Language Support for Parallel Discrete Event Simulation

Hristo Valchanov[1], Nadezhda Ruskova[2], Trifon Ruskov[3]

Abstract – **Parallel discrete-event simulation (PDES) requires in-depth knowledge of the mapping process from the physical model to the simulation model and underlying synchronization protocols in use. Languages for PDES could significantly reduce the development effort by providing the user with a set of well-defined language constructs for designing simulation models. In this paper we present an object-oriented language for PDES, called SIMOPAL. Basic features of the language and its distributed implementation are discussed.**

Keywords – **Simulation languages, Distributed Simulation, PDES.**

I. INTRODUCTION

For the past few years Parallel Discrete Event Simulation (PDES) has been regarded as a broad field for scientific research. The interest for it has been dictated by the need for increasing the processing speed in the complex systems analysis. The main reasons thereof are two: the use of parallel processors presumes simulation speed increase, on the one hand, and the large memory space of the parallel processors allows for realization of larger simulation models compared to the memory of single-processor machines, on the other hand. Due to the asynchronous nature of events occurrence and the distributed nature of the model's components performance, the main problem with PDES is ensuring of correct events processing with respect to model time. There are two main approaches for complying with this requirement: conservative and optimistic [2]. With both approaches the simulation model is presented as an aggregate of logical processes, each modeling a separate physical process of the modeled system. The logical processes exchange messages for the events taking place at certain moments of the model time.

The transition from sequential to distributed simulation needs to be as smooth as possible. The user has to focus his skills on the modeling process, free of the necessity to know the used synchronization protocols or their influence. This is what the main goal in the design and development of frameworks for PDES is.

A typical feature of these frameworks is the presentation of the PDES means – in the form of simulation language or as a library. The motivation for the development of languages for PDES instead of function libraries is in the fact that the users do not have to be familiar with the used synchronization protocols or the peculiarities of the runtime environment.

II. RELATED WORK

There are a number of requirements to the PDES languages as regards the model of: presentation of the modeled system; the framework; the synchronization protocols and the runtime environment.

The simulation languages require that the user models the physical environment in a certain way. In the terminology of PDES, the program interface by which the user can describe the modeled environment is known as 'world view'. In the PDES languages two main concepts for the 'world view' are used: event scheduling and process interaction.

Under the approach of event scheduling, the development of the model is described as a sequence of events ordered by their time of occurrence. A procedure corresponds to each event. The occurrence of the event is interpreted by executing of the corresponding procedure and has influence on certain variables of the model state. Typical representatives of this approach are the systems for PDES SPEEDS [3], TWOS [4] and WARPED [5].

The second approach considers the modeled system as an aggregate of parallel interacting processes. The actual components of the modeled system may share the same characteristics which allows for them to be combined in a class. The behavior of each class of components is described by a class process. The interaction between the processes is carried out by an exchange of messages interpreting the occurrence of certain events. Representatives of this approach are the simulation languages of the type of APOSTLE, ParSec, ModSim and YADEES [6].

The PDES languages can provide to the user both the traditional structural programming approach and the object-oriented approach. The advantage of structural programming is the simplified realization of the runtime system and the better time-saving characteristics of the model implementation. However, the object-oriented languages allow for the time of applications development to be significantly reduced and an easier modification and a reusing of the program code to be achieved. APOSTLE and ModSim provide to the users an object-oriented environment for development of their applications, whereas Parsec is a representative of the structural languages for PDES, being a language extension of C. YADDES, on its part, is a descriptive language of the type of Lex and Yacc, providing a minimum set of language structures.

The researches in the filed of PDES for the last 15 years have resulted in the development of a series of

[1]Hristo Valchanov is with the Computer Science Department, Technical University, 9010, Studentska Str., Varna, Bulgaria, E-mail: hristo@tu-varna.acad.bg

[2]Nadezhda Ruskova is with the Computer Science Department, Technical University, 9010, Studentska Str., Varna, Bulgaria, E-mail: ruskova@tu-varna.acad.bg

[3]Trifon Ruskov is with the Computer Science Department, Technical University, 9010, Studentska Str., Varna, Bulgaria, E-mail: ruskov@tu-varna.acad.bg

synchronization protocols [1]. Yet no adequate solutions are known allowing the user to choose the appropriate synchronization protocol. The simulation model can very often derive much better efficiency from a synchronization protocol, but with productivity dramatically reduced when using another one. Some of the simulation languages presented are based on the conservative approach while others use optimistic synchronization methods.

The PDES systems developed in the recent years are largely architecturally dependent, which is influenced by their use by a limited area of researches of the scientific community. For instance, APOSTLE has been developed for the Meico CS-2 multiprocessor system. WARPED has been developed for Intel Paragon and a network of Sun Sparc workstations. Parsec has been developed for IBM SP and for a network of Sun Sparc workstations. The spread of local area computer networks (LAN) provides new possibilities for organizing of large computing resources for acceleration of the simulation process as well as for its popularization. LAN are of low cost and provide a possibility for easy reconfiguration unlike the multiprocessor systems with shared memory traditionally used for the purposes of PDES. At the same time LAN use the widely spread hardware and software which makes them accessible to a wide area of users. The distribution LAN based systems allow for effective use of the available unloaded processors and for an increase of the number of computers in the network, creating conditions for the implementation of simulation models with sufficiently high degree of parallelism.

The article present the typical features of an object-oriented language for PDES – SIMOPAL, providing a possibility for organizing shared memory on distributed memory systems, at the same time ensuring transparency as regards synchronization protocols. Its potential for describing of complex systems as well as its realization has been demonstrated.

III. THE LANGUAGE SIMOPAL

In the model of SIMOPAL (SIMulation Object-oriented Paralel Language) the modeled system is presented in the form of object-oriented parallel system, considered as an aggregate of objects and messages. The object is an abstract description of an independent processing element, possessing local memory. The only admissible interaction between the objects is the asynchronous exchange of messages. These typical features of the model fully satisfy the requirements of PDES to the modeling languages in terms of: maximum performance parallelization; adequacy of describing the modeled system taking account of its inherent parallelism; a possibility for applying the object-oriented approach in view of reducing the process of models description; convenient and effective model implementation on various parallel architectures.

SIMOPAL language is based on the concept of ‘world view’ by means of interacting processes, with elements of the other events processing approach added. The modeled system is described as an aggregate of class instances. A class instance simulates a specific physical process of the modeled system. In the ideal case, each process is executed independently of the rest, on a separate processor in the distributed runtime environment. The executing of each instance (process) is determined depending on the occurrence of specific events. The occurrence of an event may result in one of the following three actions: change in the attributes of a process; scheduling of a new event in the simulation model; creation of a class instance.

SIMOPAL program is an aggregate of class declarations and an initiating section. The declarations are used for structural description of the components and features of the modeled program. The initiating section is used for initial establishment and starting of the simulation and is implicitly interpreted by the runtime system as a virtual initial process.

III.1. CLASSES

A class declaration specifies a pattern of the structure (attributes) and behavior (rules on event processing) of its instances (objects). The class instances are created dynamically during the process of simulation and are interpreted by the runtime system as active components (processes). The values of the attributes of each instance reflect its current state while the aggregate of values of the attributes of all instances determines the current state of the entire modeled system. Each attribute may be of a specific type. Due to the experimental nature of the language, only simple data types are supported: integers (**int**), reals (**real**), boolean data (**bool**), symbol data (**char**), strings, object identifiers (**oid**) as well as a structured type – array.

The class declaration includes a declaration of events, too. The events are the main component of the PDES model, reflecting the change in the state of the modeled system. The events are interpreted by the runtime system as exchange of messages between the class instances (processes). A simulation time (timestamp) simulating the moment of occurrence is associated with each event. An event may contain typified parameters. It allows implementation of an information exchange between the class instances by means of the mechanism of events scheduling in the simulation model and their handling by the processes. The events declared within a class have a local scope of visibility and are only accessible by the objects in this class. If any processing of events between objects of various classes is necessary, then these common events should be declared in a common basic inheritable class.

The execution of the simulation model (change in the current state of the simulation system) is the result of the occurrence of a certain series of events during the model time. The simulation process requires that these events are processed by the active components – the class instances (processes). The class declaration includes a pattern of the behavior of the class instances, too, specified by means of behavior rules. Each rule is defined by an **isevent** statement and refers to a single event of specific type. The algorithm of a process functioning comprises a sequential checking for received message of scheduled event by the order of rules defining. In case of coincidence, that is, when the respective rule is satisfied, the process performs the actions described.

Fig. 1 shows a program fragment of *Router* class declaring with two attributes of the *int.* type. The objects of this class may communicate with one another by means of two types of events, performing certain actions depending on the specific event received.

```
typeclass Router () {
   int hops;
   int length;

   event Update { int hop };
   event Resume { };

   isevent (Update u) {
        length = length + u.hop;
        ...
   };
   isevent (Resume r) {
        ...
   };
};
```

Fig.1. Declaration of Router class

During event processing, additional conditional behavior may be introduced by assigning a guard in the **isevent** statement. An event rule **e** with a guard **b** is satisfied if the process has received a message of event **e** occurrence and the result of guard **b** calculation is true. Fig. 2 shows an example of using a guard.

```
   isevent (Update u) when (length > 0) {
        length = length + u.hop;
        ...
   };
```

Fig.2. Using a guard

III.2. EVENT SCHEDULING

The objects behavior and hence the executing of the simulation model are determined by the occurrence of certain events. The events in SIMOPAL are explicitly scheduled by the instances in the simulation environment by means of **deposit** statement. Such explicit scheduling provides a possibility for the user to describe the functioning of the real modeled system more adequately.

By default, each new event is scheduled with a timestamp equal to the current value of the local simulation time of the process. SIMOPAL provides a possibility for the events occurrence time control by inputting a offset against the current local time of the process. Fig. 3 shows the scheduling of a *Resume* event to a process with *next_router* identifier which is to occur after 100 units of simulation time.

```
   deposit ( next_router, Resume, 100 );
```

Fig.3. Event scheduling

SIMOPAL provides a possibility for special scheduling of events (Fig. 4):

- object to itself – the keyword **self** is specified as first argument. This allows for simulation of timeout events or for generators of events;
- to all objects in the simulation system – the value of **nil** is specified as first argument. Simulation of broadcast exchange is allowed;
- to a group of objects in the simulation system – a composition of objects identifiers is specified as first argument by means of + operator. If a class identifier or a composition of class identifiers is specified, depositing of the event will concern all objects of this class or composition of classes.

```
(* Timeout event to self after 100 time units *)
   deposit ( self, Timeout, 100 );

(* Event Transfer to all processes immediately *)
   deposit ( nil, Transfer );

(* Event Route to all objects of class *)
(* OSPF_router and BGP_router *)
   deposit ( OSPF_router + BGP_router, Route );
```

Fig.4. Special scheduling of events

Depositing of events to multiple objects is one of the features of SIMOPAL which has not been realized in any of the above-mentioned simulation languages.

III.3. CLASS INHERITANCE

SIMOPAL ensures multiple class-to-class inheritance. The current class inherits the immediate superclasses of a list. The inheriting allows setting up of class hierarchy. An object of a given class inherits both the structure (attributes) and the behavior (rules) of its superclasses.

Fig. 5 shows an example of relations between three classes.

Class C is subclass of classes A and B. When inheriting, all attributes (*Attr)*, rules *(Rules)* and declarations of events (not shown on the diagram) of its superclasses A and B are added without changes. Any new properties of class C object may be additionally specified in the C declaration. If classes A and B on their part are a subclass of other superclasses, the above inheritance principle will be valid for them, too.

One of the main applications of inheritance is the expansion of the visibility scope of the identifiers in a SIMOPAL program. By default, only the class identifiers are global, while all declarations within their bodies are local. Due to the absence of global data in the distributed simulation model, objects of different classes may exchange information only by means of the events mechanism. The declarations of used events may be set apart in a superclass which on its part is to be inherited by the above-mentioned classes.

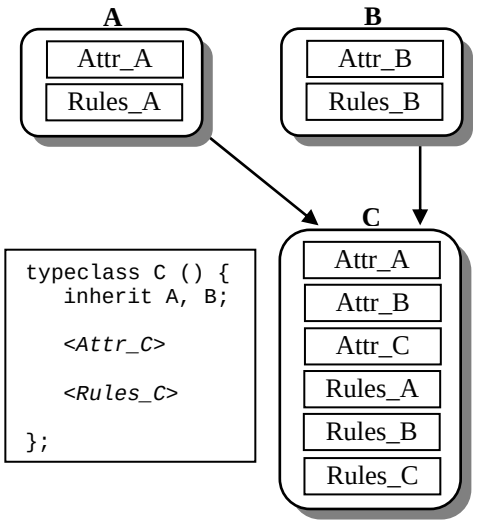

Fig.5. A class inheritance hierarchy

IV. IMPLEMENTATION OF THE LANGUAGE

The distributed runtime system of SIMOPAL language has been realized on a network of workstations, each functioning under the control of OS Linux. Its structure is of cluster organization, with a cluster of logical processes executed by a separate workstation. Each logical process (LP) interprets the functioning of a specific instance of a process defined in the language. The execution of LP in each cluster is controlled by a local dispatcher. Realization of the logical processes within a single Linux process reduces switching of the processes context to a significant extent, which increases the simulation efficiency. The communication between the individual clusters is carried out by a communication subsystem with possibilities for use of both the MPI communication standard and the SCTP protocol [7].

The distributed runtime system is based on optimistic synchronization protocol TimeWarp with aggressive and lazy message cancellation.

V. CONCLUSION

The SIMOPAL language for distributed simulation has the following characteristics:

- object orientation;
- compact and convenient description of the modeled system;
- transparency for the user as regards the distributed nature of modeling allowing for sequential or distributed simulation of the models without changing the program code;
- it is oriented to a wide area of conceptual synchronization models in PDES;
- it is realized on a distributed runtime system with transparency as regards the load balancing of logical processes;
- a possibility for logical organization of a shared memory on a distributed architecture.

In our future work we will focus on the development of graphic user interface and libraries to help accelerate the process of creation of program models in specific areas of application.

REFERENCES

[1] Ferscha A. Parallel and Distributed Simulation of Discrete Event Simulation. In Handbook of Parallel and Distributed Computing, Mc-Graw Hill, 1995.
[2] Fujimoto R.M. Parallel Discrete Event Simulation. Communications of the ACM, vol.33, N10, pp.41-52, 1990.
[3] Steinman J. SPEEDES: A Multiple Synchronization Environment for Parallel Discrete Event Simulation. Int. Journal in Computer Simulation, v.2, N3, pp.251-286, 1992.
[4] Rich D.O., Michelsen R.E. “An Assessment of the ModSim/TWOS Parallel Simulation Environment.” In Proc. of the 1991 WSC, pp. 509-518, 1991.
[5] Dale E., Wilsey P., Timothy J. WARPED Simulation Kernel Documentation, 1995.
[6] Low Y., Lim C. Cai W., Huang S., Turner S. Survey of Languages and Runtime Libraries for Parallel Discrete Event Simulation. Tech. Report, Simulation, pp.1-15,1999.
[7] H. Valchanov, I. Ruskov, N. Ruskova. A Communication Kernel of a Cluster System for Distributed Simulation. In Proc. of the CompSysTech2004, pp.IIIB.19-1 – IIIB19-5 , 2004.