*Joint NCC & IIS Bull.*, Comp. Science, 12 (1999), 6–9


# Towards animation of executional specifications of the REAL language*

E.V. Bodin

The work is devoted to a tool for simulation of executional specifications of distributed systems, simulation and testing of the behaviour of these systems. The systems are described in terms of the REAL language developed in A.P. Ershov Institute of Informatics Systems, SD RAS [1, 2, 3, 4].

## 1. Introduction

The development of the animation system is motivated by the necessity to distinguish the simulation of a distributed system from the animation of a specification, i.e., its execution according to the formal semantics.

The simulation is now a well-developed approach [6], while the animation is a new research domain due to the complexity of formal semantics of specification languages. A specific advantage of the formal semantics of the REAL language is its conciseness, so that it is suitable for a journal publication and has been published in [3, 4]. Many existing tools are known only by references, and a few of them have available demo versions, but there is no universally recognized modeling tool. Most of them (especially those modeling executional specifications) require considerable amount of hardware resources and do not run on PC. Thus, a modeling tool suitable for PC was needed.

Some of the available modeling tools (such as COVERS) have no formal input language (non-interactive facilities for description of the system being simulated).

The combined real-time specification language REAL for distributed systems and their properties has been under development since 1992 under scientific supervision of Dr. V.A. Nepomniaschy. The starting point of the research was the initial version of the specification language REAL [2] consisting of an SDL-like executional specification language and a logical specification language based on a combination of a dynamic logic and a branching time temporal logic extended by real time. The REAL language [3, 4] has several levels. Now they are Elementary-REAL, Basic-REAL, and REAL itself. Each level consists of executional and logical specifications.

## 2. Distributed system representation: a survey of executional specifications

Executional specifications are used for distributed system representation and they have a hierarchical structure based on processes. The processes are grouped into (sub)blocks which, in turn, make up higher level blocks. Channels are used to specify communication between such entities as processes, blocks and the external environment of the system being specified. Channels are intended for signals with possible parameters. Channels can have different inner structures: queues, stacks, bags. Several entities can also communicate by means of a bus which is a generalization of the channel concept.

A process specification in the REAL language, as well as an SDL process, describes a sequence of such actions as changing variable values, reading a signal from a bus or a channel, writing a signal into a bus or a channel, cleaning a bus or a channel. In contrast to SDL, each action in REAL is associated with a time interval which specifies duration of the action. Non-deterministic transitions are acceptable.

*Partially supported by INTAS–RFBR under Grant 95–0378.

The REAL language employs a new time concept, namely, the multiple clock concept synchronized by means of linear inequalities on their speed. For example, the time scale

$1ing \leq 1min \leq 20ing, 30opr \leq 1sec \leq 100opr, 1min = 60sec$

has several solutions, each of which can be considered as a special relative speed of four separate clocks with time units *ing*, *opr*, *sec*, and *min*. The scale imposes some synchronization for the special clock speeds in the multiple clock in order to ensure that all the inequalities hold.

The set of behaviours of an executional specification is restricted by fairness conditions. The REAL language considers only the "fair behaviours", i.e., the behaviours in which each fairness condition holds infinitely often.

A block diagram consists of routes which connect a subblock to another subblock (a channel route) or to a bus (a bus route). Channel and bus routes can also connect subblocks and buses to the external environment. In this case the channel or the bus is the *input* or *output* one. Otherwise, the channel or the bus is *inner*. A block diagram can have both graphical and linear (textual) form. The graphical form of the block diagram is a marked graph. Its vertices are marked by the names of subblocks, names of buses, or by the environment symbol (the vertices are represented by rectangular boxes with names within), and the edges correspond to the channel or bus access.

Informally speaking, all subblocks of a block work in parallel, and they interact with each other and with the external environment by sending signals with parameters into channels or buses. No concurrent access to channels/buses is allowed, i.e., only one subblock or environment can change the contents of a channel.

A process diagram is a generalization of the program schemata concept. It consists of transitions. Each transition consists of a control *state*, a transition *body*, a time *interval*, and a non-deterministic *jump* to the next control states. The body of a transition determines the action to perform, such as

- to read a signal from an input channel (and assign the values of its parameters to program variables);
- to write a signal into an output channel (with expressions of the corresponding type as the parameter values);
- to read a signal from an input bus (and assign the values of its parameters to program variables);
- to write a signal into an output bus (with expressions of the corresponding type as the parameter values);
- to clean a bus or a channel;
- to change the values of the process's variables according to the program.

Each of the actions is performed instantly, but it can happen only in the time interval specified for the transition, i.e., if the time passed from the moment when the state of the transition became active is in the time interval of the transition.

A process diagram can have both graphical and linear forms. The graphical form of a process diagram is auxiliary (and it may be not presented in the specification). It is a marked oriented graph whose vertices are marked by the control states (and the shape of the vertex depends on the kind of the transition) and the edges correspond to the control flow. The linear form of a process diagram is mandatory. It is a complete description of all transitions of the process.

Each process is non-deterministic but sequential, i.e., no concurrency is allowed inside a process.

## 3. Animation problem

Let us state once more that the animation of formal specifications is a new research domain in contrast to the simulation. When simulating, non-determinism may be randomized and time can be chosen according to the trustworthy expert knowledge. When animating, all possible choices of possible behaviours and time current must be considered. Moreover, due to the local choice of the step rules,

some sequences of steps may cause an abnormal termination. Therefore, the backtracking problem arises. This problem is more similar to the same problem of the distributed simulation than to the backtracking problem of logical programming. As concerns the time problem, it seems to be even more complicated than the choice of the step rules and the corresponding backtracking, since, in general, a scale has an infinite set of solutions (i.e., variants of time current), while different time currents may lead to different behaviours. To realize other possible problems (different from backtracking and the choice of time currents), it seems reasonable to develop and implement a simulation system for a time-free specification.

Presently, the simulation system consists of two parts: the Analyzer and the Executor.

• The Analyzer is written in Turbo Pascal 6.0 (and, therefore, it can be executed under DOS or under DOS-emulator under Linux). Its purpose is to translate the REAL specifications into the internal representation. The internal representation is a plain text file which has the Windows INI-file format, it consists of 'sections' containing information in the form '*variable* = *value*', where *value* is a string or a number. This internal representation makes it easy for another person to convert the output of the Analyzer into the input data for other applications (in particular, for a model-checker). Since REAL is a language under development, the Analyzer is open for modifications of its syntax. It is achieved by using the TPLex and TPYacc tools.

• The Executor is written in Borland Delphi (without using any Windows 95 specific code), so it can run under practically any Microsoft Windows (and possibly under 'wine', the Windows emulator for Linux, though this was not tested). The Executor is written using the Delphi Visual Component Library, which makes it easier to modify the appearance of the system being simulated. The entities of the REAL language are represented in the Executor by means of 'classes' (the object-oriented concept) exploiting the inheritance and the polymorphism. The Executor is an implementation of the step rules of the REAL semantics. Since we implement a simulation system, the non-deterministic choice is randomized or user-driven (during the execution, the user may affect the simulation by choosing an action to be executed from several possible actions available).

## 4. An experiment of simulation

The experiments are discussed using the example "Good passenger and slot-machine" described in detail in [4]. It is a protocol of servicing a good passenger by a slot-machine. The slot-machine keeps the money received from the passenger and has a keyboard with the station, return, and request buttons, a slot for coins, an indicator for showing a sum, a tray for change, and a booking window. A good passenger knows a station that he/she needs, has enough money and can perform the following actions: press buttons on the keyboard; drop coins into the slot; see the reading of the indicator; get coins from the change tray; get a ticket from the booking window. Informally, a protocol of servicing a good passenger is as follows. The seance begins when the passenger presses a button corresponding to the desired station on the keyboard. The slot-machine, after having received the station name, shows the price on the indicator. Then the following loop begins: the passenger looks at the indicator and, if he/she sees a non-zero sum, chooses a coin and drops it into the slot, then the slot-machine subtracts the nominal of the coin from the sum to be received from the passenger and shows a new value on the indicator. The passenger can quit this loop at any instant by pressing the cancel button, so that the slot-machine must return through the change tray all the coins received so far. The good passenger, however, does not use this option and, when he/she reads zero from the indicator, he/she presses the ticket request button. The slot-machine receives the request and prints the ticket with the station name, and the passenger takes the ticket. The seance is over for the slot-machine when it returned all sum to the passenger or printed out a ticket. The seance is over for the passenger when he/she took the money from the change tray (what the good passenger never does) or took the ticket that he/she needed.

This protocol can be specified as a block consisting of two processes. The diagram of the block `passenger_and_machine` is presented on Fig. 1. We would like to avoid some syntactical details, so we

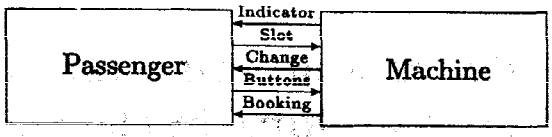


**Figure 1.** Block "passenger_and_machine"

present here only the specification of the block; complete specifications of the processes are given in [4]. The context of the block consists of the inner channel declarations and coincides with the channel declarations in the context of the logical specification. The specification of the process `machine` consists of the diagram and the fairness conditions. We would like to remark that three transitions correspond to the state `getcoin` in the linear form of the diagram: the first one corresponds to receiving a coin, the second and the last correspond to receiving the request ticket command and the cancel command, respectively. The fairness conditions for `machine` are:

`¬AT start ; ¬AT defcount ; ¬AT showcount ; ¬AT add ;`
`¬AT retcoin ; ¬AT check ; ¬AT give ;`

i.e., the behaviour of the process `machine` is fair if the process cannot stay forever in any state other than waiting for input signals. The specification of the process `passenger` is presented in a graphical form on the diagram (see Fig. 3 in [4]). In the state `continue` the passenger "decides" what he/she must do: to choose a coin (`chcoin`) or to request a ticket (`request`). Thus, two transitions correspond to this state in the linear form of the diagram. The fairness conditions of `passenger` are similar to the fairness conditions of `machine`: the behaviour of the process `passenger` is fair if the process cannot stay forever in any state other than waiting for input signals.

If in the specification of the process `machine` in the transition `add` (see Fig. 2 in [4]) where the slot-machine determines the sum that the passenger still have to pay, according to the nominal of the coin that he/she has dropped, the statement `left=left-val` is replaced by `left=left-50`, then correctness of functioning of the slot-machine will depend on the coins that the passenger drops. If the nominals of the coins (with a possible exception for the last one) are 50, then the passenger will receive the ticket to the desired station after having paid the right amount of money. If the nominals of the coins are less than 50, then he/she will pay less than the price of the ticket, but the ticket will be received. This error is discovered during the simulation of the specification: when the passenger receives the ticket, the value of the variable `paid` representing the sum of the coins dropped so far is less than the price of the ticket to the desired station.

# References

[1] E.V. Bodin, *Approaches to the verification of specifications on language REAL*, Specification and verification problems for concurrent systems, 1995 (in Russian).

[2] V.A Nepomniaschy, N.V. Shilov, *REAL92: A combined specification language for real-time concurrent systems and properties*, Lect. Notes Comput. Sci., **735**, 1993, 377–389.

[3] V.A. Nepomniaschy, N.V. Shilov, E.V. Bodin, *A concurrent systems specification language based on SDL & CTL*, Proc. of Workshop on Concurrency, Specifications & Programming, Berlin, Humboldt University, Informatik-Bericht, No 36, 1994, 15–26.

[4] V.A. Nepomniaschy, N.V. Shilov, E.V. Bodin, *Formal semantics and verification of distributed systems presented by Basic-REAL specifications*, Joint Bulletin of NCC& IIS, Ser.: Comput. Sci., 7, Novosibirsk, 1997, 35–56.

[5] *Specification and Description Language*, CCITT, Recommendation Z.100, 1988.

[6] A.A.B. Pritsker, *Introduction to simulation and SLAM II*, Moscow, "Mir", 1987.