Int. J. Appl. Math. Comput. Sci., 2007, Vol. 17, No. 1, 99–106
DOI: 10.2478/v10006-007-0009-0

amcs

A FEDERATED APPROACH TO PARALLEL AND DISTRIBUTED SIMULATION OF COMPLEX SYSTEMS

ANDRZEJ SIKORA *, EWA NIEWIADOMSKA-SZYNKIEWICZ **

* Research and Academic Computer Network (NASK)
ul. Wąwozowa 18, 02–796 Warsaw, Poland
e-mail: asikora@nask.pl

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

The paper describes a Java-based framework called ASimJava that can be used to develop parallel and distributed simulators of complex real-life systems. Some important issues associated with the implementation of parallel and distributed simulations are discussed. Two principal paradigms for constructing simulations today are considered. Particular attention is paid to an approach for federating parallel and distributed simulators. We describe the design, performance and applications of the ASimJava framework. Two practical examples, namely, a simple manufacturing system and computer network simulations are provided to illustrate the effectiveness and range of applications of the presented software tool.

Keywords: parallel simulation, distributed simulation, federated simulators, computer networks simulation, Frame Relay

1. Introduction – Parallel and Distributed Simulations

The simulation of physical systems is an important tool for researchers that allows them to analyze the behavior or/and performance of the system considered and to verify new ideas. A variety of software environments for computer simulation are available today (JavaSim, 2007; Niewiadomska-Szynkiewicz *et. al*, 2003; NS, 2007; OPNET, 2007; SFNET-Java, 2007; SSFNET-C++, 2007; Szymański *et. al*, 2002). There are a number of possible sets of criteria that could be used for simulator comparison, e.g., the model size, execution time, memory requirements, scalability, programming interface, etc. Different tools are optimized for different purposes. For example, comparative studies of some popular network simulators are reported in (Małowidzki, 2004; Nicol, 2003).

We are involved in the development of large heterogeneous systems simulating in near real time. In the case of complex real-life systems, it is natural to model these systems as a set of computing processes which then can be handled by distributed machines or processors. For the last two decades, parallel and distributed simulation has been an active research area (Chen and Szymański, 2002; Niewiadomska-Szynkiewicz *et. al*, 2003; Zeigler *et. al*, 2000). Distributed simulations not only reduce the computation time and permit to execute large programs which cannot be executed on a single machine, but they first of all reflect better the structure of the physical system to be simulated, which usually consists of several components.

Parallel and distributed discrete-event simulations can be described in terms of logical processes (LPs) that communicate with each other through message passing. LPs simulate real life physical processes FPs. Each logical process starts processing as a result of event occurrence (from the event list or having received a new message). It performs some calculations and usually generates some local events and/or messages to other processes.

In parallel simulations each logical process maintains its own local clock (LVT, i.e., Local Virtual Time). Local times of different processes may advance asynchronously. Events arriving at the local input message queue of a logical process are executed according to the local clock and the local schedule scheme. Synchronization mechanisms fall into three categories: conservative, opti-

mistic and lookback-based. They differ in their approach to time management. *Conservative* schemes, described in (Misra, 1986; Nicol and Fujimoto, 1994; Niewiadomska-Szynkiewicz and Sikora, 2004; Zeigler *et. al*, 2000), avoid the possibility of causality error occurrence. These protocols determine safe events that can be executed. *Optimistic* schemes (Jefferson, 1985; Nicol and Fujimoto, 1994; Niewiadomska-Szynkiewicz and Sikora, 2004; Zeigler *et. al*, 2000) allow for the occurrence of causality errors. They detect such errors and provide mechanisms for their removal. The calculations are rolled back to a consistent state by sending out antimessages. It is obvious that, in order to allow rollback, all results of previous calculations have to be recorded. *Lookback* schemes (Chen and Szymański, 2002; Chen and Szymański, 2003) permit to consider only causality errors, which require rollback but no antimessages, so they fall in between conservative and optimistic schemes.

2. Paradigms for Constructing Simulators

There are two basic directions when developing parallel and distributed large scale systems simulators (see (Ferenci *et al.*, 2000)):

- development of problem dedicated (specialized) simulators, specific to the environment for which they were created,
- development of general purpose simulators, designed as federations of disparate simulators, utilizing runtime infrastructure software (RTI) to interconnect them.

In the case of the first paradigm, the simulation engine, interface, libraries and tools to create new high performance simulators are defined. It is difficult, in general, for the user to modify and apply such software to a new environment.

The second paradigm results in a collection of disparate simulators (simulation entities) designated as federates. All simulators involved in a federation are viewed as black boxes. Runtime infrastructure software used for interconnecting simulators is typically designed for coarse granularity concurrence. RTI implements relevant services required by the relevant federated simulation environment. The most important services are: the synchronization of simulators, secure and efficient communication and a scalable platform architecture. An example of a federated simulation with four entities simulating four subsystems or parts of a simulated physical system is presented in Fig. 1.

An approach for federating simulators is utilized in the high level architecture (HLA) (HLA, 2007), a standard for distributed discrete-event simulations. The main advantage is a high possibility of simulation model reuse. However, we pay for this universal applicability. This approach imposes certain restrictions concerning the structure of the federation members. In addition, federates have to obey some rules that are included in the federation in order to enable interactions between the federates. In this paper we describe our approach to parallel and distributed federated simulations developed based on the software tool ASimJava.

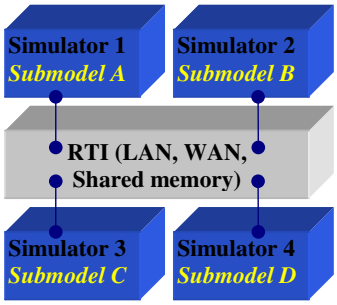

Fig. 1. Architecture of a federated simulator.

3. Asynchronous Simulation Java (ASimJava) Framework for Simulation

3.1. Design Overview. ASimJava (Asynchronous Simulation Java) is a Java-based framework for the simulation of large-scale physical systems. The ASimJava general structure enables discrete-event simulations that can be described in terms of logical processes that communicate with one another through message passing. LPs simulate real life physical processes.

Two of ASimJava's principal goals were portability and usage in heterogeneous computing environments. Two versions of ASimJava are implemented: parallel and distributed. It is possible to develop one simulator combining both of them. The combined parallel and distributed simulation is presented in Fig. 2.

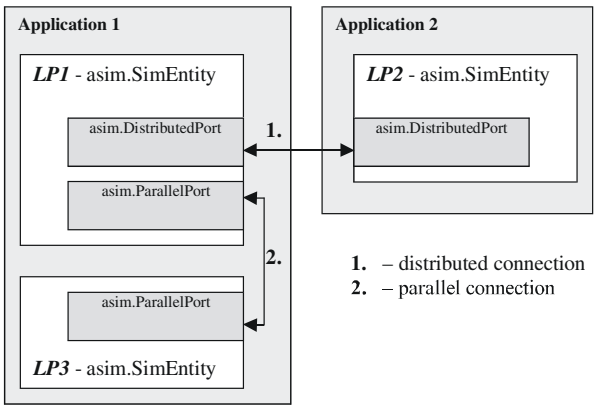

Fig. 2. Combined parallel and distributed simulation.

The JXTA technology platform provided by Sun Microsystems was used for inter-process communication in

the case of a distributed version of ASimJava. JXTA is a set of open, generalized peer-to-peer (P2P) protocols that allow any connected device in the network to communicate and collaborate as peers (see Fig. 3). JXTA protocols are independent of the programming language. Multiple implementations are provided for different environments. This technology enables developers to build and deploy interoperable P2P services and applications. JXTA protocols standardize the manner in which peers discover one another, selforganize into peer groups, advertise and discover network services, securely communicate with one another and remotely monitor one another.

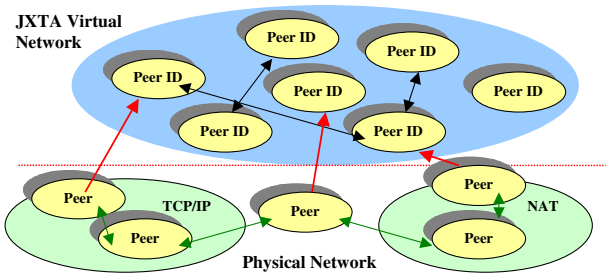

Fig. 3. JXTA logical network mapping.

Synchronous and asynchronous variants of simulators are available. In the case of an asynchronous realization, four protocols managing calculations (conservative, optimistic and hybrid) are provided: conservative protocol with null messages (CMB) (Misra, 1986), the window conservative protocol (WIN) (Nicol and Fujimoto, 1994), the optimistic Time Warp (TW) (Jefferson, 1985) and the hybrid Moving Time Window protocol (MTW) (Sokol *et al.*, 1988).

The current version of the ASimJava framework is composed of five components:

- Basic Library – a collection of classes implementing the basic elements of a simulator, such as logical processes, events, event lists, message passing, etc.;
- Synchronization Protocols Library – the library of classes implementing four synchronization algorithms (CMB, WIN, TW and MTW);
- Runtime Infrastructure – the library of classes that provides communication between the calculating processes;
- ASimL language – the XML Schema specification for building an XML file that contains the description of the system to be simulated;
- Toolboxes – collections of classes implementing the basic elements of various physical systems. Currently there is available the Computer Network Toolbox, which is a collection of classes implementing elements of computer networks, e.g., network devices (routers, hubs, switches, etc), protocols (IP, Frame Relay, ATM, etc.). The ASimJava package is flexible and can be easily extended by other toolboxes of classes which are specific to a chosen case study.

3.2. Simulation under AsimJava. During a simulation experiment performed with ASimJava one can distinguish two main stages: the preparatory one and the experimental one. At the preparatory stage the model of the system to be simulated is investigated and implemented based on ASimJava classes. Each simulator that is built upon ASimJava classes has a hierarchical structure. The simulated system is partitioned into several subsystems (subtasks), with respect to functionality and data requirements. Each subsystem is implemented as a logical process. Each LP can be divided into smaller LPs. Hence, the logical processes are nested, see Fig. 4. Computing processes belonging to the same level of hierarchy are synchronized. The approach to synchronization depends on the chosen version: the global clock in synchronous simulation and one of the above-mentioned four protocols (CMB, WIN, TW or MTW) in asynchronous simulation.

Consequently, the application developed based upon ASimJava classes is a federation of simulators (logical processes responsible for the simulation of physical subsystems) that are connected through the RTI infrastructure. Each simulator can be easily reused in many other experiments.

Two types of simulators can be distinguished:

1. A simulator consists only of classes provided in ASimJava. The structure of the simulated system together with all model parameters is read from an XML file. We can simply run the simulator writing the command line:
   ```
   java <application_name>  -f  <file_name.xml>
   ```
2. A new simulator. The user's task is to implement the subsystems' simulators responsible for the simulation of adequate physical subsystems. He or she can create this application applying adequate classes from the ASimJava framework and including his or her own code, i.e., the numerical part of the application. The ASimJava classes provide communication between running processes, their synchronization and an interface between the user applications and the RTI infrastructure. This allows the user to focus on the numerical part of the application.

To support the process of defining the analysed user application (simulator type 1), a bidirectional interface to the XML configuration and the state save file is provided. The ASimL language based on the XML Schema (www.w3.org standard) specification for building the XML file with a description of the parameterized system model was developed. It consists of several

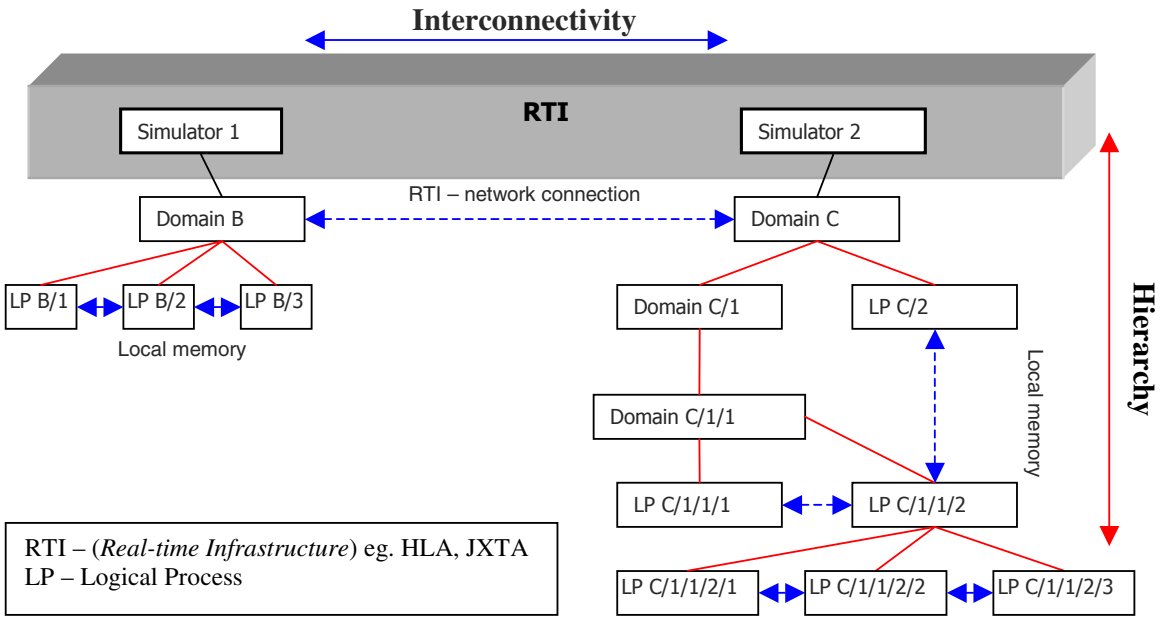

Fig. 4. Federation of simulators consisting of two members: "Simulator 1" and "Simulator 2".

defined keywords, such as simulator – the user application, logicalProcess – a logical process, domain – a group of logical processes, port – a communication port, link – a link between two ports, message – a message with events to be served, eventService – a module that services an event, parameter Type – the type of any parameter. The part of an exemplary XML file defining the model of the system to be simulated in the ASimL language is presented in Fig. 5.

```
<logicalProcess name='LP2" engine="Proc_WIN">
      <port name="LP2_port" communicationType
       ="Parallel" />
      <parameter name="timeSEND1" value
      ="3.0" />
      <parameter name="portSEND1"
       value="LP2_port" />
      <eventService messageType="SEND1"
       serviceClass="SEND"
/>
   </logicalProcess>
<link srcStrictPortName="Test simulator/Domain 1
/LP1/LP1_port1"
    dstStrictPortName="Test simulator/Domain 1/
    LP2/LP2_port1"/>< link
srcStrictPortName="Test simulator/Domain 1
/LP2/LP2_port1"
    dstStrictPortName="Test simulator/Domain 1/
    LP1/LP1_port1"/>
```

Fig. 5. Sample file with the description of the simulation model using ASimL.

The simulation model can be fully loaded (run, rerun) from the XML file. It may contain any number of user-defined parameters.

The experimental phase begins when all decisions regarding the simulated system are made. The simulation starts. The adequate calculating processes corresponding to the physical subsystems are executed. The results of the calculations are recorded into the disc file during the experiment. The user decides which data should be collected.

4. Case Study Results

The ASimJava framework was used to perform the simulation of several physical systems. In this paper, applications to a simple manufacturing system and computer networks are presented. The objective of the first series of tests was to compare the effectiveness of the synchronization protocols provided in the framework. The second case study, i.e., a Frame Relay network simulator illustrates the effectiveness of the simulator implemented based on the ASimJava classes.

4.1. Comparative Study of Synchronization Protocols. The first case study, i.e., the simulation of a sample manufacturing system (Example 1) and an IP network (Example 2) was performed to compare the effectiveness of four synchronization protocols provided in ASimJava. The first series of experiments were related to the simulation of a simple distributed manufacturing system, as presented in Fig. 6. The system considered consists of two sources $P1$ and $P2$, eight workstations $P3$–$P10$ and

one sink $P11$. Jobs enter the manufacturing system at the workstations $P3$ or $P4$.

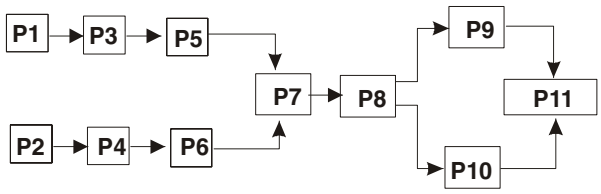

Fig. 6. Example 1: Manufacturing system.

When a job has been serviced at a workstation Pi, it proceeds to the next workstation. Service times at different workstations are different. Jobs may be queued at a station awaiting service. A workstation takes one job from its input queue when it is free, services that job, and then sends it to the queue of the following workstation. All workstations service the jobs in the first-come, first-served order. The job leaves the system after being serviced at the workstation $P9$ or $P10$. It is collected at the sink ($P11$).

The sources, the sink and all workstations were simulated by 11 logical processes. It was assumed that after entering a job with the identifier j at a workstation Pi at time t, its service began either immediately (at time t), if Pi was idle, or it began right after the departure of the job k from this Pi, where $k = \mathrm{Pred}(i, j)$ denotes the identifier of the job that had been executed at a workstation Pi just before j entered it. Let t_{A_j} be the time of the arrival of the job j at Pi, t_{D_k} the time of the departure of the job k from Pi and Δt_{ij} the service time of the job j at this Pi. Then we obtain

$$t_{D_j} = \max(t_{A_j}, t_{D_k}) + \Delta t_{ij}. \tag{1}$$

Simulation experiments were performed under the following assumptions:

- Two variants of application were considered: *Variant A* (each source generated 2 jobs), and *Variant B* (each source generated 25 jobs). The service times at different workstations are given in Table 1.

Table 1. Service times (two variants of application).

Variant	$P1$	$P2$	$P3$	$P4$	$P5$	$P6$	$P7$	$P8$	$P9$	$P10$
Variant A	2	3	2	3	3	3	7	5	3	3
Variant B	2	5	7	2	2	8	9	5	7	1

- The experiments were performed in the network of four Celeron 433 computers. The allocation of LPs simulating adequate physical processes to the computers was as follows: *Computer 1*: $LP1$, $LP3$, $LP5$; *Computer 2*: $LP2$, $LP4$, $LP6$; *Computer 3*: $LP7$, $LP8$; *Computer 4*: $LP9$, $LP10$, $LP11$.

The results of the numerical experiments are presented in Tables 2 and 3. Different aspects were considered with respect to the synchronization protocols applied:

- time of simulation,
- number of additional messages sent by computing processes (CMB – null messages, WIN – global messages used for the calculation of time window lengths, TW – antimessages, MTW – antimessages and additional messages used for synchronization),
- number of rollbacks at TW,
- different lengths of the time window at MTW.

Table 2. Simulation results — execution times of experiments.

Application	CMB [s]	WIN [s]	TW [s]	MTW [s]
Variant A	247,83	25,81	34,55	20,93
Variant B	316,86	88,43	52,34	41,25

Table 3. Simulation results for different lengths of time windows (MTW algorithm, *Variant B*).

Time window size	Simulation time [s]	Additional messages	Number of rollbacks
5	54,87	134	0
10	45,09	71	1
20	41,25	36	1
30	44,87	33	2
40	63,82	34	0
50	59,87	23	1

It can be observed (see Table 2) that the speed of simulation strongly depends on the protocol applied. The best results (decreased execution times) were obtained for the hybrid approach MTW, and the worst for the conservative CMB. This was connected with different degrees of parallelism in the case of conservative and optimistic approaches, as well as overheads (additional messages and rollbacks).

The second case study included a series of experiments related to the simulation of an IP network consisting of five LANs and one server room (13 servers), as presented in Fig. 7. Two scenarios of traffic with small packets of transmitted data (*Variant A*) and large packets (*Variant B*) were considered.

The experiments were performed in a network of four Celeron 433 computers. The allocation of logical processes simulating the analysed computer network to the computers was as follows:

- Computer 1: $LAN1$, $LAN2$;
- Computer 2: $LAN3$, $LAN4$;
- Computer 3: $LAN5$;
- Computer 4: server room.

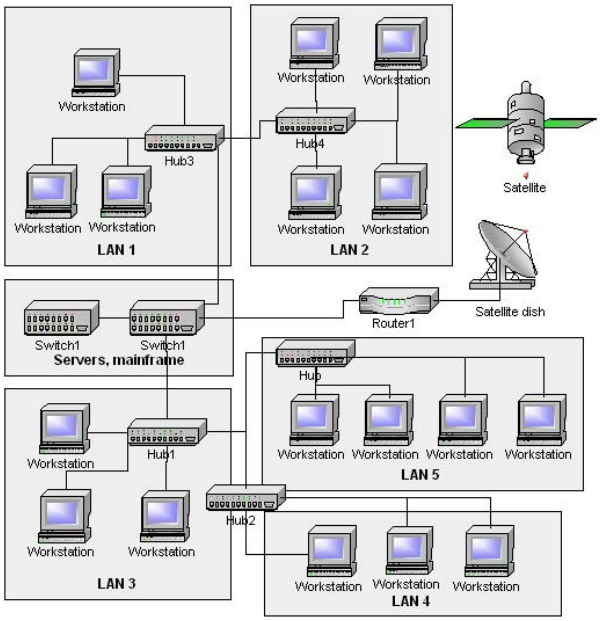

Fig. 7. Example 2: Computer network.

The results obtained for two synchronization protocols (the conservative WIN and the hybrid MTW) are presented in Table 4. Two lengths of the window in MTW (1 time unit and 10 time units) were tested.

Table 4. Simulation times for the computer network.

Application	WIN [ms]	MTW–1 [ms]	MTW–10 [ms]
Variant A	18186	18090	15742
Variant B	72072	71097	63996

The application of CMB and TW synchronization protocols produced much worse results, as the computation time increased considerably. Similarly to Example 1 (the manufacturing system), the best results were obtained for the hybrid protocol. The hybrid techniques seem to be promising with respect to the optimistic TW. It may be profitable to decrease the degree of the available parallelism, increase the number of additional messages but reduce the number of rollbacks. In the case of combining TW and window techniques, the main problem is to estimate an optimal length of the window. The simulation results strongly depend on this parameter, see Table 3 and 4. It seems that the length of each window should be calculated adaptively, taking into account the application considered and the available hardware platform, see (Sokol *et al.*, 1988). Accordingly, the degree of parallelism reduction remains the topic of a hot debate.

4.2. Frame Relay ASimJava Simulator. In the second case study we evaluated the complexity of Frame Relay networks simulation. Frame Relay is a high-performance WAN protocol that operates at the physical and data link layers of the OSI reference model (McCabe, 2003). This is a standard protocol for LAN internetworking which provides a fast and efficient method of transmitting information from a user device to LAN bridges and routers. Frame Relay is an example of a packet-switched technology. Variable-length packets are used for more efficient and flexible data transfers. These packets are switched between various segments in the network until the destination is reached. The Frame Relay traffic is described based on several characteristic parameters. Detailed information about FR parameters can be found on the Frame Relay Forum website (Frame Relay Forum, 2007).

The simulators of four network systems (Examples $E1$–$E4$) describing different network sizes and configurations were implemented upon ASimJava. Detailed descriptions, i.e., network models and traffic characteristics are given in Table 5.

The goal of all experiments was to simulate 30 seconds of physical network operation. The efficiency of the parallel federated simulation was compared with the sequential realization.

To compare the performance of packet-level simulators, we used two characteristics, i.e., the simulation time (the time of the experiment execution) in miliseconds and the average simulator speed PTS (simulated packets transmissions per second) defined as follows (Fujimoto *et al.*, 2003):

$$PTS \approx \frac{N_F P_F H_F}{T}, \tag{2}$$

where T denotes the execution time, N_F stands for the number of flows (edge router to edge router), P_F is the number of packets sent per flow, H_F signifies the average hops per flow (queuing, transmitting over link, etc.). The presented definition ignores lost packets and additional messages generated by the protocols.

All simulations were performed in a network of PC computers. Three series of experiments were executed:

- $C1$ – sequential simulation: one machine (AMD Athlon-M 1.2 GHz, 512 RAM),
- $C2$ – distributed simulation: two machines (AMD Athlon-M 1.2 GHz, 512 RAM and AMD Sempron 1.67 GHz, 512 RAM),
- $C3$ – distributed simulation: three machines (AMD Athlon-M 1.2 GHz, 512 RAM, AMD Sempron 1.67 GHz, 512 RAM and AMD Sempron 1.67 GHz, 512 RAM).

The results of the experiments for four network configurations (Examples $E1$–$E4$) performed on a single machine (Variant $C1$) are presented in Table 6.

It can be observed that the execution time of the experiment performed for the network $E4$ exceeds the real

Table 5. Tested Frame Relay network models.

E1	E2	E3	E4
1 switch	1 switch	2 switches	3 switches
2 interfaces	6 interfaces	14 interfaces	22 interfaces
2 edge routers	6 edge routers	12 edge routers	18 edge routers
2 links 1.544 Mb/s	6 links 1.544 Mb/s	12 links 1.544 Mb/s	18 links 1.544 Mb/s
		1 links 44.736 Mb/s	2 links 44.736 Mb/s

Table 6. Results of experiments: four sample Frame Relay networks (sequential version).

Example	LPs number	Number of packets	Simulation time [ms]	PTS
E1	7	$(\sim 13500 \times 2) = 27000$	4800	22.5
E2	19	$(\sim 13500 \times 6) = 81000$	12900	25.1
E3	42	$(\sim 13500 \times 12) = 162000$	24300	46.6
E4	65	$(\sim 13500 \times 18) = 243000$	34100	71.2

Table 7. Results of experiments: two sample Frame Relay networks (distributed versions).

Example		Number of LPs	Number of packets	Simulation time [ms]	PTS
model	decomposition				
E3	*SIMULATOR 1* 1 switch, 7 interfaces 6 edge routers 6 links 1.544 Mb/s 1 link 44.736 Mb/s	21	$(\sim 13500 \times 12) = 162000$	19300	58.7
	SIMULATOR 2 1 switch, 7 interfaces 6 edge routers 6 links 1.544 Mb/s	21			
E4	*SIMULATOR 1* 1 switch, 7 interfaces 6 edge routers 6 links 1.544 Mb/s 1 link 44.736 Mb/s	21	$(\sim 13500 \times 18) = 243000$	19300	125.2
	SIMULATOR 2 1 switch, 7 interfaces 6 edge routers 6 links 1.544 Mb/s 1 link 44.736 Mb/s	21			
	SIMULATOR 3 1 switch, 8 interfaces 6 edge routers 6 links 1.544 Mb/s	23			

time operation of the physical network (the simulation time is greater than the simulated time, i.e., 30 seconds).

Next, two series of experiments for distributed implementations were performed. Two exemplary networks $E3$ and $E4$ were taken into consideration. In both cases the simulators of the whole networks were partitioned into several federated simulators: two simulators in Example $E3$ and three simulators in the case of $E4$ (see Fig. 8).

A detailed description of all subnetwork configurations is presented in Table 7. The calculations of each

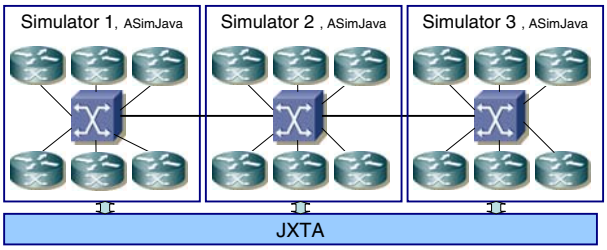

Fig. 8. Federated simulator of Network $E4$.

member of the federation were performed by a separate computer. The window conservative scheme (WIN) described in (Nicol and Fujimoto, 1994; Niewiadomska-Szynkiewicz and Sikora, 2004) was applied to the synchronization of federated simulators. The execution time of each experiment and simulator speeds are given in Table 7.

We can observe that federated, distributed simulations developed based on ASimJava can seriously speed up simulations of network operation compared with sequential implementations. The calculation speed-up depends on the size of the network model considered and the assumed degree of parallelism. It should be indicated that, in the case of a distributed implementation, the reserve of efficiency to meet real time requirements is quite large, see Example $E4$ in Table 7.

5. Summary

We have described ASimJava, a software framework suitable to solve many small and large scale problems, based on simulation. The focus was on a federated approach to parallel and distributed simulation. We demonstrated that this approach is suitable to perform fast simulations of large-scale systems. Our experience with federated, distributed network simulations confirms the ability of the federated simulation approach to execute large simulation models. The ASimJava framework is useful in all applications in which the speed of simulation is of essence, such as real time simulation.

References

Chen G. and Szymański B.K., Lookback (2002): *A new way of exploiting parallelism in discrete event simulation*. — Proc. 16-th Workshop *Parallel and Distributed Simulation*, Washington, DC, pp. 153–162.

Chen G. and Szymański B.K. (2003): *Four types of lookback*. — Proc. 17-th Workshop *Parallel and Distributed Simulation*, San Diego, USA, pp. 3–10.

Ferenci S.L., Perumalla K.S. and Fujimoto R.M. (2000): *An approach for federating parallel simulators*. — Proc. 14-th Workshop *Parallel and Distributed Simulation (PADS 2000)*, Bologna, Italy, pp. 63–70.

FRF (2007) Frame Relay Forum, `http://www.frforum.com`.

Fujimoto R.M., Perumalla K., Park A., Wu H., Ammar M.H., Riley G.F. (2003): *Large-scale network simulation: How big? How fast?*. — 11-th IEEE Symp. *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Los Alamos, USA, pp. 116–125.

HLA (2007): High Level Architecture homepage, `http://www.dmso.mil/public/transition/hla/`.

Java Sim (2007) JavaSim homepage, `http://www.javasim.org`.

Jefferson D.A. (1985): *Virtual time*. — ACM Trans. Programm. Lang. Syst., Vol. 7, No. 3, pp. 404–425.

Małowidzki M. (2004): *Network simulators: A developer's perspective*. — Proc. Int. Sym. *Performance Evaluation of Computer and Telecommunication Systems (SPECTS'04)*, San Jose, USA, pp. 1–9.

McCabe J.D. (2003): *Network Analysis, Architecture, and Design*. — New York: Morgan Kaufman.

Misra J. (1986): *Distributed discrete-event simulation*. — Comput. Surveys, Vol. 18, No. 1, pp. 39–65.

Nicol D.M. and Fujimoto R. (1994): *Parallel simulation today*. — Anna. Oper. Res., No. 53, pp. 249–285.

Nicol D.M. (2003): *Utility analysis of network simulators*. — Int. J. Simul.: Syst. Sci. Technol., Vol. 4, No. 3–4, pp. 55–69.

Niewiadomska-Szynkiewicz E., Żmuda M., Malinowski K. (2003): *Application of Java-based framework to parallel simulation of large-scale systems*. — Appl. Math. Comput. Sci., Vol. 13, No. 4, pp. 537–547.

Niewiadomska-Szynkiewicz E. and Sikora A., AsimJava (2004): *A Java-based library for distributed simulation*. — J. Telecommun. Inf. Technol., No. 3, pp. 12–17.

(OPNET, 2007) OPNET Modeler homepage `http://www.opnet.com/products/modeler/home.html`.

Sokol L.M., Briscoe D.P. and Wieland A.P., MTW (1988): *A strategy for scheduling discrete simulation events for concurrent execution*. – Proc. SCS Multiconf. *Distributed Simulation*, San Diego, USA, pp. 34–42.

SSFNET-C++ (2007): SSFNET-C++ homepage, `http://www.cs.dartmouth.edu/ghyan/dassfnet/overview.htm`.

SFNET-Java (2007): SFNET-Java homepage, `http://www.ssfnet.org`.

Szymański B.K., Saifee A., Sastry A., Liu Y., Mandnani K., Genesis (2002): *A system for large-scale paralel network simulation*. — Proc. Int. Conf. *Parallel and Distributed Simulation*, PADS 2002, Washington, USA, pp. 89–96.

NS-2 (2007): The Network Simulator ns-2 homepage, `http://www.isi.edu/nsnam/ns/`.

Zeigler B.P., Praehofer H. and Kim T.G. (2000): *Theory of Modeling and Simulation*. — Orlando, USA, Academic Press.

Received: 6 April 2006
Revised: 5 November 2006