have been intensively progressed. Now it is already impossible to imagine a serious application, working in an old MS-DOS style, i.e. with no mouse and modern windowed graphics. But the large amount of numeric programs, fundamental for modern applied systems, comes into PC-world from an antique epoch of mainframes. Actually it means that these programs were Fortran-written.

The static nature of Fortran does not allow it to be the general tool for object-oriented programming. Moreover, the current generation of programmers prefers C++ and Delphi, i.e. more dynamic languages. On the other hand, all the difficulties in numeric algorithms programming does not depend upon a language used, but only depend on understanding the algorithms themselves. As for a serious numeric program, to redesign it from Fortran to C++, one needs to know all delicate details of it, i.e. to be its author, but the author is most likely to be very far from active programming now! Using of language converters like F2C [1] cannot resolve the problem because of:

- the program is still remaining a "black box" as it was before;
- there are no converters from Fortran to C++ or Pascal.

That is why we must suppose that the large numeric programs made in the old days with using Fortran, will be forced to stay in their Fortran-incarnation in the foreseeable future. Some of these programs, being very popular in the past, still stay indispensable at present, because they personify the unique experience of outstanding specialists. Below we intend to discuss namely about such kind of programs. Let us call them for short "unique programs".

Another particular feature of the present-day programming is the tendency to distribute calculation between several computers in the computer networks. In general they may be even of different types. The existing technologies of parallelization, such as MPI [2], are oriented to decomposition of the whole computational job onto a number of smaller processes. As a rule, this decomposition is performed manually while software development or modernization. At the same time, there is no technology to integrate the large standalone made computational blocks into huge-scale distributed systems.

In [3, 4] we proposed a new technology for integration of unique programs onto more large distributed systems. This is a re-engineering technology, because the main

developer's tool became a program for converting of standalone-developed programs onto integrated parts of large distributed computer systems. The general architecture of proposed distributed computational system also seems to be rather non-traditional. User's workstation contains a client-process one of whose aims is to distribute jobs among a number of independent Computational Servers. Each of them is a separate process, which executes a corresponding program. Requests on input-output operations, needed for servers, are interpreted by Client.

The key idea is the automation of building the Computational Server from standalone unique programs, without any manual transformation of their sources. Exactly this idea is called here "re-engineering". This offers an opportunity to integrate well tested during long time old programs, made by bygone generation of experienced programmers, into large-scale modern present-day systems of data processing, including advanced visualization facilities, databases and others mechanisms for human-computer communications.

Our approach to integration is not a general solution, we do not pretend to universal usage, as the Service-Oriented Architecture (SOA) does. Our main aim is to propose the technology and software tools namely for the unique programs, mentioned above, without any modification of their source codes. Moreover, the terms "Client" and "Server" themselves are here rather relative, to not invent a new substances. They designate only two processes, which communicate with each other in Local Area Network or even inside the single computer.

The Server's aim is to solve a specific computational job, ordered by Client. While executing this job, Server can consume the external data from standalone-made files or directly from Client. Server can produce output data for Client. All needed input-output operations are ordered in Server's side using traditional Fortran operators Read/Write, but will be interpreted on Client's side. An output data visualization may be treated as the particular way of such interpretation.

As for Client - it is the single point for communication with the User. This way offers an opportunity for full separation of two stages: solving of computational problem and interpretation of results. Therefore, for each of these stages we can use both the most appropriate tools and the different and independent groups of developers.

A special F2F (Fortran-To-Fortran) converter has been created for transformation of old unique programs into Computational Server. The converted program works in own separate address-space, maybe in remote computer. Particularly, this increases the reliability of the whole system, because possible errors on Client-side cannot influence errors on Server-side. F2F organizes all needed environment for interaction with Client. F2F automatically substitutes all Input/Output operators in Fortran source for calling special subroutines, realizing a Client-Server interaction protocol. The auxiliary information is built into these subroutines: current running Input/Ouptut operator, source line number, name of current data file and current processed record number. It gives an opportunity to get a detailed information about possible I/O-Errors, that's why the converted program becomes even better than the original one. Besides, F2F substitutes all operators OPEN of opening data files for calling special subroutine, which can search specified files following various rules. Further, F2F builds into all I/O operators catching possible errors for preventing the Server against unexpected "hanging" in it. Another quite attractive F2F feature is the ability to build into converted program the debugging information about subroutines calling. It is useful while looking for errors in computational program.

This is an example of how F2F works:

```
Subroutine S              ! here is fragment of the original:
c=sin(a)+cos(b)           ! - some computations
Open ( unit=Lun,file='myfile.dat',status='old' )
Write (1),a,b,c           ! binary output to file
Write (*,*),a,b,c         ! console output
Read ( *,'(i5)',err=9 ) n ! console input with own error catching
Stop
End

Subroutine S              ! The result of converting:
Use ComFort        ! here is all environment and interface with Client
c=sin(a)+cos(b)           ! of course all computations remains intact!
mess = 'subroutine S, line 14:   Open(unit=Lun,file='myfile.dat',...'
Call Open_File( Lun, 'myfile.dat', 'old', 'formatted' )
if(fo_error.ne.0) goto 12345
mess = 'subroutine S,  line 15:   Write(1),a,b,c'
Write( 1, err=12345 )  a,b,c         ! catching of possible I/O errors
mess = 'subroutine S,  line 16:   Write(*,*),a,b,c'
Write ( iobuf, err=12345 )  a,b,c    ! format transformation
Call InterfaceIO ( jwrite )          ! then  output re-addressing
mess = 'subroutine S,  line 17:   Read(*,'(i5)',err=9),n'
Call InterfaceIO ( jread )           ! input re-addressing
Read ( iobuf, '(i5)', err=9 ) n        ! then format transformation
Call Instead_Of_Stop                 ! normal finish of Server
12345 Call IOError ( mess )          ! abnormal finish of Server
End
```

The aim of the F2F converter is to transform a program, destined for interaction directly with human, into a program for interaction with the Client-process while working in a distributed computer system. Actually, F2F is a translator from Fortran to Fortran. Namely, F2F is the basic re-engineering tool in our technology.

Though F2F has to perform rather a deep syntactical analysis of the program being converted, it works more quickly than a native Fortran-compiler. Initially, we supposed to exploit F2F only in a semi-automatic mode, because Fortran has a very complicated syntax and has no grammar at all. In some cases the human-help assumed. However, we had managed all the syntactical problems. Now F2F does not need any manual revision for resulted Fortran- code. Capacity F2F for work was practically tested on a lot of real numeric programs with huge size.

Communications interface between Client and Computational Server

A simple symmetric model is proposed for communications between Client-program and any of its Servers. The single basic unit in this model is a line of text, i.e. a sequence of chars with a special char at the end. Analyzing the input line, both sides can monosemantically interpret what to do. For example, requests from Client to Server:

For example, requests from Client to Server:

```
Execute  MyFavoriteJob.dll          - run specified program
Take Line  <arbitrary line>         - answer for text input
Give Time                           - request of elapsed time
Suspend
Resume
Stop
```

The similar requests from Server to Client:

```
Take Line  <arbitrary line >        - text output
Give Line                           - waiting for text input
```

The only structural requirement to these communications is to be non-synchronous. This allows for both sides to stay permanently active, not being in waiting status without special necessity. To meet this requirement, there is enough to use classical socket mechanism, which appeared for the first time in OS UNIX and then was successfully transferred into Windows.

Moreover, having chosen Delphi as the main tool for programming in both sides, the whole Client and Server's communication level, we used TServerSocket and TClientSocket classes from rich Delphi Component Palette. These classes were developed by Borland Inc. specially for using in inter-computer communications based on socket mechanism. They ensure required non-synchronosity, because they can process specific event "OnSocketRead".

While working in a network area, the most serious problem is a non-predicted size of information, being received by consumer in the single input. Therefore, if, for example, two lines were sent into the socket entry, there will be beforehand unknown number of "OnSocketRead" events on consumer's side (one, two or more)! In other words, information must be received "piece-by-piece", then "glued" and finally again separated into two lines! This trouble can be very elegantly avoided, if we will use the following rules while programming "OnSocketRead" event handler:

- there is a buffer, initially empty, containing the beginning part of line, being already received;
- current received portion of chars is entirely added to the buffer. If the buffer doesn't contain now the end of line - "OnSocketRead" error declared as not happened, i.e. is not processed;
- if the end of line appeared in the buffer - the first line will be entirely withdrawed from buffer, processed, then "OnSocketRead" event will be called recursively.

Corresponding code for these rules got even shorter than their verbal description!

Usage of the socket mechanism offers to Client and Server to be absolutely independent upon their mutual location - in the same computer or in the different computers in network. As for Server - there is up to it, who is connecting with it. As for Client, being forced to specify the address or name of Server, there is enough to specify the keyword "localhost" to designate the Server, located in the same computer.

The described above interface is the second basis of the proposed technology for the distributed system building. We ought to specify, that this technology does not oblige to build the Server part of system namely from the standalone Fortran-written programs. If the Computational Server is developed "from zero", the developer can choose:

1. to use his own tools observing non-burdensome requirements of our interface;
2. or to use good old Fortran with habitual operators for Input/Output, and then convert his program via F2F.

This approach particularly allows one to involve into Computational Servers development the experienced specialists on numeric methods, not wishing to exceed the framework of their habitual Fortran, while the young developers can concentrate on the Client development with usage of more contemporary technologies.

Further perspectives

The idea of automatic source converting seems to be extremely productive. Particularly, many source transformations needed to integrate with MPI-package, may be successfully performed by our F2F converter. For example, all MPI-programs must perform Input/ Output operations not for all processes, as MPI-paradigm SPMD requires, but as the rule - by the single master-process [2]. All needed checkings, broadcastings, initial and final MPI- operations can be automatically added by F2F-converter while porting the program under MPI-package. Such automation of routined works while parallelization of unique programs, initially developed for execution on single processor, can be an important direction for the F2F-technology evolution.

Conclusions

The re-engineering technology for automation of distributed computational system building from standalone Fortran-written programs is proposed and developed. The essential features of this technology are:

- usage of the specially developed F2F-converter to automate Computational Servers generation;
- usage of the specially developed standard for communications between Client and Server;
- the ability for Client to communicate at the same time to several Servers. This allows dynamically build large-scaled computational systems.

It is important for us, that the programmatic achievement for all elements of the proposed F2F - technology was performed exclusively by our own efforts, without usage of any foreigner, especially commercial software.

References

[1] S.I.Feldman, P.J.Weinberger, A Portable Fortran 77 Compiler. UNIX Time Sharing System Programmer's Manual, Tenth Edition, Volume 2, AT&T, Bell Laboratories, 1990.

[2] MPI: The complete Reference. MIT Press, Cambridge, Massachusetts. 1997.

[3] Sapozhnikov A.P., Sapozhnikova T.F., "Reengineering technology for distributed computation with using LAN". Proceedings of the International Conference "Distributed Computing and Grid-Technologies in Science and Education" (Dubna, June 29 - July 2 2004). D11-2004-205, Dubna, JINR, 2004. pp.183-190.

[4] www.jinr.ru/programs/jinrlib/f2f-technology/index.html