Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

CACI NETWORK II.5 ®

A COMPUTER AND COMMUNICATIONS NETWORK SIMULATOR

William J. Garrison
CACI, Inc.
501 Office Center Drive
Fort Washington, PA 19034

ABSTRACT

NETWORK II.5 is a modelling tool that takes a user-specified computer system description and provides measures of hardware utilization, software execution and conflicts. It allows a user to quickly look at various computer system configurations and determine which one can best handle the required workload. It can model a wide variety of computer systems; from a single CPU system to a vast network of computers. It can model the portions of a design which are of special interest at a very fine level of detail while the rest of the system is modelled at a much higher level. It runs on most major computer systems and can be used after a short period of training.

INTRODUCTION

Many good general purpose simulation languages are available to today's simulation practitioner. However, even with these aids, it can still be difficult to convince some project managers of the importance of simulation. Simulation should not be something that you use only as a last resort or because there is a contract requirement to do so. The reasons I most often hear cited for not simulating are cost, timeliness of results, and the design "never stays the same as the one simulated." Notice that the recurring theme is, either directly or indirectly, cost. Timeliness concerns can mean the cost of putting enough of the right people on the job is "too high." Outdated design concerns often means that the cost of keeping model updated is "too high." Few doubt that simulation will have a beneficial effect on the resulting design. Therefore, the best way to get more projects to use simulation would seem to be to find a way of reducing the cost of simulation.

There are few things as popular with project managers as reducing cost. The question is how? I propose that the time has come to offer more of what I call "targeted" simulation packages which allow a user to quickly specify how their system compares to some "average" and produces results. I call these packages "targeted" because they are too specific to be called general purpose and too general to be called custom. As a rough guide, where a general purpose language might include 90% of the simulation community as a part of its intended audience, and a custom package might aim for less than 1%, a targeted package might aim for 5%. There still will be a need in this world for custom simulations, because some situations are too specialized to fall into any general category. However, many general classes of problems do exist that can be satisfied with appropriately targeted packages. Few organizations would consider developing "custom" software to do their accounting, so why should they have to develop custom simulations?

An additional benefit of using a targeted simulation package is that the simulator can be an engineer from the regular product design team, instead of being a simulation expert brought in from the outside. The closer a simulation is to the people responsible for a design, the easier it will be for them to "feel good" about a simulation and therefore put their reputation on the line by using simulation results as the basis of important design decisions. Also, the accuracy of the simulation results won't be hurt by having people close to the design being close to the simulation. The chance for keeping a simulation updated to reflect all design changes will also improve when someone from the "outside" (even if "outside" means just down the hall) does not have to be called every time there is a design change.

Targeted packages reduce cost because the development cost of the model is spread over many users. The results are considerably more timely because the user is filling in the blanks into an existing structure instead of designing a new one. The design simulated can stay current because the underlying structure of the model is general enough to facilitate changes. It is very difficult to explain to the user of a custom simulation model why some seemingly trivial change to the model's operating logic will require a major restructuring of the model. This happens because every model needs some underlying assumptions, and whenever someone wants to change something relating to these assumptions, watch out!

THE COMPUTER AND COMMUNICATIONS NETWORK CLASS

The "Computer and Communications Network" class of problems consists of problems in which devices are requesting, manipulating and distributing information and making decisions based on the system state. NETWORK II.5 is designed to serve this class of problems. Telephone networks fit into this class of problem. Distributed database systems fit into this class. Local networks of computers fit into this class of problem. Many other real world applications fit into this class of problem.

SETTING PROJECT GOALS

Now that we have defined the class, how should we determine what kind of capabilities a tool that serves this class will need? The goal is to provide all of the capabilities that are needed, but no extras. Extra features can "clog up" a good design by adding useless complexity and increasing run time costs.

The approach used to develop NETWORK II.5 could probably best be described as the "MATH 101 Geometric Proof" approach. In this approach, you start with the answer and work back to the question. (At least that's the way I did it back then.) In general, the

potential users of a targeted simulation package are much more specific about the answers they want from the package than about the information they can provide to the package. So, in NETWORK II.5, we started the project by designing the reports. When we had a set of reports that filled our user's needs, we worked backwards and built a tool that produced those reports. When we had the simulation tool, we wrote another tool (which we called NETIN) to help the user produce the inputs needed by the model.

Obviously, the above description of our approach is a simplification. We learned more about the application area as we went along and went back and expanded some of the reports and added model features during NETWORK II.5's development. However, we kept our focus on answers and tried to define the minimum input data required to produce those answers accurately.

DON'T FORGET THE USER

No targeted simulation package can expect to find widespread acceptance unless it is user friendly. A user friendly simulation should provide some kind of interactive guidance when it comes time to build a simulation description and should run the model interactively so that the user doesn't feel left out of the action. Never underestimate the size of the undertaking when you declare that you are going to make a model "user friendly." The interactive front end to NETWORK, called NETIN, is actually larger than the model it supports (from a source code line count point of view). Never underestimate the range of choices that the user may wish to explore when interacting with an unfamiliar program. Also, to murder a quote from P. T. Barnum, no one ever went broke overestimating the reluctance of a user to open any document labeled "User's Manual." On-line documentation is a must (and is built into NETWORK II.5).

IMPLEMENTATION

Making a targeted simulation package available for a wide variety of host machines will greatly increase its chances for use. Writing the package in an existing powerful general purpose high level simulation language will reduce the cost of producing the package and make package enhancement easier. NETWORK II.5 was written in its entirety in the high level simulation language SIMSCRIPT II.5 (2). This includes the interactive front end, the modelling portion and the plotting portion. Because SIMSCRIPT II.5 is available for many different host machines and because the implementations for these machines is carefully controlled, NETWORK II.5 can be offered on many different mainframes. It is currently offered on CDC, IBM and VAX machines with implementations for other machines to be added as required.

NETWORK II.5 DESIGN

A fundamental design decision made in NETWORK II.5 was that because it would be impossible to build every hardware device that a user might wish to model into NETWORK II.5 we weren't even going to try. Instead, we chose to provide the user with general purpose building blocks which can be easily specified and combined to form the desired system devices. Careful examination of the problem area suggested that there were three basic hardware functions that had to be modelled; data was being processed, data was being moved and data was being stored. These functions are modelled by what we call the Processing Element, Data Transfer Device and Data Storage Device building blocks.

HARDWARE

Processing Elements are the only active hardware devices in a NETWORK II.5 simulation. Only Processing Elements can execute instructions. Each Processing Element has its own instruction repertoire which contains the description of every instruction this Processing Element can execute. Each instruction is referenced by its name and instruction definitions are local to a particular Processing Element. Therefore, the same name instruction could execute differently on different Processing Elements. Processing Elements are used to represent bus controllers, sensors, human operators at a terminal or any other devices that are more than just a data source or link.

Data Transfer Devices are the links between Processing Elements and Data Storage Devices. They also can carry data between two Processing Elements. They can be connected to as many Processing Elements and Data Transfer Devices as needed. Because this is a simulation, the actual physical implementation of the Data Transfer Device is immaterial. The Data Transfer Device could be modelling a bus, a satellite communications link, a microwave link, etc. The only characteristic of the Data Transfer Device (other than connectivity) that is significant to NETWORK II.5 is the timing of the data transfer. Each Data Transfer Device has a user-defined specification listing the bus speed and the amount of overhead to add for each transmission. Data Transfer Devices always break transmissions into what are called words (groups of bits) and blocks (groups of words). This allows the user to model up to two levels of bus protocol by means of a user specified word overhead time and block overhead time. Therefore, changes in bus protocol can be easily modelled by simple changes to the Data Transfer Device's definition instead of having to modify a characteristic of each message and file sent. For example, a packet type network could be modelled with word overhead time modelling the parity bits and block overhead time representing the packet header and trailer. Every time a message was sent, NETWORK II.5 would automatically handle breaking the transfer into proper size packets and adding the appropriate overhead.

Data Storage Devices are simply places where data are stored. Disk, drum, core and semiconductor memory all can be modelled by this building block. Data Storage Devices contain user named files and have a capacity measured in bits. The user specifies how many Processing Elements may use this device simultaneously and NETWORK II.5 will automatically handle the queuing of additional requests. The Data Storage Device also contains a realistic file structure where the user can dynamically create, modify and destroy named files. NETWORK II.5 automatically keeps track of the file structure so that conditions such as attempting to read a nonexistent file or overflowing the Data Storage Device are caught and diagnosed. For simulations where a file structure is either undetermined or not significant, there is a special case of writing information to the general storage area of a Data Storage Device, where an aggregate bit count is kept instead of a specific file structure. NETWORK II.5 doesn't care about the way the Data Storage Device is implemented except for how it relates to timing and size. Analagous to the Data Transfer Device, a Data Storage Device breaks all reads and writes into words and blocks. Different methods of storing data can be modelled by careful definition of word and block overhead times. For example, to model a disk as a Data Storage Device, you would add in any parity bits to

the word overhead time and add the seek time for a sector in as a block overhead time.

SOFTWARE

The software of the simulated system is presented to NETWORK II.5 in the form of software modules. Each module contains a specification of what Processing Elements are allowed to execute this module, when this module may run, what the module is to do when running, and what other modules (if any) to start upon completion. Modules also may be specified with a priority which will be used for resolving contention and determining whether a module should interrupt another.

There are a number of ways that a module dynamically chooses its host Processing Element. A module may be defined to run on a specific Processing Element or be provided with a list of potential host Processing Elements. If a module can run on more than one Processing Element, the module may be defined to allow copies of the same module run simultaneously on different Processing Elements.

NETWORK II.5 provides the user with the ability to specify many different kinds of conditions that must be met before a module can start execution. These module preconditions include time, hardware, message and semaphore based preconditions. Time based preconditions include starting a module at a specific time and/or iterating at a specific rate. Hardware based preconditions cause a module to wait until a particular set of Processing Element(s), Data Transfer Device(s) and/or Data Storage Device(s) are available at the same time. Message and semaphore based preconditions cause a module to wait until a given set of messages are received and/or a list of semaphores have the proper value. Semaphores can also be used to cancel the execution of modules.

HARDWARE AND SOFTWARE

Modules and Processing Elements interact through the use of instruction names. A module knows the tasks it will perform solely by the names of the instructions required to accomplish that task. A processing Element knows how to execute an instruction when given its name by a module. When a Processing Element tells a module that it is available for work, the module gives the Processing Element the name of an instruction and the Processing Element executes the instruction. When the instruction completes, the module is notified and will then either issue another instruction to the Processing Element or tell the Processing Element that it is done. Modules contain the basic operating logic of the simulation. Without a module, a Processing Element will do nothing because it doesn't know which instruction to execute. This module - Processing Element independence allows the same module to run differently on different Processing Elements. For example, if a module were assigned to special purpose hardware whenever it was available, it will run faster when on that hardware instead of a general purpose processor. Also, this independence allows easy reconfiguration of a system in response to a fault.

INSTRUCTIONS

Earlier, I claimed that NETWORK II.5 can model a system at varying levels of detail. Here's how it's done. An instruction in NETWORK II.5 is not meant to literally be an instruction in the instruction set of the machine being simulated (although it may be). Instead, it is more likely to be a macro instruction. Since NETWORK II.5 is designed to simulate the effect of an instruction on the system state, as opposed to computing a numerical result, macro instructions can be nestled in with assembler instructions. For example, a module could implement a Fast Fourier Transform by actually going through the micro instructions needed to compute the FFT or a macro instruction called FFT could be implemented which took the required number of machine cycles in one gulp.

There are four kinds of instruction building blocks in NETWORK II.5. These four building blocks perform four different functions; processing, sending messages, read/write files and set/reset semaphores. Processing instructions simply tie up a Processing Element for a given number of cycles. Message instructions send a message to another Processing Element using a bus. The receipt of that message may trigger a module at the receiving Processing Element immediately or queue up and trigger a module when other preconditions are met. Read/write instructions simply move data to or from a Data Storage Device and may dynamically change the size or location of the file stored. Semaphores are global bit flags that any module can check. Semaphore instructions may either set or reset semaphores.

GETTING ANSWERS

People simulate to get answers. The way they get answers from NETWORK II.5 is from the many reports provided. There are seven basic categories of reports provided by NETWORK II.5. They are Module Summary, Processing Element Statistics, Data Transfer Device Statistics, Data Storage Device Statistics, a Narrative Trace, a Snapshot Report, and a Timeline Report.

The Module Summary, Processing Element Statistics, Data Transfer Device Statistics and Data Storage Device Statistics reports are tabular in format and are produced at user specified times and at the end of the simulation to summarize the simulation results. The Narrative Trace report is produced interactively upon demand and chronicles the progress of the simulation event by event as they occur. This report is interactive with the user to allow the user to stop a simulation if things are going wrong and because a narrative report will be very large by its very nature and so the user should be allowed to select only those pieces that will be of use. The Snapshot Report lists the current status of every hardware device, module and semaphore in the simulation and is produced both as a part of the tabular reports and interactively during a simulation run in response to a user request. The Timeline Report is a post processed report that acts upon a database produced during a simulation run to show the status of every hardware device and every semaphore in the simulation along a time axis. The time span plotted on this report is user specified so that a user can go back and expand the time scale of a period of interest several times until the needed information is obtained.

EXAMPLE

For the purpose of illustration, an extremely simplified example of using NETWORK II.5 is presented here. See reference 1 for an example of a more realistic problem both presented and solved.

An office contains two computers on the same serial bus. Files are sent from one computer to another. Only one computer can be using the bus at a time. Computer A requests a file called "Data" from Computer B every 30 seconds. Computer B requests a file called "More Data" from Computer A every 45 seconds. "Data"

is 700 bits long. "More Data" is 770 bits long. The serial bus moves data at 1200 baud. Assume the files are stored in a 7 bit code and the hardware adds a parity bit to each character transmitted. Run the simulation of this system for 1000 seconds.

PROBLEM FORMULATION

Hardware - Each Processing Element will need two instructions: A "Request file from other PE" instruction and a "Send file to other PE" instruction. Memories need not be simulated (but they certainly could be). Processor internal speed is not significant to this simulation. Bus speed and width are significant to the simulation.

Software - Each processing element will need a module that sends a file upon the receipt of a request and a module that requests a file every X seconds.

The user prepares the input file to NETWORK through the use of the interactive program NETIN. NETIN guides the user through building a description of the system to be simulated and provides on-line documentation. A brief example of a NETIN session is presented as Figure 1.

In general, when in NETIN a user can find out what to do next or receive a further explanation of a prompt by typing "?". Also, when the user becomes an "expert," a BRIEF level of prompting may be chosen which significantly shortens the dialog. The output of NETIN is a file which becomes the input to NETWORK. The file is easily readable and useful for documenting the simulation performed. The sample portions of the file produced by NETIN to solve this example problem are given as Figure 2.

Running the simulation involves an interactive dialog with NETWORK, an example of which is included as Figure 3. As a reusult of running this data file, the user gets the end of simulation reports included as Figures 4, 5, 6 and 7. The user also could request a timeline plot (included as Figure 8) and narrative trace reports (included as Figure 9).

1. CACI, Inc., NETWORK II.5 USER'S MANUAL, Version 1.1, March 1984.

2. Russell, E. C., BUILDING SIMULATION MODELS WITH SIMSCRIPT II.5, CACI, Inc., January 1983.

```
07.12.42 >netin
WELCOME TO THE COMPUTER NETWORK SIMULATION INPUT PROGRAM
NETIN
ENTER A NETIN TOP-LEVEL COMMAND
>?
NETIN TOP-LEVEL COMMANDS AND THEIR USAGE
-----------------------------------------
"?"        ("?")   LISTS THE AVAILABLE TOP-LEVEL COMMANDS
"HELP"     ("H")   PROVIDES BRIEF EXPLANATION OF COMMANDS
"PROMPT"   ("P")   ALLOWS YOU TO SET THE LEVEL OF PROMPTING
"LOAD"     ("LO")  ALLOWS YOU TO LOAD DATA FROM A FILE
"VERIFY"   ("V")   ALLOWS YOU TO CHECK CURRENT DATA
"SAVE"     ("S")   ALLOWS YOU TO WRITE TO A FILE
"DISPLAY"  ("DI")  ALLOWS YOU TO DISPLAY A LIST OF BASIC ENTITIES WHICH ARE IN CORE
"FIND"     ("F")   LOCATES A SPECIFIED BASIC ENTITY AND ENTERS THE MANIPULATION MODE
"CREATE"   ("CR")  ALLOWS YOU TO ENTER DATA WHICH DESCRIBES A NEW BASIC ENTITY
"DELETE"   ("DE")  ALLOWS YOU TO DELETE AN ENTITY
"QUIT"     ("Q")   TERMINATES PROGRAM EXECUTION

ENTER A NETIN TOP-LEVEL COMMAND
>create ?
ENTER ONE OF THE BASIC ENTITY TYPES LISTED BELOW:
    PROCESSING.ELEMENT OR PE
    BUS                OR B
    STORAGE.DEVICE     OR SD
    MODULE             OR M
    INSTRUCTION.MIX    OR IM
    FILE               OR F
>pe
YOU WILL BE ASKED FOR THE DATA TO MAKE A NEW PROCESSING.ELEMENT
NAME (TEXT)
>computer a
BASIC.CYCLE.TIME (REAL; MICROSEC)
>30
INPUT.CONTROLLER (TEXT; YES/NO)
>pd
THE DEFAULT VALUE IS "NO"
ENTER A "YES" OR A "NO" OR ENTER "D" FOR "DEFAULT".
>d
THE DEFAULT, "NO", HAS BEEN ASSUMED.
```

Figure 1

```
* THE 1984 WINTER SIMULATION CONFERENCE EXAMPLE
***** PROCESSING ELEMENTS - SYS.PE.SET
 HARDWARE TYPE = PROCESSING
  NAME = COMPUTER A
     BASIC CYCLE TIME =      1.000 MICROSEC
     INPUT CONTROLLER = NO
     INSTRUCTION REPERTOIRE =
        INSTRUCTION TYPE = MESSAGE
           NAME ; ASK FOR "DATA" FILE
              LENGTH ;      0 BITS
              DESTINATION PROCESSOR ; COMPUTER B
           NAME ; SEND "MORE DATA" FILE
              MESSAGE ; SEND "MORE DATA" FILE
              LENGTH ;    770 BITS
              DESTINATION PROCESSOR ; COMPUTER B
***** BUSSES - SYS.BUS.SET
 HARDWARE TYPE = DATA TRANSFER
  NAME = BUS
     CYCLE TIME =     833.00 MICROSEC
    BITS PER CYCLE =       1
    CYCLES PER WORD =      7
    WORDS PER BLOCK =      1
    WORD OVERHEAD TIME =   833.00 MICROSEC
    BLOCK OVERHEAD TIME =     0.0  MICROSEC
*****MODULES - SYS.MODULE.SET
 SOFTWARE TYPE = MODULE
  NAME = REQUEST "DATA" FILE
     PRIORITY =       0
     INTERRUPTABILITY FLAG = NO
     CONCURRENT EXECUTION = NO
     ITERATION PERIOD = 3.00E+07 MICROSEC
     ALLOWED PROCESSORS =
        COMPUTER A
     INSTRUCTION LIST =
       EXECUTE A TOTAL OF ;       1 ASK FOR "DATA" FILE
```

Figure 2

```
>network
ENTER NAME OF INPUT FILE
>wsc84
SOURCE MODEL IS FROM WSC84 NETWORK P FILE.
OUTPUT WILL GO TO WSC84 LISTING P FILE.
PLOT INPUT DATA WILL GO TO WSC84 PLOTIN P FILE.

WELCOME TO CACI NETWORK II.5.
YOU ARE USING VERSION 1.1, RELEASED ON 2/1/84.

DO YOU WANT A LISTING OF YOUR INPUT FILE (OFFLINE)? (Y/N)
>y
DO YOU WANT A LISTING OF YOUR INPUT DATA
WITH DEFAULT VALUES FILLED IN? (Y/N)
>y
  NOTE:  NO FATAL INPUT ERRORS HAVE BEEN DETECTED.

ENTER THE TIME UNIT WHICH YOU WISH TO EMPLOY FOR INPUT.
IT MUST BE SECONDS(S), MILLISECONDS(MIL), OR MICROSECONDS(MIC).
>s
ENTER LENGTH OF SIMULATION (IN SECONDS)
 1000
DO YOU WISH TO HAVE PERIODIC REPORTING? (Y/N)
>y
ENTER TIME FOR FIRST REPORT (IN SECONDS)
>500
ENTER THE PERIOD FOR REPORTING (IN SECONDS)
>100

DO YOU WISH TO TRACE THE EVENT FLOW? (Y/N)
>n
SIMULATION TERMINATED AT       10.000000 SECONDS.
```

Figure 3

CACI NETWORK II.5 VERSION 1.1 2/1/84 07:52:17
THE 1984 WINTER SIMULATION CONFERENCE EXAMPLE

PROCESSOR ELEMENT UTILIZATION STATISTICS
TO SIMULATED TIME 1000.0000 SECONDS

PROCESSOR NAME	COMPUTER A	COMPUTER B
NO. OF BUS REQUESTS	57	57
AVERAGE WAIT TIME	0.	0.
MAXIMUM WAIT TIME	0.	0.
NO. INTERPROCESSOR REQUESTS	57	57
AVERAGE WAIT TIME FOR PE	0.	0.
MAXIMUM WAIT TIME FOR PE	0.	0.
NO. OF PE INTERRUPTS	0	0
AVERAGE TIME PER INTERRUPT	0.	0.
MAXIMUM TIME PER INTERRUPT	0.	0.
MODULE CURRENTLY BEING PROCESSED		
PER CENT UTILIZATION OF PE	3.95	3.95

Figure 4

CACI NETWORK II.5 VERSION 1.1 2/1/84 07:52:17
THE 1984 WINTER SIMULATION CONFERENCE EXAMPLE

BUS UTILIZATION STATISTICS
TO SIMULATED TIME 1000.0000 SECONDS

BUS NAME	BUS
NO. OF BUS REQUESTS GRANTED	114
AVG. TIME PER REQUEST (USEC)	346644.912
PER CENT OF TIME BUSY	3.952
PE CURRENTLY BEING SERVICED	

Figure 5

CACI NETWORK II.5 VERSION 1.1 2/1/84 07:52:17
THE 1984 WINTER SIMULATION CONFERENCE EXAMPLE

COMPLETED MODULE SUMMARY
TO SIMULATED TIME 1000.0000 SECONDS

MODULE	NO. OF TIMES EXECUTED THIS ON PE	AVERAGE TIME (IN USECS)
REQUEST "DATA" FILE COMPUTER A	34	237160.000
REQUEST "MORE DATA"FILE COMPUTER B	23	28973.913
TRANSMIT "DATA" FILE COMPUTER B	34	666400.000
TRANSMIT "MORE DATA" FILE COMPUTER A	23	733040.000

Figure 6

```
    S N A P    S H O T    O U T P U T    A T

          TIME = 1000.0000 SECONDS

PE " COMPUTER A" IS IDLE
  "SEND "DATA" FILE" IS IN THE RECEIVED.MESSAGE.LIST

PE "COMPUTER B" IS IDLE
  "SEND "MORE DATA" FILE" IS IN THE RECEIVED.MESSAGE.LIST

BUS "BUS" IS IDLE
```

Figure 7

```
T I M E    L I N E    S T A T U S    P L O T  --  N E T W O R K

CPU 1  ---------XXXXXXXXXXXXX-------XXX-----------------X
       +----+----+----+----+----+----+----+----+----+----+

CPU 2  -----------------XXXXXXXXXXXXXXXXXXX--------------
       +----+----+----+----+----+----+----+----+----+----+

BUS 1  --------------------------XXXXXXXXX--------------
       +----+----+----+----+----+----+----+----+----+----+
     1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0

                        TIME IN SECONDS
```

Figure 8

```
DO YOU WISH TO TRACE THE EVENT FLOW ? (Y / N)
>y
ENTER TIME TO START TRACE (IN MICROSECONDS)
>0
AT 0.0 : MODULE1 IS BEING ACTIVATED FOR EXECUTION

AT 0.0 : PE "COMPUTER A" WILL ATTEMPT TO INITIATE THE EXECUTION OF A MODULE.

AT 0.0 : ASK FOR "DATA FILE HAS BEEN ASSIGNED TO PE "COMPUTER A" WHICH IS AVAILABLE.

AT 0.0 : PE "COMPUTER A" WILL ATTEMPT TO EXECUTE INSTRUCTION "REQUEST "DATA" FILE" FROM
        MODULE ASK FOR "DATA" FILE

AT 0.0 : MESSAGE INSTRUCTION "REQUEST "DATA" FILE FROM MODULE ASK FOR "DATA" FILE
        IS BEGINNING TO EXECUTE ON PE "COMPUTER A"

AT 15.71 :   .
             .
             .
```

Figure 9