ISSN 0361-7688, Programming and Computer Software, 2013, Vol. 39, No. 5, pp. 242–254. © Pleiades Publishing, Ltd., 2013.
Original Russian Text © R.L. Smelyansky, A.G. Bakhmurov, D.Yu. Volkanov, E.V. Chemeritskii, 2013, published in Programmirovanie, 2013, Vol. 39, No. 5.

Integrated Environment for the Analysis and Design of Distributed Real-Time Embedded Computing Systems

R. L. Smelyansky, A. G. Bakhmurov, D. Yu. Volkanov, and E. V. Chemeritskii

Faculty of Computational Mathematics and Cybernetics, Moscow State University, Moscow, 119992 Russia
e-mail: {smel,bahmurov,dimawolf,tyz}@lvk.cs.msu.su

Received February 2, 2013

Abstract—Problems of analysis and design of embedded real-time systems for controlling complex engineering systems are considered by an example of a DYANA simulation environment and its development at the Laboratory of Computing Systems (LCS), Faculty of Computational Mathematics and Cybernetics, Moscow State University. Special attention is paid to the verification of the conformity of the designed system to the requirements formulated at the early stages of design. The key features of this environment are as follows: the use of a formal model of operation for distributed systems, and the analysis of various aspects of the behavior of a system, both quantitative and logical, by a unified description of systems. The application of the DYANA environment in research and engineering projects over the last 28 years is considered. A comparison with other domestic and foreign results in the field of simulation of real-time distributed embedded computing systems is presented.

DOI: 10.1134/S0361768813050058

1. INTRODUCTION

In this paper, we give a retrospective analysis of the experience of several domestic projects on the design of tools for the analysis and support of distributed real-time embedded systems (DRTESs). A general characteristic of the object of research in these projects (i.e., DRTES) and requirements on the tools are given in the editorial of the present issue of the journal.

The STEND system (1984–1990) [1] was the first such project. The aim of this project was to develop an environment for the observation and measurement of the behavior of distributed programs with the view to assess the performance of these programs on distributed systems with various target architectures, which are different from the instrumental host architecture. A program for the distributed computing system (DCS) under study was executed on an experimental DCS; the operating system provided the collection of a trace and its supply to the simulation model of the target system of the DCS to determine its performance characteristics. In this project, the first domestic object-oriented distributed operating system has been developed.

During 1994–2001, a completely software simulation environment DYANA [2] was developed. The main objectives of research in this project were the development of a mathematical model of the dynamics of a DCS [3, 4] and the implementation of a combined approach to the simulation of distributed systems [5] when both quantitative methods of analysis (time as a measurable quantity) and qualitative analysis (time is order on a set of events) were applied to the same representation/description of the system. In this project, a simulation approach to the development of software was studied for the first time; later this approach was called a model driven approach (MDA); see, for example, [6]. The results of this research were used in several research and development projects for domestic industry and in the international research project DrTESY [7].

Since 2001, research on the design of a hardware-in-the loop (HWIL) testbed[1] for an airborne system has been carried out on the basis of the experience gained during the development of the DYANA environment [8–10]. Some subsystems of the DYANA environment were adapted to the specific character of real-time operation. The designed means have also been successfully applied to the development of real shipborne computer systems.

Since 2010, a simulation environment has been developed on the basis of the experience obtained, and an analysis of a new generation of DYANA-2012 oriented to international standards has been carried out.

The paper is organized as follows. In Section 2, we describe the architecture of the simulation environment (below in the text, Environment), which sum-

[1] In customer's terminology (and in publications), the testbed has a different name: mathematical simulation testbed for on-board systems (MST OBS); however, in fact the HWIL simulation is meant.

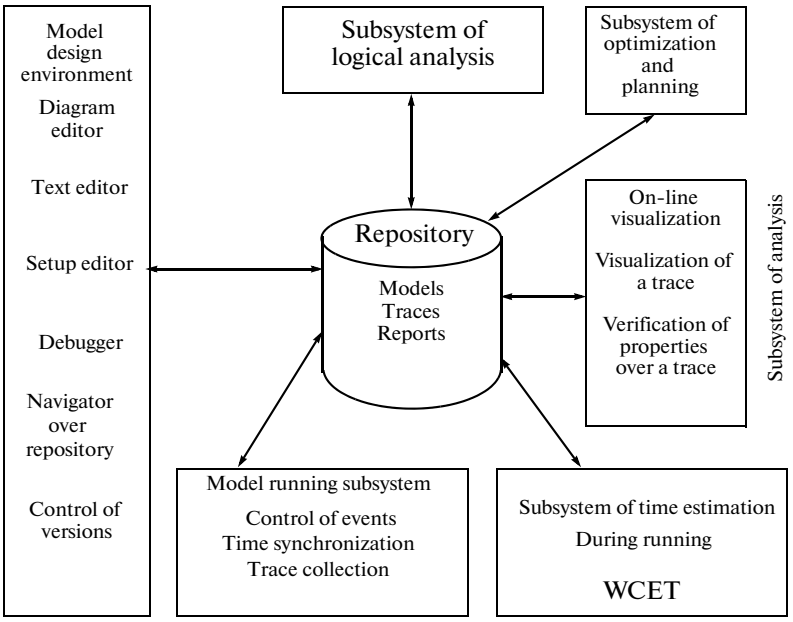

Fig. 1. Structure of software means of a DRTES simulation environment.

marizes the experience gained in the above-mentioned projects, and briefly describe the role of each subsystem. In Sections 3–8, we describe individual subsystems and their evolution from project to project. In the Conclusions, we give a brief summary and a comparison with the projects of other designers and formulate prospective problems of constructing the Environment.

2. GENERALIZED ARCHITECTURE OF ENVIRONMENT FOR THE SUPPORT OF THE SIMULATION AND DESIGN OF A DRTES

The structure of the Environment is shown in Fig. 1.

The source texts of models, intermediate results of preparation of models for execution, execution traces of models, specifications of requirements on the behavior of the system, as well as reports of the tools of analysis for models, are stored in the repository. The grouping of models into projects is supported, and there is integration with the version control system of the source texts.

Environment for the development of models contains the following components: editors of descriptions of models in graphic and text form, an editor of settings of a model in a tabular form, and an interface for navigation over the repository.

Language of the description of models (it is not shown in the scheme) is implemented in the form of description editors and a set of compilers. The principles of construction of the language and restrictions adopted when creating it are described in Section 3.

Language of the specification of requirements on the behavior of the system, whose characteristics are briefly described in Section 7.

Model execution environment is responsible for the data exchange between the components of a model, for binding the events to the model time, or, if necessary, to (astronomical) time, for parallel (on several computers) execution of actions defined in the description of the model, and for the interaction of the models with full-scale DRTES devices. The problems of constructing an execution environment for models are considered in detail in Section 4.

Subsystem of behavior analysis is intended for the determination of the quantitative and qualitative characteristics of behavior of the modeled DRTES. By quantitative characteristics we mean those characteristics of the system that depend on time as on a measurable parameter. By qualitative, or logical, characteristics we mean the properties of the system that depend on time as on a logical parameter, for example, on certain ordering on the set of actions and events in the system. The quantitative characteristics, for example, performance, are evaluated by the results of a simulation experiment. During the experiment, one collects a trace of events, which can later be repeatedly analyzed by a user from various standpoints. The separation of the processes of trace collection and processing and analysis allows one to reduce the number of expensive runs of the model, to uncouple from real time in the analysis, to compare various runs of the model, and geographically separate the place of experiment from the place of analysis. The structure and the problems of the subsystem of quantitative analysis of behavior are described in Section 5.

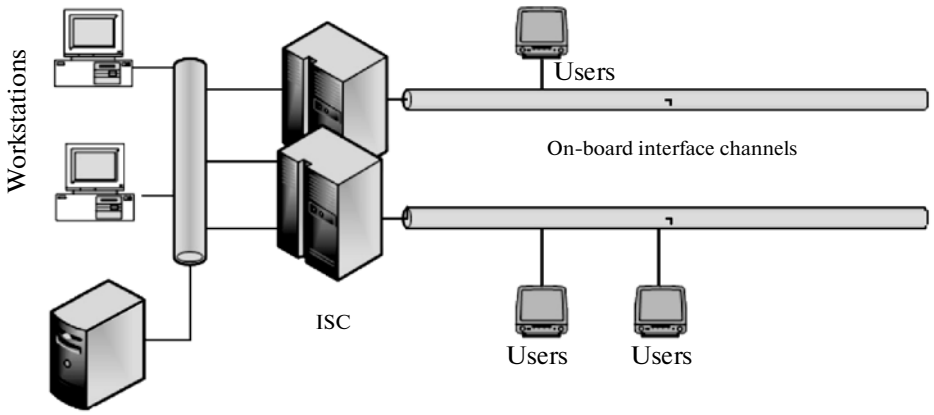

Fig. 2. Structure of hardware means of a DRTES simulation environment.

Subsystem of estimating the time complexity solves the problem of assessing the execution time of a given segment of a program code by a given type of computer. This makes it possible to decouple a program code from the hardware that executes the program. Thanks to this, one can analyze the behavior of the program on different configurations of the equipment. One distinguishes two statements of the problem: time assessment when executing a code block for a specific set of input data, and assessment of the worst execution time over all possible sets of input data. This subsystem is described in greater detail in Section 6.

Subsystem of verification is engaged in solving the problems of qualitative analysis. It allows one to check whether a model possesses some logical properties and, if not, to construct counterexamples in which these properties are not valid. The result of verification can be applied to all possible variants of operation of the DRTES, which is extremely important for validating the correctness of the mission-critical software of the DRTES. The subsystem of verification is described in greater detail in Section 7.

Subsystem of scheduling and optimization supports the choice of a DRTES structure that guarantees the required characteristics of performance and the construction of a schedule for executing computational tasks or operations of data transmission through data exchange channels. Specification of the problems of this subsystem and the order of integration into the Environment are given in Section 8.

Subsystem of analysis of fault tolerance and reliability allows one to assess the reliability of the (projected) DRTES for a given failure flow with regard to the operation of fault-tolerance mechanisms and their effect on the real-time mode, as well as to optimize the configuration of the above-mentioned mechanisms, provided that the deadline times of execution of DRTES tasks remain unchanged. The analysis of this subsystem falls outside the scope of the present paper; it is carried out in [11].

Figure 2 represents a generalized structure of the hardware of the Environment with regard to the needs of HWIL simulation. The environment for the design of models operates on one or several *automation workstations of experimenter engineer*. The model repository is situated in the *repository server*, which is accessible to all the computers of the Environment through a local-area network. One or several *instrumental computers for simulation (ICSs)* are used to run a model. The ISCs are connected to *channels of on-board interfaces* (the number and the types of these interfaces are determined by the specific features of the designed DRTES). In turn, the channels of on-board interfaces are connected to the devices of the on-board equipment presented in the full-scale form. The models of missing devices are run on ISCs; accordingly, a subsystem for running models should be installed on the ISCs. To provide a real-time data exchange between ISCs, these computers are connected by an additional *real-time local-area network*. (The problems of time control in a distributed environment are considered in more detail in Subsection 4.4.)

3. MODEL DESCRIPTION LANGUAGE

As a rule, one of the following approaches is used when developing means for describing simulation models [12]:

• construction of a new domain-specific programming language that supports simulation, for example, control of time; unified interaction mechanisms between the components of a model; automatic tracing of an experiment; etc.;

• writing special modules (class libraries and macro definitions) that solve the problem of supporting simulation for a general-purpose language.

The application of an already existing language allows one to simplify the development of models due to the fact that the designers need not study a new language and can apply existing software (such as the development environment, a translator, and a debug-

ger) and existing libraries of software components. Such an approach was used in the first simulation model STEND, which was developed by our team. As the simulation language for this system, we used C.

At the same time, general-purpose languages either do not allow or allow a restricted control of how the designer observes the requirements of simulation discipline (mechanisms of control of the model time, methods of interaction between the components, language constructions used—"projection" of the Dijkstra programming discipline [13] to the development of a software model); this fact complicates the possibilities of formal logical analysis. These languages cannot be applied to the implementation of the method of development of a model and a code through successive refinements [14].

In contrast, domain-specific simulation languages allow one to rigorously specify a simulation discipline, owing to which a formal analysis of a model within a given simulation scheme is simplified and becomes possible. A special simulation language allows one to implement an approach to the development of a model by successive refinements and can take into consideration almost all the requests of specialists who will work with this language. An example of such a special language is given by the MM language [15], which was developed for the DYANA simulation environment.

The MM language is based on a system with messages [16]. It is proved that a system with messages, as an algorithmic system, is equivalent to Petri nets [17, 18]. The list of algorithmically solvable problems within Petri nets and their modifications, in contrast to systems that can be reduced to Turing machines, most fully corresponds to the list of problems that arise in the analysis of DRTESs [5]. Therefore, the main advantage of such a language is the use of a unified description for the algorithmic and quantitative analysis of DRTESs.

The MM language of the DYANA simulation environment represents an extension of the C language by the means for the description of a DCS. These means allow one

• to describe software and hardware components of the system, as well as the connection of software components to hardware ones;

• to describe the structure of messages used during the interaction of components;

• to describe the internal operation of each process in the system and its interaction with other processes;

• to develop a model by the method of successive refinements.

There are two types of software components in the MM language: sequential processes and distributed programs, and two types of hardware components: sequential and distributed executors. A process is defined by the algorithm of its operation and interaction with other processes. A sequential executor contains information on its time characteristics. A distributed program represents a family of processes and/or other programs, and a distributed executor represents a family of sequential executors and/or other distributed executors. In the body of a sequential process, the algorithm of its operation and interaction with other processes is described. The body is described in a certain extension of the C language, and its syntax resembles the description of the C function. The MM language describes the types of components, and the instances of these types are used subsequently. The description of a type of a component consists of a header and a body. The header defines the type of a component (a process, a program, or a sequential or distributed executor), the name of the type of a component, the list of its parameters, and the list of buffers (the list of "pins" for hardware components) through which the component exchanges information with other components.

Naturally, the advantages of the MM language were obtained at the expense of degradation of other properties of the language, for example, the modularity, impossibility to include algorithms of operation in the C++ language into the description text, as well as external libraries. For example, one cannot divide the body of a process into several functions in the MM language.

Since there were no problems of qualitative analysis and the support of the design method through sequential refinement of a code in the hardware-in-the-loop (HWIL) simulation testbed, but the main problem was to guarantee real-time operation, it was decided to develop a new language—a model description language (MDL). The MDL is an extension of the C programming language. The MDL contains a number of special concepts that are used to describe the properties and the operation logic of the modeled devices. These concepts include particular models (PMs), operation modes of the PMs, the parameters of the PMs, interfaces, timers, signals, and waiting points for signals. To each concept, there correspond elements of the MDL described by means of MDL statements. The set of the MDL statements reflects the specific character of the application domain of the MDL and is intended for the description of the models of DRTES devices, timing the operation algorithms of models, and the organization of their interaction with regard to the specific features of data transmission through multiplex channels. At the same time, the direct advance of the model time in the MDL is inaccessible, in contrast to the MM language.

Despite of its detailed work-up, the MDL turned out to be too bulky for practical application. Engineers should overcome a rather high entrance threshold to start writing models in MDL. To develop a new model, one should perform a number of routine procedures on the creation of bodies and headers of model components. Therefore, for the DYANA-2012 environment, we chose the universal modeling language UML

with graphic interface [19] as the simulation language. By means of the diagrams of the UML language, one describes the static structure of a system and the dynamic aspects of its behavior.

4. SIMULATION RUNTIME ENVIRONMENT

4.1. Requirements for the Runtime Environment

In a discrete-event simulation on the logical level, a system is represented as a set of independent components that can change their state, thus giving rise to independent flows of events. The *runtime environment* is used to synchronize these components and to combine their flows into a common simulation trace by which one can analyze the properties of the system formed by these components [20].

The execution environment of a DRTES should support the HWIL simulation mode [8]; therefore, it should be a software–hardware complex. In practice, one often needs simulation experiments in which both hardware tools using a variety of channels for interaction, protocols, and data transmission formats, and software models developed by independent teams with the use of various tools and operating on different principles take part simultaneously. Such a variety forms a set of additional requirements on the interface of the runtime environment.

1. *Completeness.* The interface of the runtime environment should be complete in order to exclude the need of direct interaction between the components of the model, bypassing the runtime environment.

2. *Universality.* The runtime environment should have standard program interfaces suitable for including a wide range of existing and potential participants of simulation, i.e., both software simulation models and hardware tools (we call them full-scale components).

3. *Transparency.* Physical devices connected to the runtime environment should not distinguish between the interaction with the runtime environment and the interaction with the hardware whose operation the runtime environment simulates.

The satisfaction of the above-listed requirements and, first of all, transparency, implicitly suggests that the runtime environment of DRTES models has a number of additional qualities:

4. *Predictability.* The runtime environment should allow the control and measurement of the time parameters of all the actions of the modeled system.

5. *Performance.* The performance of the runtime environment should possess all the properties of a real-time system; for example, it should ensure the same response time of software models as that of the hardware simulated by these models.

6. *Flexibility.* The runtime environment should be able to operate in different configurations of hardware systems, including distributed systems and computer networks.

7. *Scalability.* The hardware system in which the runtime environment operates should allow the increase of its performance without any significant reconfigurations of the hardware part of the runtime environment. In turn, the runtime environment should be able to optimally plan and distribute available resources.

Below in this section we consider the main problems that arise during the construction of a runtime environment satisfying the above-described requirements.

4.2. Interface of a Runtime Environment

One of important problems of simulation that is also characteristic of programming as a whole is the problem of reuse of simulation models. However, the solution of this problem in practice becomes complicated due to the need to adjust the model interface to its new environment, i.e., to the models with which it has to interact. In the field of program engineering, this problem is efficiently solved with the help of service-oriented architectures, which break the dependence between individual components of a system, providing them a necessary set of services. A similar approach can also be applied to the development of a runtime environment: it has to provide a set of services to the participants of simulation, that is sufficient to describe the behavior of these participants and eliminate their direct interaction with each other.

For constructing DRTES models, the services of the runtime environment should break not only interface, but also logical dependence between individual components of a model. An example of such an interface is given by the service of data transmission according to the "publisher–subscriber" scheme. The publisher announces data of a given type and transfers them to the runtime environment, and the runtime environment transfers the data to all the subscribers who have requested this type of data. This mechanism allows one to easily change both publishers and subscribers without making changes in the simulation participants. The above-listed ideas were reflected in known standards [21, 22]. However, in the field of HWIL simulation, today there is no real-time simulation of such standards. There have been attempts to construct appropriate runtime environments on the basis of the high level architecture (HLA) simulation standard [23]. But this standard requires improvement in order to be used in real-time simulation [24]:

1. HLA does not provide an interface for defining parameters as a service; therefore, the participants of simulation have no right to require that the runtime environment should, for example, process an event within a given time.

2. The standard does not provide mechanisms for controlling the resources of the operating system; therefore, the operating system can, for example, suddenly download the pages of simulation processes

from the main memory, or give higher priority to other existing tasks in using the processor time.

3. HLA supports only two strategies to provide a quality of service (QoS) for data exchange: reliable and unreliable exchange (usually implemented by means of TCP and UDP protocols), which is not sufficient for real-time simulation.

Nevertheless, the rate of development and the dynamics of the expansion of HLA to related fields of simulation show that it is more expedient to update the HLA standard to real-time simulation than to develop a new interface of the runtime environment from scratch.

The size and the complexity of the simulation models of modern DRTESs do not allow one to carry out experiments using a personal computer. For a long time, there have been used various parallel and distributed runtime environments [25] for processing such simulation. For example, the runtimes created at the LCS use a cluster of personal computers. When designing a HWIL simulation testbed at the LCS, an additional local area network with a stack of own real-time protocols was developed. However, such an approach is unacceptable when carrying out experiments with the use of geographically remote components of a model.

4.3. Distribution vs Centralization

An important problem in the organization of a distributed runtime environment is sharing its functionality between instrumental computers. For example, important functions of the runtime environment are the support of the coherence of states, the synchronization of the components of a model, and operations of low-level interaction. The method of implementation of these functions has a crucial effect on the performance of the runtime environment. There are known two main approaches to such an implementation: centralized and distributed ones. The distributed implementation of the runtime environment implies that the local components of the environment one per each instrumental computer) are equivalent and self-sufficient [26] and requires the application of complicated algorithms for their coordination. Therefore, designers usually rule out distributed architecture and introduce a centralization of one or other functions, which allows one to centralize, for example, the synchronization of all participants of simulation on the same computer. In this case, the role of local components reduces to the data transmission from a participant of simulation to the *central component* and back. The software architecture employing the central component is called a centralized architecture, and the architecture without the central component is said to be decentralized or peer-to-peer [27]. For example, the HWIL simulation testbed system is a peer-to-peer system, whereas the CERTI system [28] is a clearly pronounced centralized system.

The results of testing the modern distributed simulation systems show that decentralization usually provides better performance [29]. However, both approaches have their merits and demerits; therefore, designers often try to use tradeoff decisions. One of such decisions is a *cascade* architecture of the runtime environment. It means the division of local components into groups and the connection of each group to own *intermediate component* that is central for the members of this group. In turn, intermediate components are connected to the central component as ordinary local components. Repeated application of the described approach allows one to construct several cascades of components of various levels. Each of these components can independently perform synchronization in its group [30].

4.4. Synchronization of Flows of Events

One of the central problems of distributed simulation is the problem of coordination of flows of events from independent components of a model in uniform time [26, 31]. There exist two essentially different solutions to this problem. The first assumes the use of a communication environment in which an earlier sent message is always received earlier than the messages sent later. This can be achieved in uniform clock systems, i.e., in systems where the clocks are synchronized on the physical level in all instrumental computers. In this case, the messages processed by different computers are ordered automatically. For example, the mechanism of common clocks is effectively used in an HWIL simulation testbed system. However, this solution requires hardware support and is unacceptable for networks.

The second approach suggests the support of uniform model time algorithmically. Algorithms for this purpose are divided into conservative and optimistic ones [26]. A conservative algorithm does not allow a participant of simulation to advance its model time until another participant can send to it a message with a time tag equal to the present instant of the model time. As is shown in [31], such an approach leads to a considerable delay in the operation of the system.

In contrast, optimistic algorithms never restrain the participants. The task of these algorithms is to solve the arising problems of synchronization by rolling back the model time. Optimistic algorithms are harder to implement, require the possibility of preservation and recovery of the states of participants, and require a lot of memory. Moreover, the rollback operation significantly complicates the use of optimistic algorithms for solving simulation problems bound to real time.

In addition to the time advance algorithms considered, researchers sometimes distinguish mixed algorithms, which allow one to unite the participants of a model that work by different schemes [31]. When appropriately applied, mixed algorithms allow one to

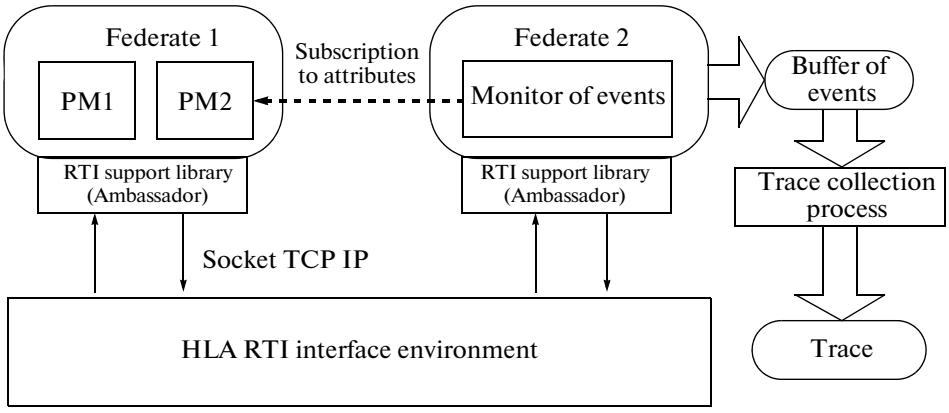

Fig. 3. Scheme of trace collection by a federate collector.

combine the advantages of these schemes; at the same time, wrong matching of different time advance algorithms may lead to a significant decrease in the performance. Therefore, the use of mixed algorithms requires a careful analysis of a simulation model.

5. TOOLS FOR THE COLLECTION AND ANALYSIS OF SIMULATION RESULTS

To collect and analyze the results of simulation, one should solve the following problems:

- Allow an experimenter to specify what events are of interest.
- Organize recording of events that are significant for the experimenter in the DRTES under study.
- Choose a format to record the results of simulation.
- Choose a form to display the results of simulation.
- Choose methods for the analysis of the results of simulation.

5.1. Choice of Events for Recording

The means of recording and tracing the simulation events trace changes in the states of the model components and of the participants of simulation and the exchange of messages between them [9]. Each recorded element is called an event; such an element is surely accompanied by a time tag. The recorded events are displayed by on-line visualization tools, as well as retained in the form of a sequence called a trace. The retained traces are processed by the tools for analysis of the results of simulation of a DRTES.

In the DYANA environment and the HWIL simulation testbed, there is a preset list of events for recording. It contains events related to the transfer of messages between the components of the model and to a change in the parameters of the models themselves. In addition to the preset events, a user may define its own events and model operation modes, which are also recorded in a trace. Note that the possibilities of storing information in such events in both simulation environments are more modest than those in the preset list of events.

5.2. Tools for Recording and Tracing Events

There are known two approaches to the recording of events and collecting traces: centralized and distributed ones.

Centralized data collection is an approach that assumes a uniform collection point for the data of a simulation experiment. This approach requires that the collector of a trace should intercept all the necessary data at one point of the network. The main advantages of the centralized approach are simplicity, lack of the need for time sorting of data, and flexibility and simplicity of management of the tracing process from a single point of data collection. The main shortcoming is the accumulation of a large volume of traffic at a single point of the network, which may lead to overloads and jams.

Distributed data collection suggests the existence of several points of data collection. At each point, only a part of data of a simulation experiment is collected. Thus, a simulation process results in a set of traces collected at different points of the network. The main advantage of the distributed collection is that it prevents an overload of the network at one point. However, this approach requires additional processing of the collected data. The centralized scheme was used in the DYANA project, while the distributed scheme, in the HWIL simulation testbed [9].

The DYANA-2012 environment, which is compatible with the HLA standard, employs the centralized version of the scheme (shown in Fig. 6.1). The idea of this decision consists in the creation of a separate federate (federate 2 in Fig. 3) that is engaged exclusively in the collection of information on the events in the simulation environment and their recording into a trace.

More promising is the distributed collection of information on the basis of a multiagent tracing sys-

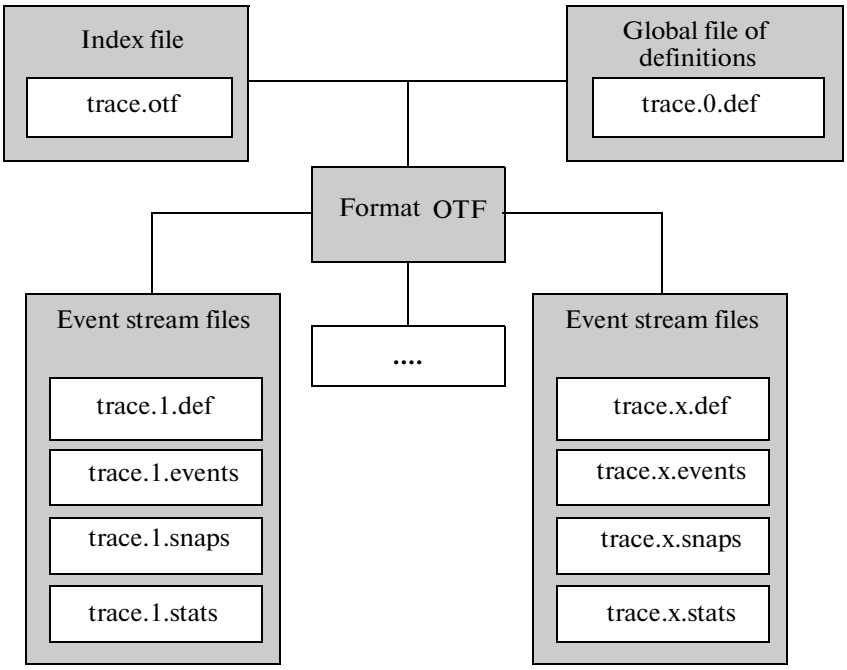

Fig. 4. File structure in the OTF format.

tem [32]; however these schemes are difficult to implement.

5.3. Formats of the Traces of Events

The next question that needs to be solved in a tracer of events is the choice of a format in which a trace is stored. On the one hand, this format should be informative enough to provide storing all the necessary data of a simulation experiment; simple in order that the process of collecting a trace should manage to store a trace with regard to all the incoming events; and compact, since the volume of data accumulated during simulation can be large enough, for example, it may amount to several terabytes.

The DYANA simulation environment employs a specially developed TRC format of a trace. Experience has shown that the format of a trace is too rigid, insufficient, and difficult to expand. These drawbacks were eliminated in the HWIL simulation testbed, in which information on the events is stored in both the main and the external trace. The file of the external trace represents a set of non-fixed-size records. In the external trace, additional information on an event is recorded, while, in the main trace, a reference is given to the record of the corresponding event in the external trace. For example, for a "Parameter updating" type event, the value of the parameter to be changed is recorded in the external trace.

Thus, one can consider the multilevel structure of a trace and the increased time of reading and recording into the trace as the shortcomings of this format. The advantages of the TRC format are the flexibility of the structure of records on the events and the possibility of merging the traces recorded by different components of the simulated system. In the new version of the DYANA-2012 environment, an open trace format (OTF) [33,34] is chosen as the main format for storing a trace. OTF is an open format of traces developed at the Center of High-Performance Computations, University of Dresden, to deal with large-scale parallel platforms. Since the primary objective of the OTF was to store traces for parallel programs, rather than DRTES models, we compared the abstractions of OTF and DRTES. The file structure of the OTF format is illustrated in Fig. 4. One should place special emphasis on the possibility of flexible adjustment of the structure of events due to the presence of the global file of definitions.

5.4. Tools for the Visualization of the Traces of Events

All the simulation environments developed at the LCS since the middle of the 1990s employ a Vis tool for the display and analysis of traces. It provides the following facilities to work with a trace:

- obvious display of the structure of a simulated DRTES;
- visualization of the time line (lifeline) of each component of simulated systems, changes in the states of the components, and the duration of stay of each component in a specific state;
- visualization of the interaction between components and of the attributes of interactions;
- scaling and navigation along a trace;

• search of events and states;

• possibility of application of filters of component-wise, event-wise, and state-wise display.

According to the practice of introduction of a simulation environment into industry, the implemented type of display is not always convenient. The means of visualization should be adjustable to the type of display depending on user's requirements.

5.5. Methods of Quantitative Analysis of Simulation Results

The methods of the quantitative analysis of the results of simulation are reduced to the methods of trace analysis. One can distinguish three types of trace analysis problems: search for an event with preset properties; search for fragments that are "similar" to a given template; and comparison of two traces. All these problems are solved in the Vis environment.

To solve the problems of the two last types, the events of a trace are encoded in some alphabet, and a trace is considered as a row of symbols in this alphabet. Therefore, the problems of trace analysis are reformulated in terms of the search and comparison of sequences of symbols in a row. The following methods are applied: a method of search by means of regular expressions [35], and methods of fuzzy search on the rows [36].

6. TOOLS FOR ESTIMATING THE EXECUTION TIME

As soon as a process is bound to a computer in a DRTES model, the problem arises of estimating the execution time of a program coded block on this computer. This is necessary, for example, to verify the constraints on the execution time of the program as a whole. If a software model is not yet specified up to complete software implementation (in a high-level language or in assembly language) or the architecture of the target computer is not specified, then the time estimate must be explicitly set by the designer of the DRTES model. When the architecture and the instruction set of the computer are known, the time estimation can be automated. The most straightforward method of automation consists in the inclusion of a cycle-accurate instruction set simulator. However, this method significantly slows down the execution of a code block and of the model as a whole.

To speed-up the estimation of the execution time of a code block, a *static–dynamic approach* [37] was developed in the DYANA project. At the translation stage of a model, a static analysis of the text of the model is carried out, and, for every basic block of a code in the program fragment analyzed, an expected execution time is predicted. Then the program block is run on an instrumental computer. During the runtime, the total execution time is determined on the basis of the earlier made static estimates (with certain correction).

An essential role in predicting the execution time is played by the architectural features of a target executor (computer) on which the computations are performed. To map the computations onto the architecture of a specific target computer, one introduces a model of a computer in which a certain set of parameters is established that affect the execution time of the program. For the time estimating subsystem, the following models have been designed within the DYANA project:

• The model of a conventional sequential complex instruction set computer (CISC) [38, 39], which is based on known algorithms of optimal code generation for such computers. This model requires specification of a *generalized instruction set* of the computer along with their execution times. The target computer does not need a cross-compiler of the C language. A description of the instruction set was developed for an Intel 80286 processor.

• The model of a restricted instruction set computer (RISC), which needs a cross-compiler for the target computer and carries out a static analysis on the level of assembly language. This model allows one to statically determine the time delays that arise on pipelined devices due to the dependence of data instructions. The analysis of the behavior of the instructions-cash memory is also carried out in the static mode. Supported processors are Sun MicroSPARC II and Motorola DSP96002.

• The model of an RISC with a vector kernel [41], which combines high (up to a cycle duration) accuracy of estimation with the high execution efficiency of programs (from 4 to 100 times faster than the cycle-accurate instruction set from the chip manufacturer). In contrast to the above-mentioned models, in which the semantics of the estimated block of a code is determined by the semantics of the C language on the instrumental computer, the present model is equipped with a fast instruction set simulator operating according to the rule of translation of the machine codes into C codes for the instrumental computer. A supported processor is NM64403 with NeuroMatrix^{R} architecture.

Along with the problem of estimating the execution time of a program for a specific set of input data, the problem of estimating the worst case execution time (WCET) was considered within the DYANA and DYANA-2012 projects. According to [42], for an arbitrary program, it is impossible to find an estimate for the WCET in the general case. However, under a number of admissible constraints imposed on a DRTES (integer data, explicitly defined bounds for the number of iterations of cycles, and disabled recursion), the problem becomes solvable.

According to the survey in [42], there exist a number of commercial WCET tools, along with a number of promising research prtojects. A WCET method was developed at the LCS for processors with pipelined architecture [43].

To implement an analyzer for estimating the WCET in the DYANA-2012 environment, a static method was chosen with model-checking technology. The advantage of the static method is that it does not require running a program on the target computer and executes a model of the system under study on an instrumental computer almost without any delay. When one applies a verification of programs on models, one can also analyze programs with nondeterministic behavior; at the same time, one does not need to enumerate the entire space of execution paths (as, for example, in the exhaustive method). The implementation is based on a Metamoc tool [44], which involves an UPPAAL verifier for the analysis of a WCET estimate. The analyzer supports programs written in a subset of the C programming language and supports the following structures of the language: integer variables, arrays of integer data, pointers, arithmetic on integer data and pointers, and conditional operators.

7. LOGICAL ANALYSIS

As already mentioned, the main problem of logical (qualitative) analysis is to verify if the behavior of the system complies with the requirements of the specification. This is one of the main problems in the integrated approach to the simulation of DCSs [5]. However, the application of known methods of logical analysis is associated with a fast growth in the number of states of a model as the number of parallel processes increases. Therefore, in the practical implementation and application of verification systems, one has to apply methods of reducing the system of transitions: partial order reduction, the use of different types of symmetries in the model, and abstraction. Another problem is that verification systems require a representation of a model that is different from its representation for quantitative analysis. For practical application of verification systems in simulation systems, one should transform a model from one form of representation to another, which is often a difficult task, especially for the models of complex software systems. In case of error detection, one should transfer the results of verification to a real system and then correctly transfer the corrections made to the verification system. Therefore, a designer of simulation tools faces the following dilemma: either develop his own means of verification, or integrate with an existing verification system.

In the early 1990s, both the native language of specification MM-spec [45] based on temporal logic and the first Russian verification system [46, 47] were developed in the DYANA system. In the latter verification system, a program written in the DYANA simulation language automatically constructs a model that verifies the validity of formulas of temporal logic. These formulas display the requirements (restrictions) imposed on the behavior of the system [48]. In comparison with the semi-automatic and automated methods of verification, the verification system in the DYANA environment does not require that the user should have special knowledge in the field of mathematical logic and methods of programming theory. The system also allows the expert to choose the abstraction level of the initial program and specify the key components of the system.

In the DYANA system, a program in the MM language is translated into the internal representation for verification (the MDL language) [46, 48]. A specification of the behavior of a simulated system is created in the MM-spec language. Then the specification is translated into a set of formulas of branching-time logic CTL [49]. This logic is an extension of propositional logic and allows one to construct specifications in terms of not only states, but also actions. It is important that algorithms for checking the validity of formulas of CTL logic on a distributed program model have linear complexity with respect to the size of a formula and of the model [50].

The model written in the MDL language represents a parallel composition of nondeterministic finite automata extended to admit integer variables. Each automaton is defined by a parameterized description, and the interaction between the automata is performed through common variables. During the implementation of a parallel composition, each automaton is specified by the substitution of the variables through which the interaction is performed for the formal parameters. During the verification, depending on the method chosen (which is specified in the configuration file of the problem), either a graphic or a symbolic representation of the model is constructed according to the MDL-description of the program. Admissible reductions are carried out. The results of verification (a counterexample) are transformed by the interface component into the computation trace of the MM program. This trace is visualized by the trace display means Vis.

Another approach, an UPPAAL open public domain tool [51], was used in the DYANA-2012 system. This approach is described in detail in article [52], which was published in the same issue.

8. INTEGRATION WITH OPTIMIZATION TOOLS

The most topical optimization problems associated with the development of DRTESs can be classified into the following groups:

• choice of the structure of DRTES hardware (set of computers and the structure of exchange channels); a typical problem statement is the minimization of complexity (cost) while preserving the specified schedule times;

• scheduling the execution of tasks on the DRTES computers and exchanges in information exchange channels;

• distribution (connection) of software and hardware components; a possible variant of the problem

statement is as follows: on the basis of the description of the functions of the system, decide which functions are implemented in hardware and which in software, and on what hardware components;

Since 1996, research has been carried out at the Laboratory of Computer Systems on the application of the methods of combinatorial optimization to the above-listed problems [53, 54]. On the basis of the results obtained, a system of automatic planning (Scheduler CAD) of exchange through a channel with centralized control (the MIL-STD-1553B standard or domestic GOST R 52070-2003) [55] has been implemented.

The integration of the HWIL simulation testbed with the Scheduler CAD [56] was performed that enables one

• to use, in simulation, the data on the structure of messages and the structure of the exchange schedule from the Scheduler CAD Database;

• to carry out an automatically created schedule of information exchange on a DRTES model that includes a model of a channel and a model of devices on the channel.

While working on the adaptation of the HWIL simulation testbed to marine navigation systems [57], additional possibilities of defining the exchange schedule through a channel were implemented in a format compatible with the system software (SS) of DRTES devices. This allows one to include the created code, worked out on the models of devices, into the SS of actual DRTES devices.

In the DYANA-2012 environment, the interaction between a DRTES model with schedule optimization tools is organized as follows. The task-execution schedule is specified in the form of an XML document (it is created by the user either manually or automatically by the schedule optimization tool). The object model of this document is constructed on the basis of the formal definition of a schedule from [58] and [59]. The DYANA-2012 environment includes a facility for transforming a schedule into a text of the DRTES model in which the tasks are executed according to this schedule.

There are plans to implement facilities for detailing the model mentioned up to a complete program.

9. CONCLUSIONS

In this article, we have considered a retrospective of several domestic research and development projects in the field of simulation and analysis of the operation of real-time embedded distributed computing systems. We have considered various approaches to the construction of these systems and their merits and demerits and pointed out the main scientific problems on the way to the development of such systems and ways of their solution. For each project, we have shown examples of its application in industry.

Comparing the results obtained at the LCS during the last 30 years with the results of other researchers and software producers, we can note the following.

We are not familiar with any earlier works on an integrated approach to the study of quantitative and logical properties of DRTESs.

We have implemented the approach involving the joint design and simulation (codesign) of software and hardware tools of DRTESs independently and almost simultaneously with foreign research groups. For instance, we designed the prototypes of the main subsystems of the DYANA environment in 1994–1996 and issued the final version of the environment of the second generation in 1999. Foreign publications on the codesign of environments have appeared since 1994 (see the survey [60]).

We can point out the following directions as prospective fields of research:

1. Combination of conservative and optimistic schemes of synchronization of the model time in the execution environment of models.

2. Development of a modular execution environment for discrete-event simulation models adjusted to the type of simulated objects.

3. Automation of the processing of simulation results.

4. Development of new methods of visualization of the results of simulation of DRTESs, including online visualization.

5. Development of methods and tools for planning of simulation experiments.

6. Development of a distributed runtime environment with real-time control with the use of software-defined networks [61].

ACKNOWLEDGMENTS

This work is supported by the Ministry of Education and Science of Russian Federation (state contract no. 14.740.11.0399) within the Federal Target Program "Scientific and Scientific and Pedagogical Personnel of Innovative Russia." The work was partly supported by the Russian Foundation for Basic Research (project nos. 95-01-01590-a, 98-01-00151-a, 01-01-00263-a).

REFERENCES

1. Basiladze, S.G., Smelyansky, R.L., Karavaev, A.I., Emel'yanov, M.V., Kosov, V.F., and Eloev, O.Z., Experimental multiprocessor system STEND, in *Moduli ø programmnoe obespechenie system avtomatizatsii eksperimental'nykh issledovanii: Uchebno-metodicheskoe posobie* (Modules and Software for Systems of Automation of Experimental Investigations: Tutorial and Methodical Book), Basiladze, S.G., Ed., Moscow: Mosk. Gos. Univ., 1990, pp. 120–129.
2. Bakhmurov, A., Kapitonova, A., and Smeliansky, R., DYANA: An environment for embedded system design and analysis, *Proc. 32nd Annual Simulation Symposium*, San-Diego, California, 1999, pp. 50–57.
3. Smelyansky, R.L., Operation model of distributed computing systems, *Vestn. Mosk. Univ., Ser. 15: Vychisl. Mat. Kibern.*, 1990, no. 3, pp. 3–21.

4. Smelyansky, R.L., Operation theory of distributed computating systems, *Trudy mezhdunarodnoi konferentsii "Parallel'nye vychisleniya i zadachi upravleniya" (PACO'2001)* (Proc. Int. Conf. "Parallel Computation and Control Problems" (*RASO*'2001), (Moscow, 2001), Moscow: Inst. Probl. Upr. Trapeznikova RAN, 2001, pp. 161–182.
5. Molonov, V.G., and Smelyansky, R.L., A complex approach to the simulation of distributed computing systems, *Programmirovanie*, 1988, no. 1, pp. 57–65.
6. Broenink, J.F., Groothuis, M.A., Visser, P.M., and Orlic, B., A model-driven approach to embedded control system implementation, *Western Multiconference on Computer Simulation, WMC 2007*, San Diego, 2007, pp. 137–144.
7. Smelyansky, R.L., Chistolinov, M.V., Bakhmurov, A.G., and Zakharov, V.A., On international project in the field of verification of software for embedded systems, in *Programmnye sistemy i instrumenty: Tematicheskii sbornik fakul'teta VMiK MGU im. Lomonosova* (Software and Hardware: Thematic Collection of the Faculty of Computational Mathematics and Cybernetics, Moscow State University), Korolev L.N., Ed., Moscow: MAKS Press, 2000, pp. 24–30.
8. Gribov, D.I. and Smelyansky, R.L., Complex simulation of the on-board equipment of a flying vehicle, in *Methods and Means of Information Processing: Proc. 2nd All-Russian Scientific Conf.* (Moscow, 2005), Moscow: Mosk. Gos. Univ., 2005, pp. 59–74.
9. Balashov, V.V., Bakhmurov, A.G., Volkanov, D.Yu., Smelyansky, R.L., Chistolinov, M.V., and Yushchenko, N.V., Hardware-in-the-loop simulation testbed for the design of embedded computing systems, in *Methods and Means of Information Processing: Proc. 3rd All-Russian Scientific Conf.* (Moscow, 2009), Moscow: Mosk. Gos. Univ.; MAKS Press, 2009, pp. 16–25.
10. Smelyansky, R.L. and Bakhmurov, A.G., Application of the HWIL simulation method to the design of on-board equipment of a flying vehicle, in *Proc. 3rd All-Russian Scientific and Engineering Conf. "Supercomputer and Multiprocessor Computing Systems (MVS'2001)* (Taganrog, 2002), Taganrog: TRTU, 2002, pp. 135–140.
11. Volkanov, D.Yu., *Programmirovanie* (in press).
12. Nance, R.E., *A History of Discrete Event Simulation Programming Languages*, Blacksburg: Virginia Polytech. Inst. State Univ., 1993.
13. Dijkstra, E.W., *A Discipline of Programming*, Englewood Cliffs: Prentice-Hall, 1976.
14. Bakhmurov, A., Kapitonova, A., and Smeliansky, R., DYANA: An Environment for Embedded System Design and Analysis, in *Proc. 5th International Conference TACAS'99*, (*Amsterdam, 1999*), Amsterdam: Springer, 1999 (LNCS vol. 1579, pp. 390–404).
15. Bakhmurov, A.G. and Smeliansky, R.L., DYANA—the pilot project of investigation of distributed programs and computer systems, *Proc. 2nd Russian–Turkish Seminar on New High Information Technologies*, Gebre, 1994.
16. Riddle, W.E., An approach to software system behavior description, *Comput. Languages*, 1979, vol. 4, pp. 29–47.
17. Riddle, W.E., An approach to software system modelling and analysis, *Comput. Languages*, 1979, vol. 4, pp. 49–66.
18. Peterson, J., *Petri Net Theory and Modeling of Systems*, Englewood Cliffs: Prentice-Hall, 1981.
19. Gomaa, H., *Designing Concurrent, Distributed, and Real-Time Applications with Uml*, Boston: Addison-Wesley, 2000.
20. Sanchez, P.J., Fundamentals of simulation modeling, in *Proc. 39th Winter Simulation Conference (WSC'07)* (Washington, 2007), pp. 54–62.
21. *IEEE Std. 1278.1a-1998*: Simulation Interoperability Standards Committee of the IEEE Computer Society IEEE Standard for Distributed Interactive Simulation: Application Protocols, 1998.
22. *Simulation Interoperability Standards Committee of the IEEE Computer Society*: IEEE Standard for Modeling and Simulation (MS) High Level Architecture (HLA) Federate Interface Specification, 2000.
23. Chaudron, J. B., Saussié, D., Siron, P., and Adelantado, M., Real-time aircraft simulation using HLA standard, *IEEE AESS Simulation in Aerospace*, Toulose, 2011.
24. Adelantado, M., Siron, P., and Chaudron, J.B., Towards an HLA Run-time infrastructure with hard real-time capabilities, *International Simulation Multi-Conference (ISMC'10)*, Ottawa, 2010.
25. Fujimoto, R. M., Perumalla, K., Park, A., Wu, H., Ammar, M. H., and Riley, G.F., Large-scale network simulation: How big? How fast?, *Proc. 11th IEEE/ACM Symposium on Modeling, Analysis and Simulation of Computer Telecommunication Systems (MASCOTS'03)*, Orlando, 2003.
26. Kazakov, Yu.P. and Smelyansky, R.L., Organization of distributed simulation, *Programmirovanie*, 1994, no. 2, pp. 45–64.
27. d'Ausbourg, B., Siron, P., and Noulard, E., Running real time distributed simulations under Linux and CERTI, *European Simulation Interoperability Workshop*, Edinbourgh, 2008.
28. Noulard, E., Rousselot, J.Y., and Siron, P., CERTI, an open source RTI, why and how, *Joint 2009 Spring Simulation Interoperability Workshop (SIW)*, 2009.
29. Malinga, L. and Le Roux W.H., HLA RTI performance evaluation, *European Simulation Interoperability Workshop*, Istanbul, 2009, pp. 1–6.
30. Chemeritskiy, E.V., Towards a HLA-based hardware-in-the-loop simulation runtime, *Proc. 6th Spring/Summer Young Researchers' Colloquium on Software Engeneering, SYRCoSE-2012*, Perm, 2012.
31. Fujimoto, R.D., *Parallel and Distributed Simulation Systems*, Wiley Interscience, 2000.
32. Song, H.J., Shen, Zh.Q., Miao, Ch.Y., Tan, A.H., and Zhao, G.P., The multi-agent data collection in HLA-based simulation system, *21st International Workshop on Principles of Advanced and Distributed Simulation (PADS'07)*, 2007.
33. Knupfer, A., Brunst, H., Malony, A.D., and Shende, S.S., Open Trace Format API Specification Version 1.1, Dresden: Dresden Univ. of Technology, 2006.
34. Knupfer, A., Brunst, H., Malony, A.D., and Shende, S.S., Open Trace Format (OTF) Tutorial: Presentation, *Dresden: Univ. of Dresden*, 2006.
35. Bochkov, S.O. and Smelyansky, R.L., Program debugging in distributed computing systems, *Programmirovanie*, 1988, no. 4.
36. Volkanov, D.Yu. and Cherei, M.V., A study of the applicability of fuzzy search algorithms to the analysis of the results of simulation of real-time computer systems, in *Programmnye sistemy i instrumenty: Tematicheskii sbornik*,

8 (Software and Hardware: Thematic Collection no. 8), Moscow: Mosk. Gos. Univ., 2007, pp. 137–147.
37. Yushchenko, M.V., Estimating the execution time of programs by a static–dynamic method, in *Programmnye sistemy i instrumenty: Tematicheskii sbornik fakul'teta VMiK MGU im. Lomonosova* (Software and Hardware: Thematic Collection No. 2 of the Faculty of Computational Mathematics and Cybernetics, Moscow State University), Korolev L.N., Ed., Moscow: Mosk. Gos. Univ., 2001, pp. 157–167.
38. Kapitonova, A.P., Smelyansky, R.L., and Terekhov, I.V., Hardware system for evaluating the laboriousness of computations in programs, in *Sistemnoe programmirovanie i modeli issledovaniya operatsii* (System Programming and Models for Operations Research), Moscow: Mosk. Gos. Univ., 1993, pp. 57–72.
39. Kapitonova, A.P., Smelyansky, R.L., and Terekhov, I.V., System for evaluating the time characteristics of programs: Architecture and Implementation, in *Programmno-apparatnye sredstva i matematicheskoe obespechenie vychislitel'nykh system* (Software–Hardware and Mathematical Tools for Computer Systems), Moscow: Mosk. Gos. Univ., 1994, pp. 92–103.
40. Balashov, V.V., Kapitonova, A.P., Kostenko, V.A., Smelyansky, R.L., and Yushchenko, N.V., Method and means for estimating the execution time of optimized programs, *Programmirovanie*, 1999, no. 5, pp. 52–61.
41. Savenkov, K.O. and Yushchenko, M.V., Method for describing the behavior of a processor to estimate the execution time of a program, in *Methods and Means of Information Processing: Proc. 2nd All-Russian Scientific Conf.* (Moscow, 2003), Moscow: Mosk. Gos. Univ., 2003, pp. 486–491.
42. Wilhelm, R., and Engblom, J., The worst-case execution time problem—overview of methods and survey of tools, *ACM Transactions on Embedded Computing Systems*, 2008, vol. 7, no. 3, article 36.
43. Prus, V.V., Method for estimating the worst-case execution time for a processor with pipelined architecture, in *Methods and Means of Information Processing: Proc. 2nd All-Russian Scientific Conf.* (Moscow, 2005), Moscow: Mosk. Gos. Univ., 2005, pp. 167–174.
44. Dalsgaard, A.E., Olesen, M.Ch., Toft, M., Hansen, R.R., and Larsen, K.G., *METAMOC: Modular Execution Time Analysis Using Model Checking*, 2010.
45. Bakalov, Yu.V., and Smelyansky, R.L., Behavior specification language for distributed programs, *Programmirovanie*, 1996, no. 5, pp. 41–51.
46. Tsar'kov, D.V., Application of modularity to the verification of distributed programs, in *Programmnye sistemy i instrumenty: Tematicheskii sbornik, 1* (Software and Hardware: Thematic Collection No. 1), Moscow: Mosk. Gos. Univ., 2000, pp. 128–136.
47. Tsar'kov, D.V., System of formal verification of distributed programs in the DYANA simulation environment, in *Proc. Int. Conf. "Parallel Computation and Control Problems" (RASO'2001)* (Moscow, 2001), Moscow: Inst. Probl. Upr. Trapeznikova RAN, 2001, pp. 161–182.
48. Zakharov, V.A., and Tsar'kov, D.V., Efficient algorithms for the verification of the executability of CTL temporal logic formulas on the model of their application for the verification of parallel programs, *Programmirovanie*, 1998, no. 4, pp. 3–18.
49. Clarke, E.M. and Emerson, E.A., "Design and synthesis of synchronization skeletons for branching time temporal logic, in *Proc. Logic of Programs: Workshop*, (Yorktown Heights), New York: Springer, 1981. (Lecture Notes in Computer Science 131.)
50. Clarke, E.M., Jr., Orna, O., and Peled, D., *Verification of Program Models: Model Checking*, Moscow: MTsNMO, 2002.
51. Bengtsson, J., Larsen, K.G., Larsson, F., Pettersson, P., and Yi, W., UPPAAL—a tool suite for automatic verification of real-time systems, *Lecture Notes in Computer Science*, 1996, vol. 1066, pp. 232–243.
52. Volkanov, D.Yu., Zakharov, V.A., Zorin, D.A., Konnov, I.V., and Podymov, V.V., *Programmirovanie* (in press).
53. Kostenko, V.A., The problem of schedule construction in the joint design of hardware and software, *Programming Comput. Software*, 2002, vol. 28, no. 3, pp. 162–173.
54. Kostenko, V.A. and Smelyansky, R.L., Method and algorithms for the design of the structures of computing systems by information on the behavior of programs, in *Proc. 2nd All-Russian Scientific Conf.* (Moscow, 2005), Moscow: Mosk. Gos. Univ., 2005, pp. 564–571.
55. Balashov, V.V., Kostenko, V.A., and Smeliansky, R.L., A tool system for automatic scheduling of data exchange in real-time distributed avionics systems, in *Proc. 2nd EUCASS European Conference for. Aerospace Sciences*, Brussels, 2007.
56. Balashov, V.V., Bakhmurov, A.G., Chistolinov, M.V., Smeliansky, R.L., Volkanov, D.Yu., and Youshchenko, N.V., A Hardware-in-the-loop simulation environment for real-time systems development and architecture evaluation, in *Proc. Third International Conference on Dependability of Computer Systems DepCoS-RELCOMEX 2008*, Szklarska Poreba, 2008.
57. Balashov, V.V., Bakhmurov, A.G., Volkanov, D.Yu., Smelyansky, R.L., Chistolinov, M.V., and Yushchenko, N.V., Application of a HWIL testbed to the design of computing systems for a marine navigation complex, in *Programmnye sistemy i instrumenty: Tematicheskii sbornik, 9* (Software and Hardware: Thematic Collection No. 9), Moscow: Mosk. Gos. Univ., 2008, pp. 153–165.
58. Kalashnikov, A.V. and Kostenko, V.A., Parallel algorithm for the simulation of annealing for constructing multiprocessor schedules, *Izv. Ross. Akad. Nauk: Teor. Sist. Upr.*, 2008, no. 3, pp. 101–110.
59. Zorin, D.A., A method for representing and transforming schedules in iterative algorithms for the structural synthesis of real-time computing systems, *Programmnye sistemy i instrumenty: Tematicheskii sbornik, 12* (Software and Hardware: Thematic Collection No. 12), Moscow: Mosk. Gos. Univ., 2011, pp. 163–171.
60. de Micheli, G., and Gupta, R.K., Hardware/software co-design, *Proc. IEEE*, vol. 85, no. 3, pp. 349–365.
61. Smelyansky, R.L., Software configurable networks, in *Open Systems*, 2012, no. 09 (http://www.osp.ru/os/2012/09/13032491/).

Translated by I. Nikitin