CONSTRUCTION OF QUEUE-BASED SIMULATORS FOR WEB APPLICATIONS DISTRIBUTED ON COMPUTER NETWORKS

LEONARDO PASINI AND SANDRO FELIZIANI

Departments of Mathematics and Computer Science,
University of Camerino,
Madonna delle Carceri, 62032 Camerino, Italy
{leonardo.pasini, sandro.feliziani}@unicam.it

(Received 1 June 2005; revised manuscript received 4 September 2005)

Abstract: Modern Internet and web applications involve interactions among remote host computers connected by communication networks. Simulation modelling is an important technique to evaluate performance in the implementation of a web-distributed application over given networks formed by computer hosts connected by heterogeneous networks. In this paper we extend a previous work concerning the construction of a queue-based simulator of communication networks. We define a set of new queue-based object types which permit us to specify the behaviour of a web-distributed application whose software components run over different hosts in a given communication network. We apply this technique to show how to build a simulator to evaluate the performance of various software architectures for a web-distributed application run over a given computer network.

Keywords: queuing systems, web-distributed applications, computer networks simulation, communication systems, web interaction paradigms

1. Introduction

In [1] we introduced a method of traffic specification in telecommunications systems. The method was based on definitions of the following set of devices/items: Flux, Package, Host, Routers, Connection Lines between Host and Router, and Connection Line between Router and Router. Each one of these devices was characterised through a description of its functional requirements in a telecommunications system.

For each of the above-listed devices, we defined a model with a queue- and user-based architecture that enabled us to simulate its functioning in a telecommunications system.

The work enabled us to create a library of items and a method to automatically generate a simulator of a given telecommunications system. We defined a procedure that, given a telecommunications system described by the specification method, automatically generates a system simulator. The procedure required an input of a list of the system's components with values of their characteristic parameters.

The present work remains within this context. Its aim is to develop a level above that implemented in our previous work. We have developed a method to specify and simulate software applications carried out while distributed on a number of computers interconnected via a telecommunications network. We have considered the problem of defining a method to specify the functioning of a software application whose components are carried out at different Hosts. In order to achieve that, we have defined the following new types of structural objects of queue-based architecture: *Terminal* and *Software*. We have also defined a new type of non-structural objects that is a subtype of the user type, *viz.* `WEBAPP`.

A *Terminal* object must be connected to a Host object and it is where a software component runs. Execution of a software component generates a flow of requests towards remote hosts where other components of the considered software application run. The flow is managed at the Host level and at the telecommunications network level as described in our previous work.

A *Software* object individuates a software component of a distributed web application. This object enables us to describe the stages of local execution of the web application to the specific Hosts where its components are set up. The stages are specified by describing the elaboration software blocks required by the application on devices within the Terminal objects linked to the Hosts where its components are set up. Local execution of a software block ends when the application's execution or the Software object sends a request to a remote Host. The request forwarded from Software A set up at Host HA to Software B set up at Host HB is forwarded from HA to HB via a communications network.

A `WEBAPP` object individuates the execution flux of a web application distributed over a computer network. Software blocks of local execution in single components of the application and messages forwarded therefrom through the network during execution are specified with this object.

The specification method elaborated in the present work to describe and simulate the execution of a software application distributed over a given computer network consists in fulfilling the following activities to characterise the structural part of the elaboration system:

- listing all the Software components used by the web application;
- associating every Software component to a Host of the computer network.

The behavioural part of the elaboration is characterised through the following specification activities describing the execution flux of the application:

- listing the application's execution stages;
- stating for each elaboration stage the software blocks performed on the Terminal objects where the application's software components have been set up and the size of messages activated through the network.

2. The `WEBAPP` object

We would like to introduce here a new type of library objects specifying the requirements of a web application in our context. An object of this type does not have a queue-based architecture and is defined as a subtype of the Customer type.

WEBAPP is introduced to describe the execution flux of a web application whose software architecture shows the following execution characteristics on a given network composed by Hosts.

A computational component A of an application located at host HA needs the result of a service, and there is another host HB that can be involved in the execution of this service. We have focussed our investigation on two paradigms of interaction among different hosts.

Firstly, we consider the Client-Server (CS) paradigm, according to which component B offering the service is located at host HB. Its resources and the knowledge necessary for the service's execution are also located at HB. The client component, A, located at HA, requires the execution of the service to component B. This component realises the service using the knowledge and resources located at HB.

Secondly, we consider the Mobile Agent (MA) paradigm, according to which knowledge is owned by component A located at HA, but the necessary resources are located at HB. In order to carry out the service component, A migrates to HB, bringing the necessary knowledge with it. Once HB is reached, A performs the service using the resources available at HB.

Other paradigms can be described in a similar way, including the Code on Demand (COD) paradigm and the Remote Evaluation (REV) paradigm.

In Figures 1 and 2 execution diagrams [2] are shown relative to two examples of software models for web applications structured according to the CS and MA paradigms. The diagrams are divided in three columns. The left column describes the software execution blocks of component A of the web application at host HA. The central column describes the software blocks performed by the network. The right column contains the software block performed by component B at host HB.

Each software execution block relative to component A is described by three integers of the following meaning:

1. number of visits to the HA terminal;
2. number of operations at the HA CPU;
3. number of accesses to the HA disk.

The blocks performed by the network are described by an integer marking the size (in bytes) of the http message sent through the network from component A to component B.

Each software execution block relative to component B is described by a couple of integers of the following meaning:

1. number of operations at the HB CPU;
2. number of accesses to the HB disk.

The definition code of the WEBAPP object type in the context of programming with the QNAP2.V9 language [3] is given below. The ways in which a realisation of this kind expresses the specifications of a web application are also discussed with the help of Figure 3. The specifications are expressed in the execution phases that can be described by means of performance diagrams similar to those shown in Figures 1 and 2:

```
CUSTOMER OBJECT WEBAPP;
INTEGER AT,ACPU,AD,N,BCPU,BD;
INTEGER OP,L;
```

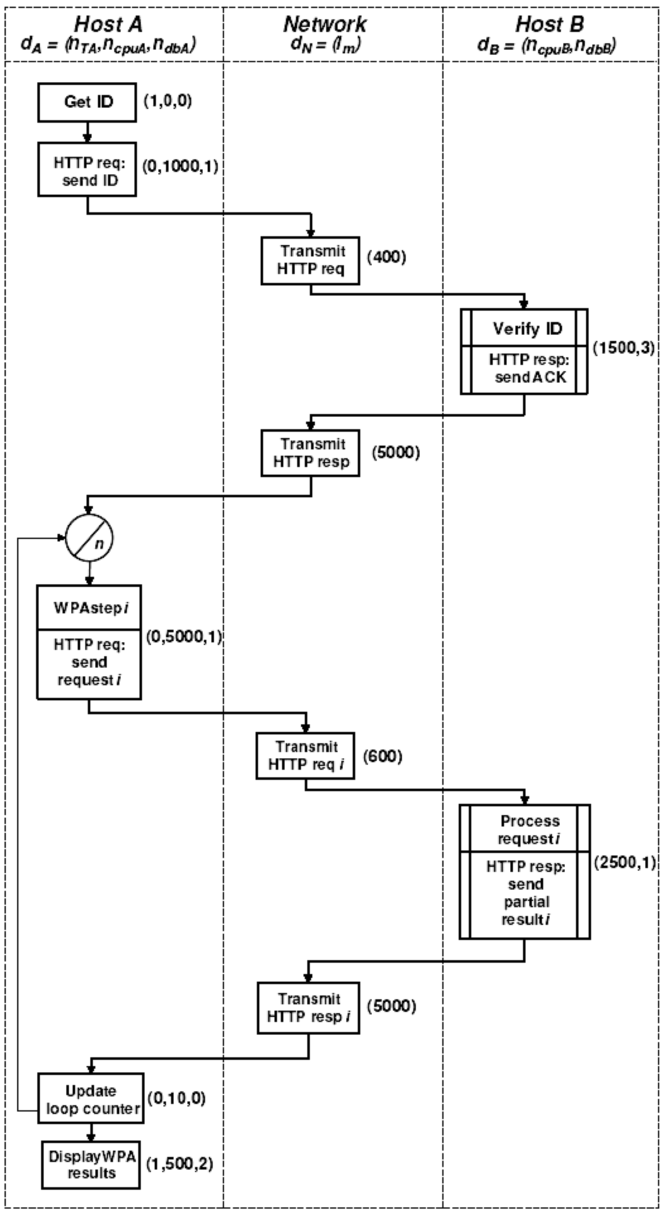

Figure 1. Execution diagram with the CS paradigm

```
INTEGER WTOUT;
INTEGER SEQ_N;
INTEGER SOUR_NP,DEST_NP;
INTEGER NF,R;
END;
```

The concept of execution phase for a WEBAPP object is fundamental to model the way in which a WEBAPP user causes the reading from a description file the data

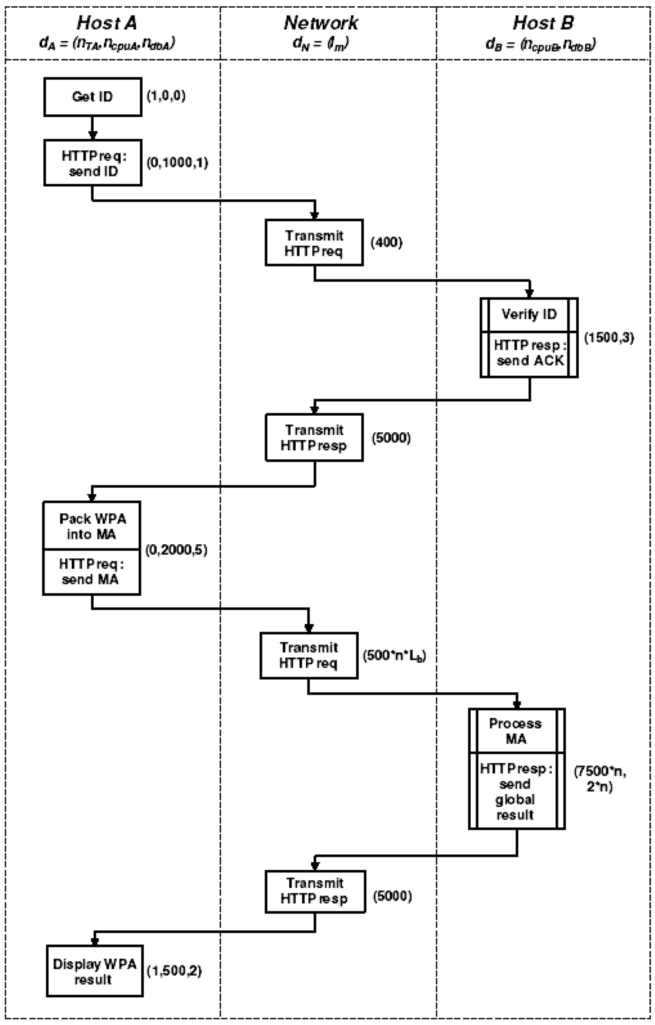

Figure 2. Execution diagram with the MA paradigm

characterising the software blocks of execution, during the simulation run within the service device of a component. This mechanism is illustrated in detail below, in the paragraphs describing the architecture of Software and Terminal objects. The execution plan of a web application on the considered computer network is supplied through a text file listing, in the chronological order of execution, for each phase the software block descriptions relative to component A, to the network and to component B, as described previously in Figures 1 and 2. This file is formed by a list of n-tuple ordinate of six integers, whose meaning is illustrated hereafter. The length of the list states the number of phases in the execution of the web application of the computer network. The six integers of any n-tuple of the web application's execution plan are given according to the inner variable of the `WEBAPP` object stated hereafter, characterising the software execution blocks of the corresponding phase:

1. `ACPU`: states the number of operations required in the phase at the CPU of the HA host;
2. `AD`: states the number of disk accesses required in the phase at the HA host;
3. `AT`: states an estimate of the time the application will have to spend during the phase for the operations of I/O with the user;
4. `N`: states the number of bytes transmitted in the phase via the network;
5. `BCPU`: represents the number of operations required in the phase at the CPU of the HB host;
6. `BD`: represents the number of disk accesses required in the phase at the HB host.

A schema of execution is shown in Figure 3; it concerns a web application where single software execution blocks divided into phases are stated.

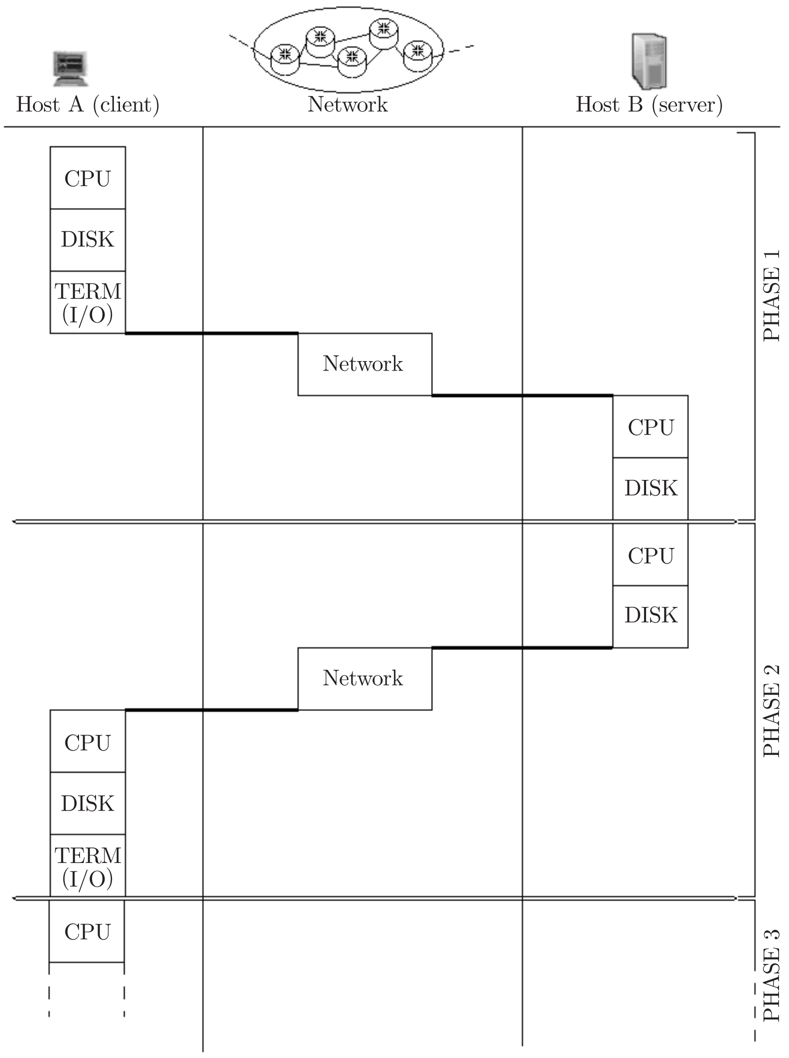

Figure 3. Execution scheme of a web application

The package transmitted via the network during a web application's execution phase is sent from the software component through the Terminal object to the connected Host. From there the package is directed to a network Router and then towards the Host, where the second software component used by the web application is connected to. Having reached the destination Host, the package is transmitted according to the protocol procedures to the above Terminal. Then it is transmitted to the connected Software component executing the end-of-phase software block. All the transmissions through the communication network devices depend on the size of the package sent.

Tables 1 and 2 list the execution phases of two web applications, specifying in the respective files the applications' execution plan. The applications use the Client-Server and the Mobile Agent paradigms. Their execution plans have been derived from the execution graphs shown in Figures 1 and 2. The applications are structured according to different paradigms, but – from a functional point of view – they perform the same tasks.

Table 1. Execution plan for the CS application

ACPU	AD	AT	N	BCPU	BD
1000	10	100	400	1500	3
1000	5	200	5000	750	1
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5
5000	4	300	600	2500	2
1000	10	200	5000	100	5

The `WEBAPP` object has variable members of the following functional meaning in the service algorithms within the queues crossed by the user while simulating the web application's execution flux:

Table 2. Execution plan for the MA application

ACPU	AD	AT	N	BCPU	BD
1000	10	100	400	1500	3
1000	5	200	5000	750	1
2000	5	100	150000	45000	12
5000	1	300	5000	45000	12

- OP: states the next operation to be performed within a phase (*e.g.*: access to disk, transmission via the network, *etc.*);
- L: states the side where the execution of the web application takes place (A or B);
- WTOT: states the time-out time, after which a transmission is considered as failed and is repeated;
- SEQ_N: states the position occupied by the package in the network in the transmission sequence and helps to discover packages duplicated in reception (the mechanisms of retransmission and recognition of the sequence are described in detail in the description of Hosts [1]);
- SOUR_NP: specifies the Software that produced the package (in the TCP protocol this username is called a door number);
- DEST_NP: specifies the Software to which the package is addressed;
- NF: states the current phase of the application (this variable is used for the filing of statistics and results);
- R: states the position in an array of the file used to store the simulation's results and statistics.

The network traffic generated by the Hosts that are not locations of Software components' elaboration has the characteristics described in our previous work [1]. This traffic utilixes the following user classes:

1. EMISSION: is the class of the requests issued by the Host;
2. RICHIN: is the class of the requests coming to the Host;
3. RISPOSTA: is the class of the answers coming to the Host.

We have introduced here a new class of users, WEBAP, that identifies the execution flux of the web application on the communication network.

3. The Software object

The architecture of the Software object is shown in Figure 4. This device has only one Program queue. This queue receives the flux of users coming in from the Terminal object to which the Software object is connected. The users served by

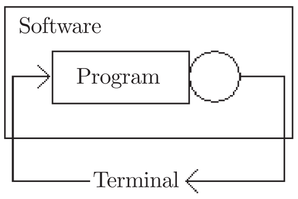

Figure 4. The Software object

the Program are then sent to the SCH queue of the Terminal object. The Program queue serves users of the WEBAPP type. When a WEBAPP user enters the device at the beginning of a web application's execution phase, the service algorithm reads the data corresponding to the phase from the planning file, assigning them to the variables of the WEBAPP user. The WEBAPP is then transferred to the SCH queue that is a control device of the flux in the Terminal object. Here, according to the value of variable L in the WEBAPP, the SCH service algorithm recognises whether the web application's execution is to take place on side A or side B (as per the previous paragraph). The SCH service algorithm manages the forwarding of the WEBAPP in the queues of the terminal object, updating the WEBAPP's inner variable OP according to the data relative to the first software block of the phase.

A definition code of the new Software object type in our programming context is given below. We also report comment on some parts of the service procedure in the Program queue:

```
OBJECT SOFTWARE;
    QUEUE PROGRAM;
    INTEGER EXIT;
    INTEGER IDS;
    REF FILE FA;
    REF WEBAPP WA;
    INTEGER BUF;
    INTEGER C_SEQ_N;
END;
```

In particular we describe the use of the following inner variables:

- EXIT is used in the simulator construction phase to store the ID number in the network of the terminal object to which the software object is connected;
- IDS is used to manage the beginning of the web application's execution. Its values are read from the data file that enables generation of the simulator contextually to the reading of the identifier numbers of the HA and HB hosts of the network where the web application's components are performed. If IDS has a value different from zero, the Software object corresponds to the component performed on the HA side of the web application and the queue Program generates a WEBAPP user in the system by a BOOT class user;
- FA is a pointer-to-file object. Once the simulator is constructed, in the presence of a web application using the Software object, the text file containing the execution plan of the web application is assigned to FA.

The QNAP2.V9 programming context enables us to define in general terms the service procedures of a queue in any realisation of a given type of object defined by the user. In this context, we have applied these mechanisms in particular to define the service procedures of queues in software and terminal objects:

```
/STATION/
    NAME = *SOFTWARE.PROGRAM;
    INIT(BOOT)=1;
    SERVICE (BOOT)= BEGIN
        IF (IDS=0) THEN BEGIN
            PRINT ("CUSTOMER FUORI IDS",IDS);
            TRANSIT (OUT);
        END
        ELSE
        BEGIN
            WA:=NEW (WEBAPP);
            WA.OP:=0;
            WA.L:=0;
            WA.SEQ_N:=1;
            WA.TYPE:="web";
            WA.TYP:="app";
            OPEN (FA,1);
            WA.NF:=0;
            WA.R:=IDS;
            FILASSIGN(SRT(IDS),FA.FILASSGN//".srt");
            OPEN(SRT(IDS),2);
            FILASSIGN(SRN(IDS),FA.FILASSGN//".srn");
            OPEN(SRN(IDS),2);
            WA.SOUR_ID:=GET(FA,INTEGER);
            WA.SOUR_NP:=GET(FA,INTEGER);
            WA.WTOUT:=GET(FA,INTEGER);
            NewLn(FA);
            WA.DEST_ID:=TERM#(EXIT).IDT;
            WA.DEST_NP:=IDS;
            TRANSIT (WA,PROGRAM,WEBAP);
        END;
    END;
    SERVICE (WEBAP)= BEGIN
        WITH (CUSTOMER::WEBAPP) DO BEGIN
            BUF:=SOUR_ID;
            SOUR_ID:=DEST_ID;
            DEST_ID:=BUF;
            BUF:=SOUR_NP;
            SOUR_NP:=DEST_NP;
            DEST_NP:=BUF;
            ACPU:=GET(FA,INTEGER);
            IF (ACPU=-1)THEN BEGIN
                PRINT ("END");
                WRITELN (SRT(R),"Total phase ",NF," ",TIME);
                CLOSE (SRT(R));
                TRANSIT(OUT);
            END;
            AD:=GET(FA,INTEGER);
            AT:=GET(FA,INTEGER);
            N:=GET(FA,INTEGER);
            IF (N=0) THEN PRINT ("ERROR: inconsistent data");
            BCPU:=GET(FA,INTEGER);
            BD:=GET(FA,INTEGER);
            NewLn(FA);
            WRITELN (SRT(R),"Total phase ",NF," ",TIME);
            NF:=NF+1;
            OP:=1;
            SEQ_N:=SEQ_N+1;
            TRANSIT (TERM#(EXIT).SCH);
        END;
    END;
    TRANSIT=OUT;
```

All the Program queues of any given Software object are initialised with a BOOT class user inside. This user is rejected if the Software object is used in the web application's execution as a B component. Otherwise, before being rejected it generates a new WEBAPP user that is reinserted in the Program queue with the WEBAP class to start the first execution phase of the web application.

When a WEBAPP user enters the Program server, destinations and sources are exchanged. In fact, if in a phase the application has run through the network from an HA host to an HB host, it will have to make the reverse path in the following phase (*i.e.* from HB to HA). Additionally, a reading from the FA file of a new n-tuple of integers is made. These numbers are assigned to the inner variables by the ACPU, AD, AT, N, BCPU and BD WEBAPP users. The sequence number is increased so that the new transmission cannot be confused with the previous reception.

At every file reading, a check on the first value of the n-tuple of integers is performed. If the value of -1 is obtained, the application has finished its execution. A control on the quality of transmitted bytes is also performed and it should be other than zero.

The service algorithm of the Program queue registers the moments of the execution being finished at each software block, for every phase of the web application (also in a text file).

4. The Terminal object

The Terminal object represents the whole of the local hardware structures for the elaboration of the web application's software blocks. It is connected to a Software object above from which it receives the flux made by the WEBAPP user to execute the software block of the beginning of the phase. It is also connected to a Host below where it sends the WEBAPP user after a software block has been executed in order to send a request through the communication network to the remote host used by the Web application. A realisation of the Terminal object always receives the WEBAPP user of the Host below for the execution of the second software block of the phase, after transmission from the remote host. Also in this case, the inner SCH queue recognises the WEBAPP user and sends it to the inner devices of the terminal object to execute the computation requests relative to the second software block of the phase. At the end of the second software block's execution, the SCH queue sends the WEBAPP user in the Program queue of the Software object above in order to start a new execution phase of the web application.

Figure 5 illustrates the architecture of the terminal object type. The structure of this new kind of object is rather complex. It contains five queue devices of the following functions, with a single server:

1. the USER queue is a source of traffic. This queue is active in those Terminal realisations that are not elaboration locations of web applications. They generate a flux of requests of the EMISSION class that are inserted in the network by the HOST below. This traffic is present in the communication network at the same time as the traffic WEBAP-class requests generated by the web application's execution;

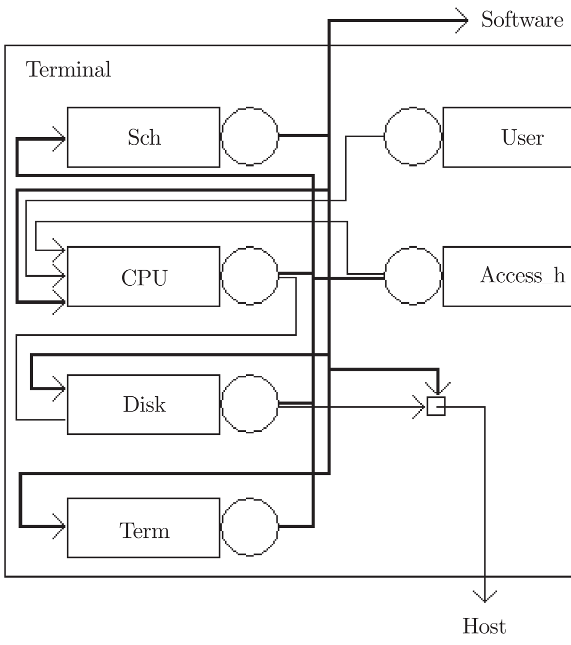

Figure 5. The Terminal object

2. the `SCH` queue serves `WEBAP`-class users only. It is active only in realisations of Terminal objects that are locations of an elaboration of a web application. Its service algorithm manages sending of the `WEBAPP` user in the `CPU`, `DISK` and `TERM` devices of the Terminal object for the execution of the web application's software blocks. This control activity is performed through inner variables `OP` and `L` of the `WEBAPP` object;
3. the `CPU` queue serves all classes of users. For users of the `WEBAP` class it has an average service time that depends on the average number of operations requested of the CPU in the execution of a particular software block of the web application;
4. the `DISK` queue also serves all classes of users. For users of the `WEBAP` class it has an average service time that depends, directly proportionally, on the average number of disk accesses requested in the execution of a particular software block by the web application;
5. the `ACCESS_H` queue is an entrance-to-Terminal device for packages of any class coming from the host below;
6. the `TERM` queue serves `WEBAP`-class users only. It has an average service time that depends on the cumulative think time required for the user in the execution of a web application's software block.

A definition code of the new type of Terminal object is given below. We also give some service procedures of the queues in the new object type, which we have already described functionally:

```
OBJECT TERMINAL (IDT);
    QUEUE USER, ACCESS_H, CPU, DISK;
    QUEUE SCH,TERM;
    BOOLEAN ACTIVES;
    INTEGER IDT;
    INTEGER ID_H;
    STRING NAME;
    INTEGER REQLENGTH;
    REAL PROCESS, PROC_D, US_TIME;
    REF REQUEST RQ;
    REF QUEUE NET;
END;
```

The following member variables of a Terminal-type object are related to the functioning of service algorithms of the queues in the object:

1. `ACTIVES` is used for the activation of the `USER` source;
2. `IDT` is the unequivocal identification number of a Terminal realisation;
3. `ID_H` is the identification number of the host to which the Terminal realisation is connected;
4. `REQLENGTH` is the length of a generic request for a CPU or a disk in a Terminal realisation;
5. `PROCESS` is the time necessary for the CPU to carry out an operation;
6. `PROC_D` is the time necessary to access a disk.

```
/STATION/
    NAME = *TERMINAL.SCH;
    SERVICE(WEBAP) = BEGIN
        WITH (CUSTOMER::WEBAPP) DO BEGIN
            IF (OP=0) THEN TRANSIT (SW#(DEST_NP).PROGRAM);
                IF (OP=1) THEN BEGIN
                    OP:=2;
                    TRANSIT (CPU);
                END;
            IF (OP=2) THEN BEGIN
                OP:=3;
                TRANSIT (DISK);
            END;
            IF (L=1) THEN BEGIN
                IF (OP=3) THEN BEGIN
                    OP:=5;
                    TRANSIT (NET);
            END;
. . .
/STATION/
    NAME = *TERMINAL.CPU;
    SERVICE (WEBAP) = BEGIN
        WITH (CUSTOMER::WEBAPP) DO BEGIN
            IF (L=1) THEN EXP(BCPU*PROCESS)
            ELSE EXP(ACPU*PROCESS);
            WRITELN (SRT(R),"CPU TIME",TIME);
            TRANSIT (SCH);
        END;
    END;
    SERVICE = EXP(PROCESS);
    TRANSIT = DISK;
/STATION/
    NAME = *TERMINAL.DISK;
```

```
          SERVICE(WEBAP) = BEGIN
              WITH (CUSTOMER::WEBAPP) DO BEGIN
                  IF (L=1) THEN EXP(BD*PROC_D)
                  ELSE EXP(AD*PROC_D);
                  WRITELN (SRT(R),"DISK TIME",TIME);
                  TRANSIT (SCH);
              END;
          END;
          SERVICE(EMISSION) = BEGIN
              EXP(PROC_D);
              TRANSIT(NET,EMISSION);
          END;
          SERVICE(RICHIN) = BEGIN
              EXP(PROC_D);
              TRANSIT(NET,RICHIN);
          END;
          SERVICE(RISPOSTA) = BEGIN
              EXP(PROC_D);
              TRANSIT(OUT);
          END;
          TRANSIT = OUT;
      /STATION/
          NAME = *TERMINAL.TERM;
          SERVICE (WEBAP) = WITH (CUSTOMER::WEBAPP) DO BEGIN
              EXP(AT);
              WRITELN (SRT(R),"TERMINAL TIME ",TIME);
          END;
          TRANSIT = SCH;
      /STATION/
          NAME = *TERMINAL.ACCESS_H;
          SERVICE (WEBAP) = WITH (CUSTOMER::WEBAPP) DO BEGIN
              IF L=0 THEN L:=1 ELSE L:=0;
              WRITELN (SRT(R)," NETWORK TIME ",TIME);
              TRANSIT (SCH);
          END;
          SERVICE = BEGIN
              EXP((LUNGH*8)/CAP_BUF);
              TRANSIT(CPU);
          END;
      /STATION/
          NAME = *TERMINAL.USER;
          TYPE = SOURCE;
          SERVICE = BEGIN
              IF (ACTIVES)THEN BEGIN
                  EXP(US_TIME);
                   RQ: = NEW(REQUEST);
                   RQ.ORIG:=IDT;
                   RQ.SIZE:= REQLENGTH;
                   DEST: = RINT(1,NHTEST);
                   RQ.DESTI: = DEST;
                   LUNGH: = RQ.SIZE;
                   TRANSIT(RQ,CPU,EMISSION);
              END
              ELSE CST(T_MAX);
          END;
          TRANSIT = OUT;
```

We have previously seen that the service algorithm of the Program queue of the Software object, when it generates a `WEBAPP`-class user at the start of a web application's execution, generates two text-format files where one can trace the progress of the application's execution during the simulation.

Service algorithms of the `CPU`, `DISK` and `TERM` queues write on one of the two files, at every passage of the `WEBAPP` user, the moments of service termination collected through the system variable of system `TIME` during the simulation. The service algorithm of the `ACCESS_H` queue writes in the same file the moment of entrance of the `WEBAPP` user coming from the communication network, the Terminal object for the execution of the software block of end of phase. The chronological sequence of the moments of execution of each software block allows us to trace the temporal diagram of a web application's execution in a computer network, measured during its simulation.

We have also defined the `LINE_TH` object type to realise a way of fully duplex communication in order to connect a realisation of the Terminal object with its access Host to the communication network. The `LINE_TH` object type is the level of connection between the objects of the library introduced in this work and the objects of the library defined in our previous work [1].

The definition code of this type of object is given below:

```
OBJECT LINE_TH(TH);
QUEUE QH,HQ;
REF QUEUE TOUT,TIN;
REAL TH;
END;
```

Inner queues `QH` and `HQ` are delay centres without waiting time and simulate the transmission delay on the channel singled out from the variable member `TH`. In particular, the queues are used in the following ways:

1. the `QH` queue sends data from a Terminal object to a Host object. In this case, in the simulator's construction phase, the `BuildMod` procedure performs connections for the Terminal-`LINE_TH`-Host flux performing the following assignations. The `QH` of `LINE_TH` queue is given to the `NET` variable of the Terminal. The `USER` queue of the Host is assigned to the `TOUT` of `LINE_TH` variable;
2. the `HQ` queue sends data from a Host object to a Terminal object. In this case, in the simulator's construction phase, the `BuildMod` procedure performs the connections for the Host-`LINE_TH`-Terminal flux performing the following assignations. The `HQ` of `LINE_TH` queue is given to the `HOTE` variable of the Host. The `ACCESS_H` queue of the Terminal is assigned to the `TIN` of `LINE_TH` variable.

5. Constructing the Simulator

In this context, generating a simulator of a given system requires the execution of the following steps:

1. generation of the communication network. It consists in generating realisations of `HOST`, `ROUTER`, `LINE_HR`, `LINE_RR`, *etc.* object types forming the communication network and assigning their connections. We thus have a communication

infrastructure of the system where the traffic of generic packages and one of the packages relative to the execution of web applications will flow;

2. generation of the hardware components aimed at elaboration of the software and the management of interaction with the users. This step consists in generating realisations of the Terminal and `LINE_TH` objects, which are present in the system in the assignation of their connections. At this stage one can also perform the connection between the `LINE_TH` objects and the corresponding Host object in the system. Thus, a connection is made between the level formed by software elaboration devices and the generic traffic generation present in the system with the communication infrastructure of the system implemented in the first step;
3. implementation of a distributed web application in an elaboration network. This point consists in generating objects of software type realisations, made according to the architecture of the web application and in creating connections with the Terminal object type where the software blocks of the web application will be carried out. At this point one must also specify the file containing the execution plan of the web application whose execution we want to simulate. The service algorithm of the queues inside the software objects read the data relative to the execution phases on this file.

We have automated the execution of the above points by defining a `BuildMod` procedure. This procedure automatically generates a simulator of a given system, starting from the system's description file. The description file of the system is called `Model.dat` and it is in text format. The `BuildMod` procedure reads the data from the `Model.dat` file and, by realising the following points, it yields the system simulator as an output:

- generating the system's global variables and assigning the specific values of the studied system;
- generating realisations of the various types of objects that constitute the system devices and assigning specific values to their inner parameters;
- assigning the connections among the various system devices.

We have redefined here the `BuildMod` procedure originally introduced for the context dealt with in [1]. The new definition is so structured that in addition to point one the procedure also realises steps two and three above. The extended part of the definition code is given below:

```
& PROCEDURE BuildMod
PROCEDURE BuildMod;
INTEGER
N_Hosts,N_Router,N_Vie,N_soft,idh,idr,idv,I,J,K,idter,idhost;
BEGIN
& GLOBAL VARIABLES
N_Hosts: = GET(INTEGER);
N_Router:= GET(INTEGER);
N_Vie:=GET(INTEGER);
N_soft:=GET(INTEGER);
. . .
&---------------------------------------------------
& TERMINAL CONSTRUCTION AND TERMINAL-HOST CONNECTIONS
&---------------------------------------------------
```

```
IF (N_Hosts > 0) THEN
FOR I:=1 STEP 1 UNTIL N_Hosts DO
    BEGIN
        idter: = GET(INTEGER);
        TERM#(idter):=NEW(TERMINAL,idter);
        WITH TERM#(idter) DO BEGIN
            TERM#(idter).ACTIVES: = GET(BOOLEAN);
            TERM#(idter).NAME: = GET(STRING);
            PRINT ("GEN",TERM#(idter).ACTIVES, TERM#(idter).NAME);
            TERM#(idter).REQLENGTH: = GET(INTEGER);
            TERM#(idter).PROCESS: = GET(REAL);
            TERM#(idter).PROC_D: = GET(REAL);
        END;
        LINE_TH#(idter):=NEW(LINE_TH,GET(REAL));
& CONNESSIONI HOST-TERMINAL
        idhost:=GET(INTEGER);
        TERM#(idter).ID_H: = idhost;
        HOSTS#(idhost).HOTE:=LINE_TH#(idter).HQ;
        LINE_TH#(idter).TIN:=TERM#(idter).ACCESS_H;
        NewLine;
    END;
& CONNECTIONS TERMINAL-HOST
IF (N_Hosts > 0) THEN
FOR I:=1 STEP 1 UNTIL N_Hosts DO
    BEGIN
        TERM#(I).NET:=LINE_TH#(I).QH;
        LINE_TH#(I).TOUT:=HOSTS#(TERM#(I).ID_H).USER;
    END;
&---------------------------------------------------
& SOFTWARE CONSTRUCTION
&---------------------------------------------------
IF (N_soft > 0) THEN
FOR I:=1 STEP 1 UNTIL (N_soft) DO
    BEGIN
        SW#(I):=NEW (SOFTWARE);
        SW#(I).IDS:=GET(INTEGER);
        SW#(I).EXIT:=GET(INTEGER);
        IF (SW#(I).IDS<>0) THEN BEGIN
        SW#(I).FA:= NEW (FILE);
        FILASSIGN(SW#(I).FA,GET(STRING));
    END
    ELSE
        SW#(I).FA:= SW#(GET(INTEGER)).FA;
    NewLine;
END;
&-------------
END;
```

A part of the `Model.dat` file relative to the description of the system is shown in Figure 6. The communication network of this system coincides with the communication network studied in our previous work. In this context, we have added the levels described in steps 2 and 3 of the simulator construction scheme. The component of the `Model.dat` file shown below has been added to the previous version of the `Model.dat` file found in [1], where it described the system's sub-network communication. The component added to the `BuildMod` procedure generates the new system components described in stages 2 and 3, reading the data given below from the `Model.dat` file. We have chosen this approach to be able to evaluate the variations of the traffic fluxes present on the network due to addition of the levels described in points 2 and 3 above.

```
. . .
& Terminal
&IDT  Actives   Name  Length   CPU Time   Disk Time  LINE_TH IDH
1      TRUE     "T1"   120       0.01        20       1.20    1;
2      FALSE    "T2"   123       0.001       20       1.30    2;
3      TRUE     "T3"   124       0.01        20       1.26    3;
4      TRUE     "T4"   135       0.01        20       1.00    4;
5      TRUE     "T5"   120       0.01        20       1.10    5;
6      TRUE     "T6"   123       0.01        20       1.20    6;
7      TRUE     "T7"   201       0.01        20       1.30    7;
8      TRUE     "T8"   204       0.01        20       1.50    8;
9      FALSE    "T9"   112       0.001       20       1.20    9;
10     TRUE     "T10"  180       0.01        20       1.03   10;
11     TRUE     "T11"  110       0.01        20       0.96   11;
12     TRUE     "T12"  112       0.01        20       0.85   12;
13     TRUE     "T13"  302       0.01        20       1.30   13;
14     TRUE     "T14"  145       0.01        20       1.13   14;
15     TRUE     "T15"  143       0.01        20       1.22   15;
16     TRUE     "T16"  100       0.01        20       1.47   16;
& Web Application
&IDS  exit File
 1     2       "wp.app";
 0     9          1;
```

The last two lines of the `Model.dat` file describe the level specified in point 3 in the simulator generation chart. The software components of the implemented web application are set up at terminals 2 and 9 of the system shown in Figure 6. The application's execution starts from the Software object connected to terminal 2 and the data relative to the execution plan of the application are read by the wp.app text file set in the work directory.

We have thus generated two simulators relative to two different systems, each of them implementing one of the two web applications described with execution diagrams shown in Figures 1 and 2. The execution plans of these two web applications are given in Tables 1 and 2 and contained in two different files, `wp1.app` and `wp2.app`. In both cases, the software components of the application are located at Hosts 2 and 9 of the system.

We have created a simulation plan for each system varying the local CPU time on the two execution sides. In particular, we have considered the following combinations of values of the `PROCESS` parameter of the `CPU` queue:

1. `CPUA.Process` = 0.01 ms and `CPUB.Process` = 0.01 ms;
2. `CPUA.Process` = 0.001 ms and `CPUB.Process` = 0.01 ms;
3. `CPUA.Process` = 0.001 ms and `CPUB.Process` = 0.001 ms.

The temporal values collected in the simulation of service algorithms of the queues in the Terminal 2 and Terminal 9 objects have enabled us to built graphs illustrating the progress in execution of the considered web applications.

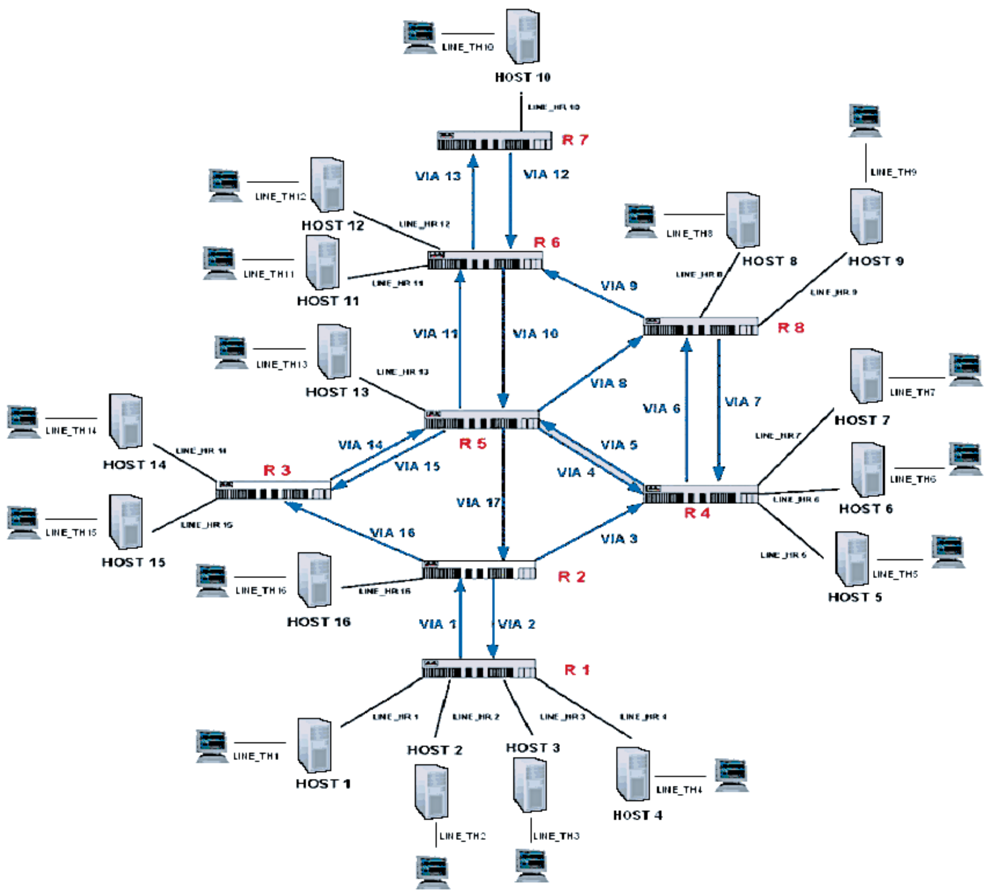

Figure 6. Network of elaboration systems for executing distributed web applications

The following three diagrams illustrate the progress monitored during the simulation of the web application built according to the Client-Server paradigm and they refer to the above combinations of CPU time values.

In Figures 7–9 every point in light gray (rotated squares) states a moment when a service is terminated in the inner queues of the Terminal object relative to the execution of the web application. The moments when these events take place enable us to trace the application's execution during the simulation. Each of these events marks the end of an operation in a web application execution phase, according to the kind of schematisation we have introduced. The points in dark gray (squares) correspond to the moments of ending an execution phase. The triangular black points correspond to the moments of re-sending a package caused by a loss or an excessive delay of the consignment.

The total execution time of the web application shown in Figure 7 is 121 600ms. In this case, the application's execution is characterised by multiple re-sending packages in the network, causing considerable delays in execution.

The total execution time of the web application shown in Figure 8 is 84 990ms. In this case, the execution is more compact. Please note the increase of the CPUA power and less frequent re-sending events. Delays due to transmission in the network

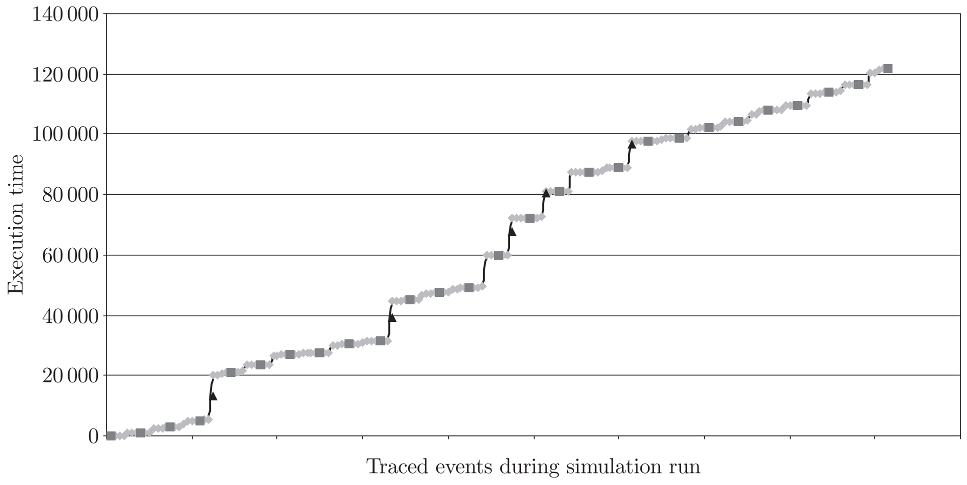

Figure 7. Traced events during simulation run, CS application – `CPUA` = 0.01 ms, `CPUB` = 0.01 ms

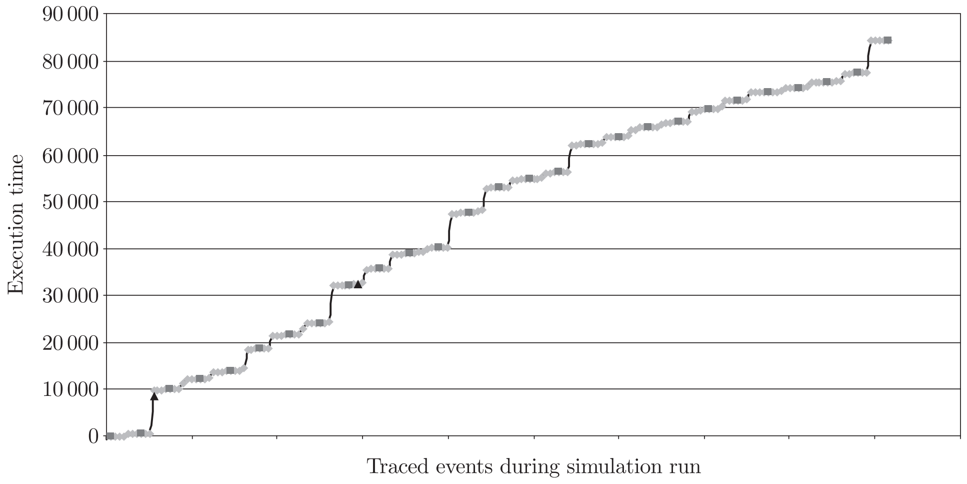

Figure 8. Traced events during simulation run, CS application – `CPUA` = 0.001 ms, `CPUB` = 0.01 ms

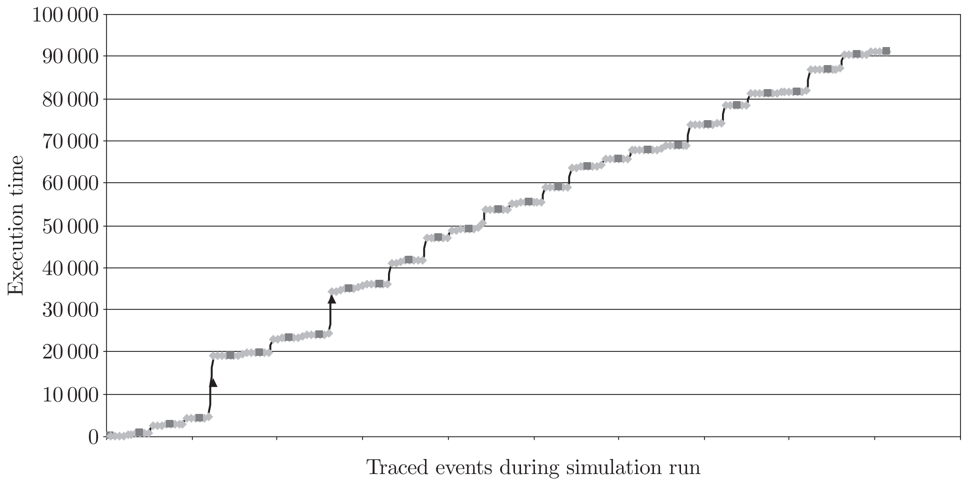

Figure 9. Traced events during simulation run, CS application – `CPUA` = 0.001 ms, `CPUB` = 0.001 ms

are noticeable, but their extent is limited and they are distributed along the whole execution of the application.

The total execution time of the web application shown in Figure 9 is 91 290ms. In this case, a delay of about 15 000ms took place during the fourth phase of the first part of the application's execution due to the effective loss of the application package in transmission. This event was partly compensated for by the increase in the CPUB power in comparison to the two previous cases.

Similarly, we have built the following diagrams referring to web application simulations according to the Mobile Agent paradigm. We have used the same graphic annotation to state the end of service events, the end of phase one and sending back a package in the network during the application's execution. Some instances of application packages being re-sent were caused by delayed consignment of the corresponding "ack" package that exceeded the time-out parameters of the devices. In this case, the re-sending event had no influence on the execution phase and the copy sent back was then erased by the system.

Figure 10 shows the web application generated according to the Mobile Agent paradigm to have a number of phases inferior to those of the Client-Server architecture. The MA paradigm is characterised by transferring the whole application from side A of the execution to side B, as previously described. In this simulation, the transfer from Terminal 2 to Terminal 9 takes place in the third phase of the application's execution. This phase is characterised by three events of packages being re-sent, which have strongly influenced the total execution time. The diagram shows an execution time slightly inferior to that shown in Figure 7, corresponding to the CS paradigm.

Figures 11 and 12 illustrate the improvements in the global execution time obtained with the MA architecture, increasing the power of CPUA and CPUB. In particular, the improved power of CPUB greatly improves the execution performance of the of the web application build according to the MA paradigm that carries the bulkiest software block on side B. The improved capacity of CPUB compensates the delay due to sending the application via the network.

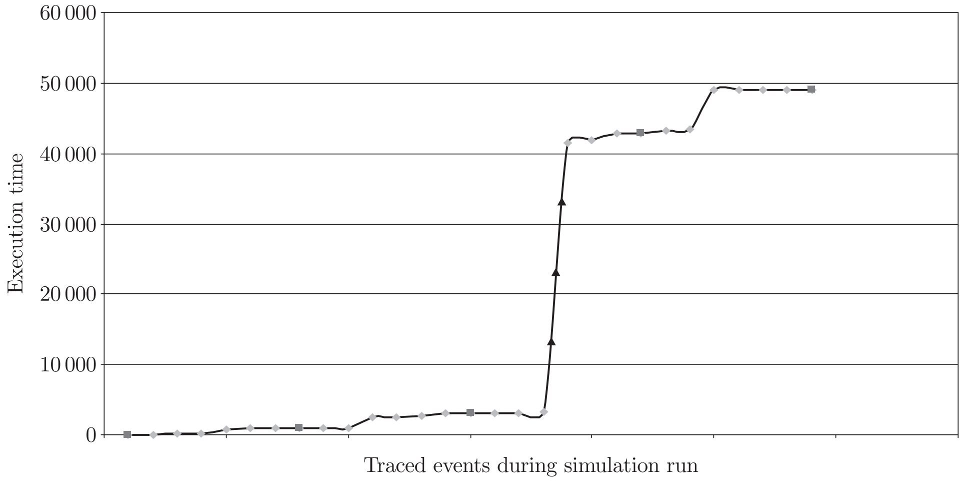

Figure 10. Traced events during simulation run, MA application – CPUA = 0.01 ms, CPUB = 0.01 ms

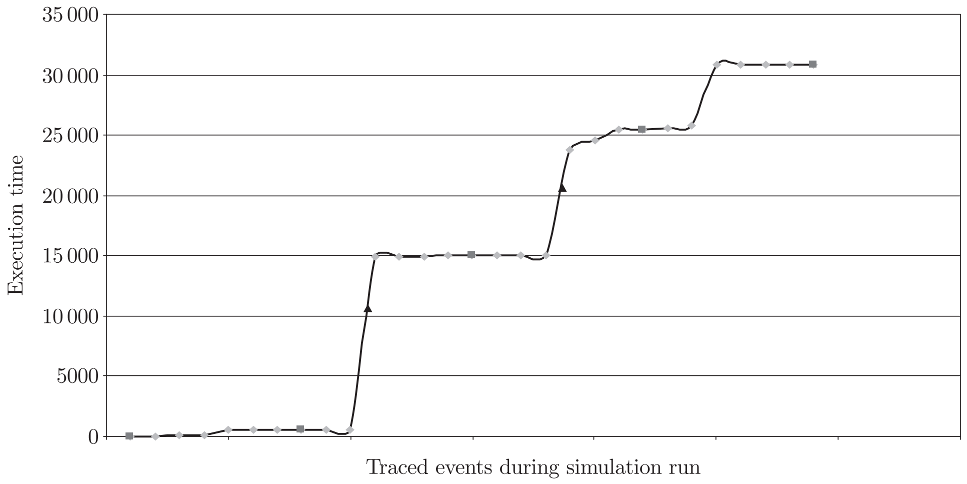

Figure 11. Traced events during simulation run, MA application – `CPUA` = 0.001 ms, `CPUB` = 0.01 ms

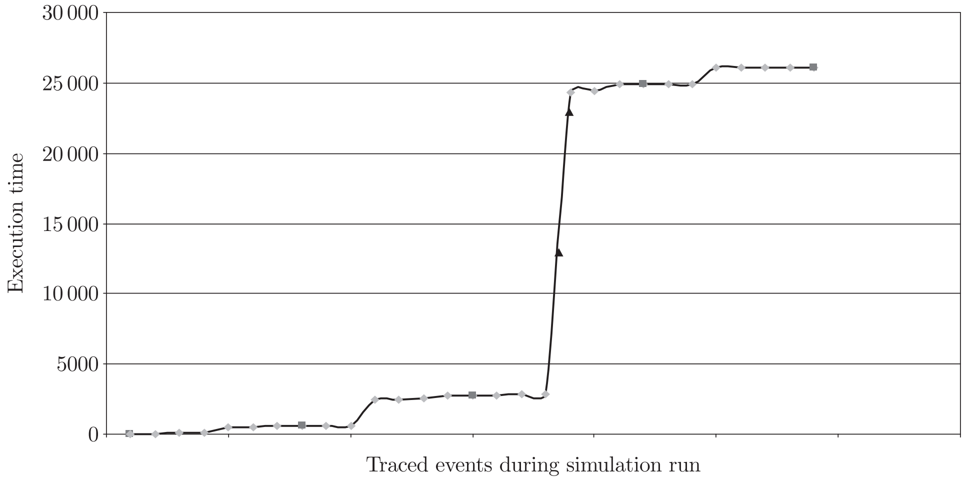

Figure 12. Traced events during simulation run, MA application – `CPUA` = 0.001 ms, `CPUB` = 0.001 ms

Figure 13 offers a comparison of the global execution times of of web applications realised by the two architecture Client-Server and Mobile Agent. The comparison has been made on the basis of a simulation plan characterised by three applications of values relative to the power of CPUA and CPUB. We have used the graphic notation of `L` = 0.01 ms and `V` = 0.001 ms. This diagram highlights the level of communication of the system simulated in the net infrastructure shown in Figure 6. The infrastructure shown in the figure and used to simulate the web application's execution was an interdepartmental academical network, consisting of 100.0 Mbps Departmental LANs Ethernet interconnected by a 1.0 Gbps Fast Ethernet network. The traffic bottleneck of this communication system was the entrance and exit doors of the hosts towards the net. These devices have a capacity varying between 0.2 Mbps and 1.0 Mbps. Therefore, we can assume that the HA and HB hosts where the software components of the web application were carried out must be connected respectively to the IT Depart-

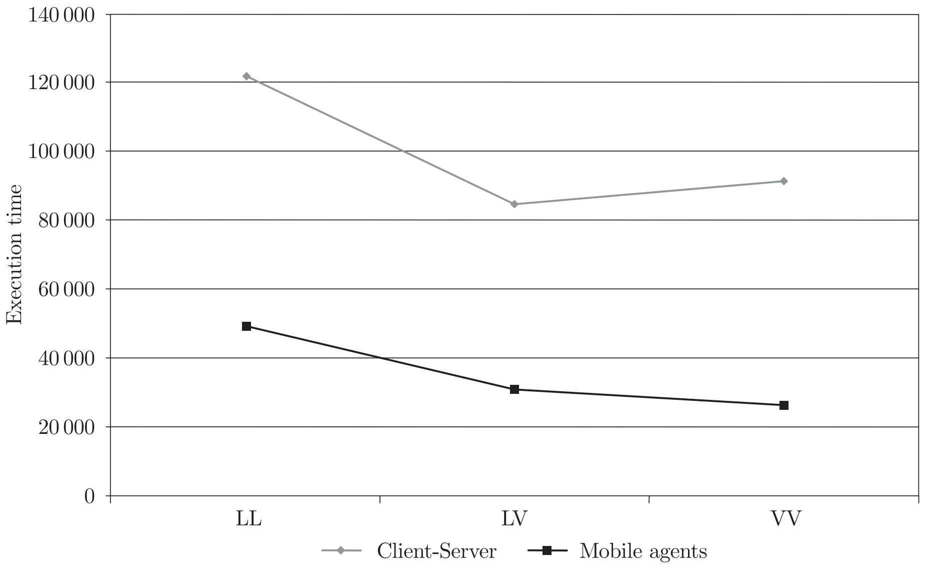

Figure 13. Global execution time of the web application

ment's and the Physics Department's networks and inter-connected to the University Campus network. In this context, Figure 13 clearly shows that the web application built according the Mobile Agent paradigm is more efficient than that built according to the Client-Server paradigm. This case is an example where the system's level of communication does not constitute a bottleneck of the global elaboration system. This factor, in conjunction with the high power of CPUA and CPUB, has determined the choice of the Mobile Agent paradigm for the web application's implementation in the network.

6. Conclusions

In this paper we have extended the library of item types defined in our previous work to construct simulators of communication networks. The object types defined previously and those introduced here have an architecture that specifies their functioning based on the concepts of queue and user. The new types of objects introduced in the library enable implementation above the level formed by the communication network, a level representing the software elaboration plants and a level representing the functioning of the software itself. In particular, this last level enables execution of software applications whose software components are carried out at various elaboration plants present in the system. The software components are elaborated locally in a synchronous manner and the results and requests are communicated to the remote components through the communication network. We have applied these results to the study of a specific case to choose a paradigm of web interaction on which the structure of a software application can be based. We have compared the Mobile Agent and the Client Server paradigms applied to a distributed web application executed on two different hosts placed connected by an inter-departmental network of a University Campus. The results of this comparison complement the case studied in [2]. By using other simulation techniques, we have valued the two paradigms of web interaction in an elaborated distributed system formed by local LANs interconnected by a WAN net to 100 Kbps.

References

[1] Pasini L and Feliziani S 2004 *TASK Quart.* **8** (3) 333

[2] D'Ambrogio A, Iazeolla G and Pasini L 2004 *Simulation Practice and Theory*, Elsevier (in print)

[3] *Simulog*, QNAP2 Reference Manual, ver. 9.3