WINDOWS DISTRIBUTED MACHINE (WDM) SYSTEM - THE FIRST STEP TO DISTRIBUTED SIMULATION WORLD

Robert Kobyliński and Andrzej Karbowski

Institute of Control and Computation Engineering
Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
e-mail: rkobylin@ia.pw.edu.pl, A.Karbowski@ia.pw.edu.pl

phone: (+48 22) 660 76 32 **fax**: (+48 22) 825 37 19

KEYWORDS

Computer software, network computing, multiprocessors, distributed processors, combined simulation

ABSTRACT

In this paper[1] we present a software environment for performing distributed computations under the Microsoft Windows family operating systems. It was designed mainly for a parallel implementation of numerical algorithms – both in synchronous and asynchronous version and for a distributed event-driven simulation. The Windows Distributed Machine (WDM) system consists of several components: a console program, two types of demons, a library of functions, and a kernel. They are thoroughly described. The advantages of WDM are presented on the background of the most popular distributed programming tools. Finally two typical applications implemented under WDM – an asynchronous solution of the system of linear equations and a discrete simulation of wormhole routing algorithm on a multiprocessor machine are presented.

INTRODUCTION

At present the computing power of clusters of personal computers, working together in LAN or WAN, may be compared with that of big supercomputers. Distributed numerical calculations and simulation become an ordinary thing. More and more products appear that offer programmers tools, which make distributed programming easier and faster. Current operating systems offer many mechanisms that can be used for distributing of applications over the network. Today the most popular are: Remote Procedure Call (RPC), sockets (Quinn and Shute 1996), Distributed Component Object Model (DCOM) (Pattison 1998). There are also other software tools developed by academia or industry consortia, such as: Parallel Virtual Machine *(PVM),* Message Passing Interface *(MPI), EJB (Enterprise Java Beans) and CORBA (Common Object Request Broker Architecture)*. While some of these tools operate on too low level, for example they do not have the important barrier synchronization mechanism, the other are too general or too advanced. Most numerical applications need only very small part of all procedures and the calculations are performed in clusters of machines of the same type. Now the most popular are 32-bit MS Windows family operating systems. For beginners, like students writing distributed applications running in Windows NT network in an university laboratory it would be also difficult to start appropriate services without Administrator rights.
Another drawback of the mentioned tools is that most of them either do not have or have a very modest, being something additional, graphical interface.

Hence, there is some sense in working out from the very beginning a system, which provides a small, but sufficient for writing most applications, subset of functions, designed for Windows environment, with graphical interface.

WDM PROJECT ASSUMPTIONS

As it was mentioned, Windows Distributed Machine system is dedicated for the Microsoft Windows platform. We took the following project objectives:

- communication should be based on messages, sent between components of the system
- Winsock 1.1 library should be used for the low-level communication.
- programmers should not devote too much time to learn how the system works to start programming

[1] *The work reported herein was supported by KBN grant No 8 T11A 011 15, Microsoft sp. z o. o. and Compaq Computer sp. z o. o.*

- the graphical user interface should be an inherent part of the system;
- the basic aims of the graphical interface are as follows:
  - it should help the user to define the computing systems (machines, paths, programs, etc.) and to start and to stop the application
  - on the operator console some icons representing state of each process during computation should appear
- it should not be important how many computers take part in computations; each component should have the possibility to be run on any machine in the network
- after writing and compiling the program, the application should be obtained by linking with the WDM library
- The WDM library should offer basic functions and procedures for sending and receiving data between processes, corresponding to the most important functions from other well-known systems.

WDM COMPONENTS AND ARCHITECTURE

The WDM system architecture is presented in Fig. 1.

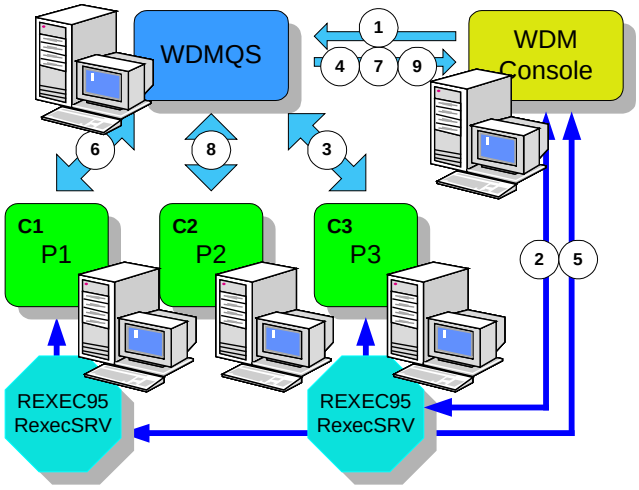

Fig. 1. WDM Components and communication between them

WDM Console, WDM Queue Server and Remote Execute Service are the major components of WDM. They create an "environment" for the distributed application. This application uses WDM library functions for communication. Of course each component can be run under Windows 95, Windows 98, Windows NT 4.0 or Windows 2000.

WDM Console is used to create project (that is to assign processes / programs to computers), remotely start and stop the process, monitor its state, and send keyboard input to a remote application or view the screen of a remote application.

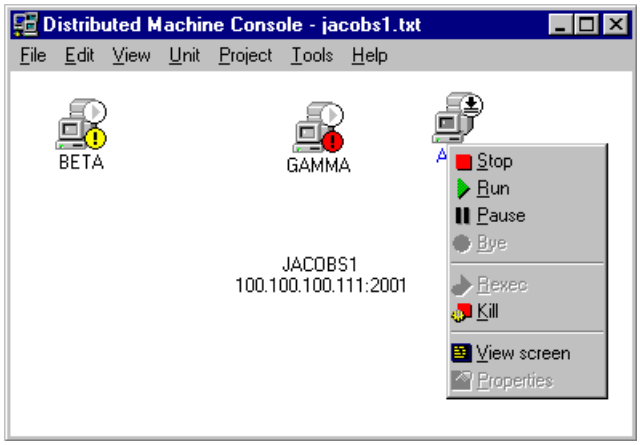

Fig. 2. Windows Distributed Machine Console

WDM Queue Server *(WDMQS)* is the heart of the system. This component is responsible for queuing all messages sent between units. It also sends the requests (like run, stop, pause) from the console to the destination unit. Here the buffers for all units are allocated as well as units screens stored. In order to monitor applications during simulation or computation, WDM Console must be connected to the WDMQS. Thirteen icons in the Console window represent the current state of the application (Fig. 3).

Icon	Current state
	- defining units; the project has not yet been registered
	- the project has been registered, but the unit has not yet been started
	- start pending, the command *start* has been sent to the unit from console
	- a failed attempt to start the unit, an error in communication with the service *Remote Execute Service*
	- the unit has been started, but not registered in WDMQS
	- the active unit *(wdm_UnitRegister was called)*
	- pause pending, the command *pause* was sent from the console to the unit
	- the unit called *wdm_barrier()*
	- the unit is suspended
	- stop pending, the *stop* command was sent to the unit from console
	- the unit finished *(wdm_UnitStop)*
	- the unit is waiting for keyboard input *(wdm_scanf)*
	- the unit is waiting for data from another unit in a blocking mode *(wdm_recv)*

Fig. 3. Icons of the WDMConsole.

Windows NT does not offer a mechanism, which enables running application on remote machines like *rexec* in UNIX. WDM solves this issue using its own services - demons (Miller 1998). Remote Execute Service is an NT service. It must be installed and activated on any machine before remote program execution is possible. Rexec95 is another version of that demon, dedicated for Windows 95 and Windows 98 (Petzold 1996).

WDM LIBRARY

Functions available in the WDM library can be grouped in four categories.

registration:

```
WORD wdm_UnitRegister(int argc, char **argv);
void wdm_UnitUnregister();
void wdm_UnitStop();
```

Using wdm_UnitRegister is mandatory. This way our application establishes connection with WDMQS and receives UID. Information about project and location of WDMQS is passed automatically to the program when remote execution is used. Program can be run manually on any machine but in this case this information must be passed by the operator as command line parameters.

communication and synchronization:

```
int  wdm_recv(int src , char  *ptr);
int  wdm_recv_any(WORD *rsrc, char *ptr);
int  wdm_async_recv(int src , char  *ptr);
int  wdm_async_recv_any(WORD *rsrc, char *ptr);
int  wdm_send(int dst , char  *ptr, int len);
int  wdm_send_over(int dst , char  *ptr, int len);
int  wdm_broadcast(char  *ptr, int len);
int  wdm_broadcast_over(char  *ptr, int len);
void wdm_barrier(int nUID, char *name, ...);
void wdm_barrier_all(char *name);
```

Communication functions *wdm_send()*, *wdm_recv(), wdm_broadcast()* are used to send and receive data from particular unit. Function *wdm_recv_any()* takes the first message from their queue. These functions work in blocking mode. If we do not know whether the data is ready, we can use nonblocking function *wdm_async_recv().* Sometimes data must be sent to all units at the same time. To do that programmers can use *wdm_broadcast().*

input-output:

```
void wdm_printf(char *fmt, ...);
void wdm_scanf(char *s);
```

These functions communicate with the WDMQS. They are very useful when we have no access to the remote computer. From WDM Console we can view remote screen and send keyboard inputs.

other functions:

```
int wdm_rexec( char *host, WORD port,
    char *cmdline, char *username, char *password);
int wdm_kill( char *host, WORD port,
    DWORD PID, char *username, char *password);
void wdm_ParseArgs( int argc, char **argv,
     char *ip, WORD *port, char *project,
     char *name);
```

The programmers could use these functions to communicate with Remote Execute Services working on remote computers. They make it possible to dynamically create or destroy the processes and to deliver their parameters to the program.

Below short sample explains process of creating and running distributed program.

```
#include "ulib.h"
#define ALFA  101
#define BETA  102
#define GAMMA 103
void main(int argc, char **argv)
{
int UID;
int x,y,z;

   UID = wdm_UnitRegister(argc, argv);
   wdm_barrier(3, "DANE", ALFA, BETA, GAMMA);

   switch (UID)
   {
   case ALFA: // unit ALFA code
      x = 10;
      wdm_send(GAMMA, &x, sizeof(x));
      break;
   case BETA: /* unit BETA code
      y = 100;
      wdm_send(GAMMA, &y, sizeof(y));
      break;
   case GAMMA: // unit GAMMA code
      wdm_recv(ALFA, &x);
      wdm_recv(BETA, &y);
      z = x + y;
      wdm_printf("z = %d\n",z);
      break;
   }
   wdm_UnitStop();
}
```

In this application we define 2 processes (ALFA, BETA) responsible for calculating two values and sending them to the master process GAMMA. This process collects information form its neighbors and prints the result on the virtual screen. As you can notice, there is only one program for all units. During registration function wdm_UnitRegister() communicates with WDM Queue Server and receives UIDs. Since now each of them knows which part of the code to execute. wdm_barrier() helps to synchronize computation. None of the processes will go forward before each of them reaches this place of the code. Using this function is not actually necessary in this sample, because data sent to GAMMA is buffered in WDMQS before delivery. Sequence of running processes does not matter in this case, because wdm_recv() is a blocking function. GAMMA process will wait in this place for the requested data to be available in

WDMQS. Since we have our application ready, all we need is to assign each process to the computer and start them using WDM Console (Fig. 2).

CASE STUDY 1: SOLUTION THE OF SYSTEMS OF LINEAR EQUATIONS

We want to solve the system of linear equations:

$$Ax = b$$

for a given nonsingular matrix A *of size* $n \times n$ *and a given vector* $b \in R^n$ *, looking for a vector* $x \in R^n$ *.*

We assume, the matrix A *is diagonally dominated (that is* $\forall i \ \ |a_{ii}| > \sum_{j \neq i} |a_{ij}|$*). In this case (Bertsekas and Tsitsiklis 1989) the solution may be obtained by an iterative method:*

$$x := x - D \cdot (Ax - b)$$

where D *is a diagonal matrix, such that* $d_{ii} = \frac{1}{a_{ii}}$*.*

The vector x *may be partitioned and its parts allocated to different units (processors / processes / computers) in an arbitrary way.*
Let us assume, that there are p *units, that is*

$$x = [x_1, x_2, ..., x_p]^T$$

where $x_i \in R^{n_i}$, $\sum_{i=1}^{p} n_i = n$*. Then, the* i *- th unit modifies only subvector* x_i *by the iteration:*

$$x_i := f_i(x_1, x_2, ..., x_{i-1}, x_i, x_{i+1}, ..., x_p)$$

obtaining subvectors $x_j, j \neq i$ *from other units.*

We used WDM to implement this algorithm and compute all subvectors on different machines in MASTER-WORKER model. In this model, the master program must decompose matrix A and distribute it to the WORKERS. In asynchronous version of this method, WORKERS do not depend on each other and they can use maximum speed of their processors. They perform as many iterations as needed to reach local stop criteria. The results are sent to the MASTER and new data is requested. MASTER collects results from all WORKERS and checks the global stop criteria. When global stop criteria is reached, MASTER stops all WORKERS.
Below we present the major procedures of both programs.

```
void worker()
{
   int msg, dif, i=0;
   do {
      printf(".");i++;
      wdm_recv(MASTER, (char *)&msg);
      if (msg==STOP_UNIT)
         break;
      wdm_recv(MASTER, (char *)&x);
      iteration();
      dif = xcompare(delta_x);
      msg = UPDATE;
      wdm_send(MASTER, (char *)&msg, sizeof(msg));
      wdm_send(MASTER, (char *)&beg, sizeof(beg));
      wdm_send(MASTER, (char *)&end, sizeof(end));
      wdm_send(MASTER, (char *)&dif, sizeof(dif));
      wdm_send(MASTER, (char *)&i,   sizeof(i));
      wdm_send(MASTER, (char *)&xl[beg],
               (end-beg+1)*sizeof(double));
   } while (1);
   printf("\n");
   for (i=beg;i<=end;i++)
      wdm_printf("%3d %+10.6f \n",i,xl[i]);
}

void master()
{
   int active;
   int beg,end,dif,iter;
   WORD xuid;
   int msg;

   local_stop = (int *)malloc(p*sizeof(int));
   memset(local_stop,0,sizeof(int)*p);
   active = p;

   msg = NEW_X;
   wdm_broadcast((char *)&msg, sizeof(msg);
   wdm_broadcast((char *)&xo[0],
                 size*sizeof(double));


   do {
      wdm_recv_any(&xuid, (char *) &msg);
      wdm_recv(xuid, (char *) &beg);
      wdm_recv(xuid, (char *) &end);
      wdm_recv(xuid, (char *) &dif);
      wdm_recv(xuid, (char *) &iter);
      wdm_recv(xuid, (char *) &xo[beg]);

      local_stop[xuid-UBASE]=dif;
      if (total_stop()==0) {
         msg = STOP_UNIT;
         wdm_send(xuid, (char *)&msg, sizeof(msg));
         active--;
      }
      else
      {
         msg = NEW_X;
         wdm_send(xuid, (char *)&msg, sizeof(msg));
         wdm_send(xuid, (char *)&xo[0],
                  (size)*sizeof(double));
      }
   } while (active);
   free(local_stop);
}
```

CASE STUDY 2: WORMHOLE ROUTING ALGORITHM

Multiprocessor machines with local memory have many processors working independently. Each of them has its own memory and can access data located only in this memory. Information exchange between processors is based on messages. The scalability of such machines is better than machines with global memory, where access to this common memory is a bottleneck. On the other hand, sending

messages between processors in machines with local memory is not an easy thing and depends heavily on the topology of the system. At present the most popular is mesh topology - one of the simplest (Fig. 4).

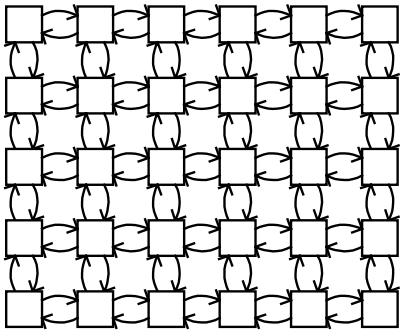

Fig. 4. $m \times n$ **processors mesh**

In early machines packets were sent from node to node (*store and forward).* Transmission delay of such model is:

$$\Delta t_{s\&f} = \frac{L}{B} D$$

where L - packet length, B - channel bandwidth, D - distance between two processors (that is the number of links between processors). At the beginning of 90's new technique of reducing transmission time has been proposed. It is called *wormhole routing* (Lionel *et al.*1993). In this method each packet is split into a number of very small pieces - flits (Flow Control Digit). The leading flit clears the way for the rest. Other flits follow immediately the first one. They are being sent from one processor to another according to a routing algorithm. Flits are stored in small buffers in the mesh nodes. If the required channel is being used by other packet, they must wait till the channel is released.
Transmission delay of such model is:

$$\Delta t_{whr} = \frac{L_f}{B} D + \frac{L}{B} = \frac{1}{B}\left(L_f D + L\right)$$

where L_f flit length. Because $L_f D << L$ (flits are very small) we can transform it to the following:

$$\Delta t_{whr} = \frac{1}{B} L$$

As it is seen, the distance between end points has no impact on the packet transmission time.
The problem is that wrong routing algorithm may cause deadlock, which is shown in Fig. 5.

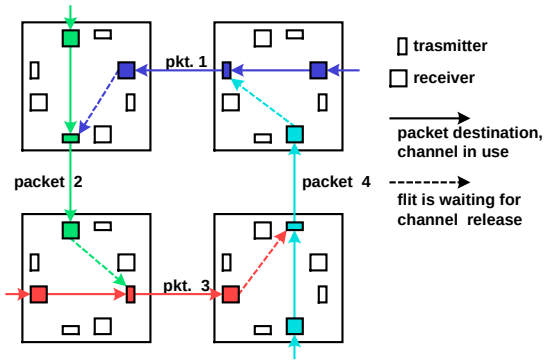

Fig. 5. Deadlock in wormhole routing

WDM system has been used to simulate such model and compare two routing algorithms.

Algorithm 1 – XY routing

We assign two numbers (x, y) *(position) to every node in* $m \times n$ *mesh. In XY routing algorithm flits go first in* x *direction (until they reach the abscissa of the destination node) and then in* y *direction. Denoting by* (x, y) *current node position, by* (x_d, y_d) *the destination node position and by* (x', y') *the next node position, we can describe this algorithm by formula:*

if $x_d \neq x$ *then*

$$(x', y') = \begin{cases} (x-1, y) & \text{when } x_d < x \\ (x+1, y) & \text{when } x_d > x \end{cases}$$

else

$$(x', y') = \begin{cases} (x, y-1) & \text{when } y_d < y \\ (x, y+1) & \text{when } y_d > y \end{cases}$$

Three packets transmitted due to XY routing algorithm are presented in Fig. 6.

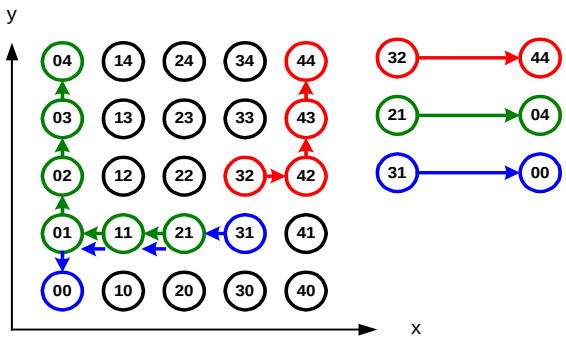

Fig. 6. XY routing algorithm

Algorithm 2

We enumerate nodes of $m \times n$ *mesh according to the rule (Fleury and Fraigniaud 1994)*

$$\text{if } u=(x,y) \text{ then } L(x,y)=\begin{cases} y*m+x & \text{when } y \text{ is even} \\ y*m+m-x-1 & \text{when } y \text{ is odd} \end{cases}$$

where x, y *are node's coordinates. Such mesh is shown in Fig. 7. Now we can define the routing method: if* u *is our current node, and* v *is the destination node, than routing function* $R(u,v)=w$ *is:*

$$L(w)=\begin{cases} \max_z \{L(z) \mid L(z) \le L(v) \ \ z \text{ neighbour of node } u\} & \text{when } L(u)<L(v) \\ \min_z \{L(z) \mid L(z) \ge L(v) \ \ z \text{ neighbour of node } u\} & \text{when } L(u)>L(v) \end{cases}$$

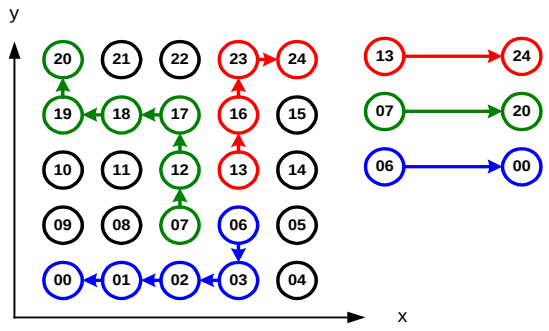

Fig. 7. Wormhole routing algorithm No. 2

We have simulated 5x5 mesh (Fig. 8), using a event-driven synchronous simulation. We decided to write two programs: one for units and one (master) sending requests, collecting data and synchronizing simulation. Master unit starts all 25 units on remote machines using *wdm_rexec()* function. Unit obtains its UID using *wdm_UnitRegister()* and calculates the node position. Having the position, it automatically knows its neighbors and simulation can start. Master unit generates traffic reading data from file. Each unit simulates routing process and sends local time stamp to the neighbors and master unit. Then the global simulation time T is calculated using formula: $T=\min(t_1,t_2,...,t_n)$ by the master unit. The time stamp is sent to all units and they can simulate up to time T. We used null messages to avoid situation when the local time cannot be calculated because no event has occurred on one of the links.

Below we present main function of unit program, just to show WDM library usage.

```
int main(int argc, char **argv)
{
   BOOL  STOP_NODE = FALSE;

   myUID = (WORD )wdm_UnitRegister(argc, argv);

   wdm_barrier_all("START"); // wait for the rest
   wdm_recv(MASTER, (char *)&M);
   wdm_recv(MASTER, (char *)&N);
   wdm_recv(MASTER, (char *)&ROUTE);
   InitLocalNodes(myUID); // define neighbors
   PrintLocalNodes();

   wdm_barrier_all("START2"); // all nodes ready

   do {
      T = NewT;
      printf("NewT = %d.\n",NewT);
      Check_Flit_Generator();
      Send_All_Flits();
      Recv_All_Flits();

      Send_Local_Time();
      wdm_recv(MASTER,(char*)tmp);//synchronization
      NewT = tmp->t;

      if (tmp->msg == COMPLETED) break;  // Stop
      if (tmp->msg == GENERATE)
          InitGenerator(tmp);
      Send_All_ACK();
      Recv_All_ACK();
   } while (1);

   wdm_printf("*** COMPLETED ***\n");
   wdm_UnitStop();
   return  0;
}
```

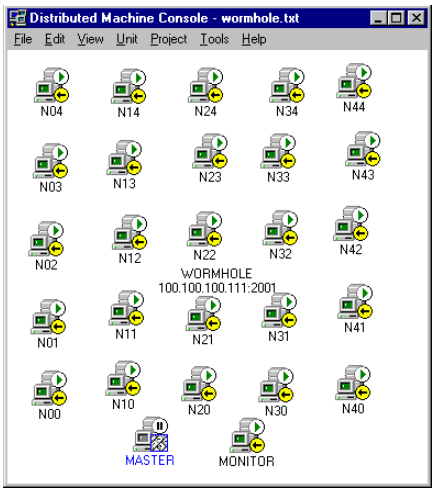

Fig. 8. Wormhole routing simulation

CONCLUSIONS

In many cases, distributed simulation or computations can be more effective and more natural than sequential programming. Windows Distributed Machine can be found as the very easy and useful tool for creating such applications. Programmers do not need to care on the details any more and can focus on computations of the simulated process. However some futures offered by WDM can be found in other well-known systems, there are several completely new ones: remote screen preview and sending keyboard inputs. The graphic user interface helps to monitor application.

WDM was successfully tested on two problems of different type: distributed solution of a set of linear equations and distributed event-driven simulation.
WDM system has been also used as an auxiliary tool to the class „Introduction to Parallel and Distributed Computations" at our university. It turned out, that many students preferred it than standard packages: PVM, MPI, Corba, etc., due to simplicity of use in their standard Windows environment and very small size. The binary files have only 1.2 MB. All programs and wdm library were compiled using Microsoft Visual Studio 6.0 Enterprise Edition.

WDM can be downloaded from:
http://www.ia.pw.edu.pl/~karbowsk/wdm
The distribution files with documentation in postscript after compression has only 1.3M.

REFERENCES

Bertsekas, D.P. and J.N. Tsitsiklis. 1989. *"Parallel and Distributed Computation: Numerical Methods"*, Prentice-Hall, Englewood Cliffs, N.J..

Davis, R. 1994 *"Windows NT Network Programming"*, Addison-Wesley, Reading, Massachusetts.

Fleury, E. and P. Fraigniaud. 1994. "Strategies for multicasting in meshes", Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, Research Report No. 94-07

Lionel, M; D. Ni; and P. K. McKinley. 1993, "A Survey of Wormhole Routing Techniques in Direct Networks", *IEEE Computer*, vol. 26. No. 2, pp. 62-76,

Miller, K. 1998. *"Professional NT Services"*, Wrox Press Ltd., Chicago, Illinois.

Pattison, T. 1998. *"Programming Distributed Application with COM and Microsoft Visual Basic 6.0"*, Microsoft Press, Redmond, Washington.

Petzold, C. 1996. *"Programming Windows 95"*, Microsoft Press, Redmond, Washington.

Quinn, B. and D. Shute. 1996. *"Windows Sockets Network Programming"*, Addison-Wesley, Reading, Massachusetts.

BIOGRAPHY

Robert Kobyliński was born in Skarżysko-Kamienna, Poland, in 1970. He received the M.S. degree in computer science from Warsaw University of Technology, Faculty of Electronics and Information Technologies, in 1999. He is a Microsoft Certified System Engineer and Microsoft Certified Professional + Internet. In years 1994-1996 he was working for **Accord Development** as Novell NetWare Network Administrator, then in 1997-1998 he was working for **XEROX** Poland as IT System Analyst and Network Administrator. Currently he works as Microsoft Consultant for **COMPAQ** (previously **Digital Equipment**), in Warsaw. He is responsible for supporting and implementing Microsoft Systems and environments for the biggest companies in Poland.

Andrzej Karbowski was born in Augustów, Poland, in 1958. He received M.S. degree in electronic engineering, specialization automatic control from Warsaw University of Technology, Faculty of Electronics, in 1983. He received Ph.D. in 1990 in automatic control and robotics for thesis "Structures and Mechanisms for Control of Multireservoir Systems during Flood". His scientific interests concentrate on optimal control in risk conditions, decomposition and parallel implementation of numerical algorithms, global optimization, tools for complex systems management and control. His results were described in about 10 articles published in international journals and about 40 conference papers. He works as assistant professor (adjunct) at Faculty of Electronics and Information Technologies of Warsaw University of Technology. For students of this faculty he gives lecture "Introduction to Parallel and Distributed Computations". Other teaching activities concern theory and implementation of control and decision support systems.
