Bull. Nov. Comp. Center, Num. Anal., 12 (2003), 49–55

A programming instrument for developing distributed applied systems for the mathematical modeling*

V.I. Kozlov

1. Introduction

Modern problems of the mathematical modeling include a very wide range of computational tasks. Those tasks are based on solving different problems of mathematical physics, especially, in engineering. Such systems can be considered as passing of sets of data through the nodes of a graph. An example of such a graph is presented in Figure 1.

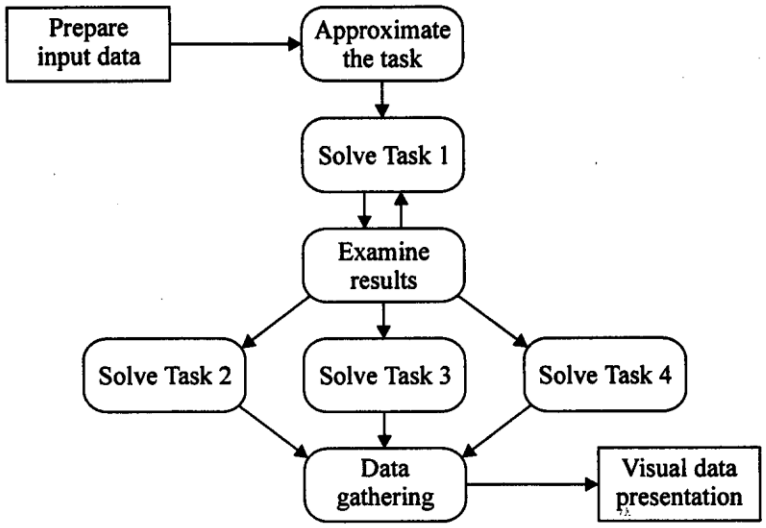

Figure 1

Obviously, the resource requirements are different for each process of a diagram. For example:

- Preparing the input data requires a usual desktop workstation;

*Supported by the Russian Foundation for Basic Research under Grant 01-07-90367.

- Approximation of the original task can require a powerful computer with single processor;
- Solving algebraic tasks requires a multiprocessor unit;
- The visual data presentation usually requires a powerful graphical workstation.

As consequence of the above-said distributed computational modeling systems must be designed as systems deployed on different computer units with different operating systems. Now there are a few techniques allowing development of such kinds of distributed systems. Here are the most famous:

- Microsoft COM/DCOM;
- CORBA;
- Sun Java beans;

All of them are based on the standard called Remote Procedure Call (RPC), which can be used for developing distributed computational systems. But all those technologies were designed for exchanging data in commercial applications. They ultimately fit for developing the Web applications. In addition, they require that each computational unit should realize the same special protocol. Programming the protocol is a very hard task and in addition, not all programming languages support such technologies. So, our task is to develop the system allowing the formation of the computational process from independent executable modules, which could be run on different computers, for solving a specific task of mathematical modeling.

2. Problem definition

Imagine, we have a set of executables for solving different tasks of computational mathematics (such as approximation of a 2D or a 3D domain: as well as for eigenvalue problems, iterative solutions to algebraic equations, etc.) distributed on a computer network. In this case, we can consecutively run some of our modules by passing to them appropriate input data, which were prepared (calculated) by the previous module. Naturally, we must use some remote terminal protocol (such as telnet or Secure Shell) for running executables and File Transfer Protocol (FTP) for transferring data between computers. But, doing so is a very boring and complicated work. Instead, we could develop a graph, each node being a description of computational process. The description could consist of:

- The name of an executable;
- The IP address of the computer, where the executable is placed;

- The name of the work directory for the executable;
- The name of a file with input data;
- The name and the password of the user on behalf of who the task can be run;
- The description of the format of input data;
- Description of the format of output data.

At the second step, a program is developed, which will pass through a computational graph and run the proper executable on the proper computer with proper data. Of course, there are many other problems, for example, development visual tools for working out a computational graph. But we come to nothing more than the first two tasks.

3. Program realization

First of all, let us consider the types of a computational graph, to be realized:

- It is an oriented one;
- It can have a circuit;
- The graph has only one start node and one end node;
- Each node (excluding the start and the end ones) can have several in branches and several outside of them;
- Each node has an out branch leading to the end node. This means the exit due to the fail.

Realizing such a kind of a graph is very difficult, because we will use some combination of simple graphs. This combination consists of graphs of three levels. An example of the first level graph is shown in Figure 2.

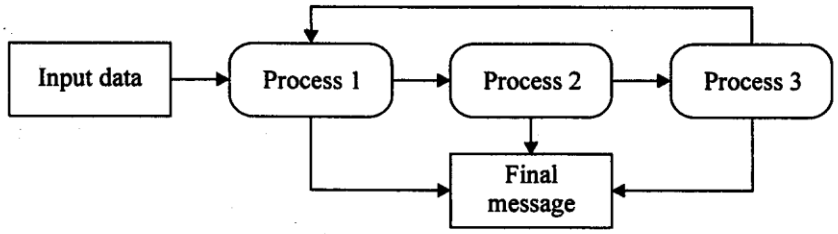

Figure 2

4. "Global" nodes

The main features of such a graph are:

1. It has only one start node and one finish node;
2. Each inner node has up to two in-branches and up to three out-branches (we will call them "global nodes");
3. One of the outer branches necessarily leads to the finish node (this means that we can terminate the modeling at any moment).

The "global" nodes are mainly intended for defining possible paths of a data flow. From a program standpoint this is a container of three objects. Two objects are pointers to other "global" nodes. We will call them the right exit and the left exit. The right exit corresponds to moving forward through a graph. The left exit is a return to some previous node. The third object determines the interfaces between the current "global" node and its in and out nodes. We will call it the interface object.

5. Interface object being a graph as it is

The graph consists of four nodes, which we call "functional" (Figure 3):

1. Preparing data – this functional node is responsible for forming input data for Calculation processes from output data of the previous "global" node;
2. Calculation processes are a set of objects called Calculating unit. These objects are considered below;
3. Gathering data – on this node we make up the output data set from the output data of Calculating units;
4. Analyzing results – at this stage we make the decision, which node will be next: the right exit, the left exit or the finish node.

All the programming work is made for this kind of an object. However before explaining the "core" of this job we should discuss the roles for interfaces between nodes. All such interfaces are based on the XML. When some calculating process is included in the system the format of its input and output data is described in the XML as Document Type Definition. All program modules accomplishing a certain kind of an interface use the DTD files for determining data transformation roles. So, for including some calculating process in the modeling system we must do the following:

- Describe its input and output data in the XML;

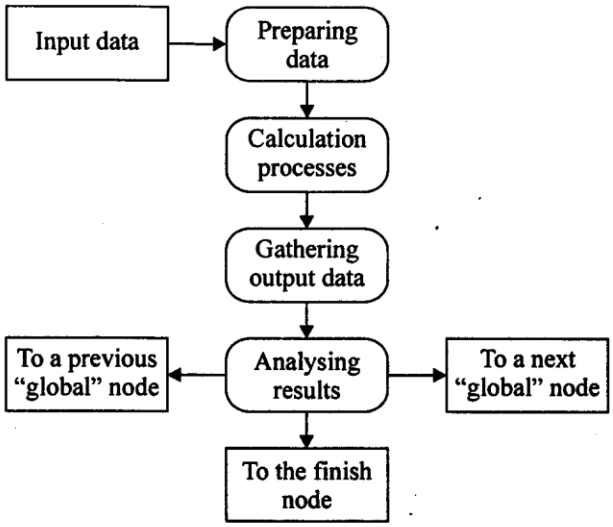

Figure 3

- Develop the program transformation of incoming data in the input data format for the calculating process;
- Develop the program transformation of output data in the XML document;
- Develop the program estimation of the results of calculating process for deciding which exit must be next.

As we can see, for including any calculating module in the system repository one should not make any changes in the module itself. It is very important and useful for using the work results of different members of a scientific group in the big application system. Now we are ready for the discussion of calculating units.

6. Calculating unit

The calculating unit is a set of the information required for running an executable:

- IP address of the computer, where an executable is placed;
- Working directory;
- The name of an executable;
- The name and the password of the user on whose behalf the executable is run;

- The name of a file with input data;
- The name of a file for output data;
- The type of a remote terminal service (Telnet or Secure Shell);
- The port number for the remote terminal service;
- The command for running executable;
- The FTP server port.

This information is sufficient for running any executable on any type of computer with any type of operating system. Of course if our user has resources on the computer.

7. State-of-the-art

Now we have Java classes encapsulating the "global" nodes and calculating units. We have also developed the interface classes for some elementary calculating processes and combined them for solving a thermostatic problem. Our experience proves that resources of a small scientific group can be effectively combined in a powerful system for solution of different kinds of applied tasks.

References

[1] Snider Y. Effective Programming TCP/IP. – SPB–Piter, 2002.

[2] Postel J., Reynolds J. File Transfer Protocol. – USC/Information Sciences Institute, May 1985. – RFC 959.

[3] Tovar. Telnet Extended ASCII Option. – Stanford University-AI, July 1975. – RFC 698.

[4] Crispin M. Telnet Logout Option. – Stanford University-AI, April 1977. – RFC 727.

[5] Postel J., Reynolds J. Telnet Binary Transmission. – USC/Information Sciences Institute, May 1983. – STD 27, RFC 856.

[6] Postel J., Reynolds J. Telnet Echo Option. – USC/Information Sciences Institute, May 1983. – STD 28, RFC 857.

[7] Postel J., Reynolds J. Telnet Suppress Go Ahead Option. – USC/Information Sciences Institute, May 1983. – STD 29, RFC 858.

[8] Postel J., Reynolds J. Telnet Status Option. – USC/Information Sciences Institute, May 1983. – STD 30, RFC 859.

[9] Postel J., Reynolds J. Telnet Timing Mark Option. – USC/Information Sciences Institute, May 1983. – STD 31, RFC 860.

[10] Postel J., Reynolds J. Telnet Extended Options – List Option. – USC/Information Sciences Institute, May 1983. – STD 32, RFC 861.

[11] Waitzman D. Telnet Window Size Option. – BBN STC, October, 1988. – RFC 1073.

[12] Hedrick C. Telnet Terminal Speed Option. – Rutgers University, December 1988. – RFC 1079.

[13] VanBokkelen J. Telnet Terminal Type Option. – FTP Software, Inc., February 1989. – RFC 1091.

[14] Telnet Environment Option / Ed. D. Borman. – Cray Research, Inc., January 1993. – RFC 1408.

[15] Telnet Authentication Option / Ed. D. Borman. – Cray Research, Inc., January 1993. – RFC 1409.

[16] Gololobova S.P., Kozlov V.I. The on-line library of algorithms and program ACCORD // NCC Bulletin. Series Numerical Analysis. – Novosibirsk: NCC Publisher, 2002. – Issue 11. – P. 27–33.