The Process Algebra Usage for Simulation Purposes

Olishchuk S., Volk M.

Abstract — **The Formal description of existing simulation methods with the help of process algebra is considered in the report. Mathematical representation of simulation process by means of operations group performed onto it is given.**

***Index Terms*—Simulation, process, alphabet, class, event**

I. Introduction

TODAY several methods of the simulation have gotten a wide distribution. They allow to display most naturally the essence of the concrete simulated system for solving problems of certain scientific or technical area. However, there is a problem of absence of formal apparatus describing construction of such systems. This work is directed on the solution of this problem owing to the process algebra application, which describes the essence of the simulation process in the form of regular set of algebraic laws.

The ultimate goal is to form the transparency of systems construction, which will make it possible to see both current and future problems in a new light. Owing to this, it will become possible to solve these problems more economically, reliable and which is still more important to minimize their number.

II. The Basic Concepts of Process Algebra

For the beginning we'll describe the basic concepts which are necessary as mathematical basis.

We'll consider event as base concept - any action done in a context of process. The term process is introduced to denote behavior of an object which can be described within a limited set of events chosen as its alphabet. The alphabet is understood as a set of names of the events chosen for the concrete description of the object. The alphabet is considered to be constant, the predetermined property of the object. Participating in the event beyond its alphabet is logically impossible for the object.

Manuscript received March 14, 2009. This work was supported in part by the Ukrainian Department of GRID systems research.

S. O. Olishchuk is with the Kharkiv National University of Radio Electronics, Electronic Computers Department. Address: Ukraine, 61166, Kharkov, tel. (057)-702-13-54, e-mail: serg_olish@mail.ru.

M. O. Volk is with the Kharkov National University of Radio Electronics, Electronic Computers Department. Address: Ukraine, 61166, Kharkov, tel. (057)-702-13-54, e-mail: volk@kture.kharkov.ua.

It's considered that concrete event in object's lifetime occurs instantly i.e. a basic action, not having extents in time. Continuous, i.e., demanding time, action should be considered as a pair of events, the first of which marks the beginning of action, and the second - its end. Duration of the action is defined by an interval between its onset-event and end-event; during all this time there can be other events. Two continuous events can overlap in time, if the beginning of each of them precedes the end of another.

Let's introduce following agreements:

1. We'll denote the names of the events by the words composed of lowercase letters, for example: *event1*, *event2*, as well as the letters a, b, c, d, e.
2. We'll denote the names of the concrete processes by the words made of capital letters, for example: SIMULATION, SERVING.
3. Letters A, B, C are used to denote of set of events.
4. Letters X, Y are used for the variables, denoting processes.
5. The alphabet of process P is denoted αP, for example: αSIMULATION = {*start*, *stop*}

Process with the alphabet A is such that it doesn't happen any event from A in it, we'll name it STOP A. This process describes the behavior of a broken object: having physical ability to participate in events from A, it never uses it. We distinguish objects with different alphabets even if these objects do nothing.

In summary, we define simple system of designations which will help in describing the objects, which already have ability to some actions.

Let think *x* - event, P - process. Then (x → P) (is read as "P for x") describes the object, which participates in event x in the beginning, and then behaves in accuracy as P. Process (x → P) has by definition the same alphabet, as P; therefore this designation can be used only under the condition that *x* belongs to the same alphabet. More formally: $\alpha(x \rightarrow P) = \alpha P$, if$\rightarrow \alpha P$.[1]

III. The Process Algebra Usage

Depending on the concept that is at the basis of simulation model, the methods are divided into event-oriented, transact-oriented and agent-oriented [2]. The latter method is used less frequently, so we will focus our attention on the first two.

In the event-oriented simulation system there are following agreements:

- model moves through time from event to event, which alter the state of the model;
- the logic of events determines the sequence of changing states of the model that are associated with the occurrence of these events;
- the time is moving from event to event;

The algorithm description is given in [2].

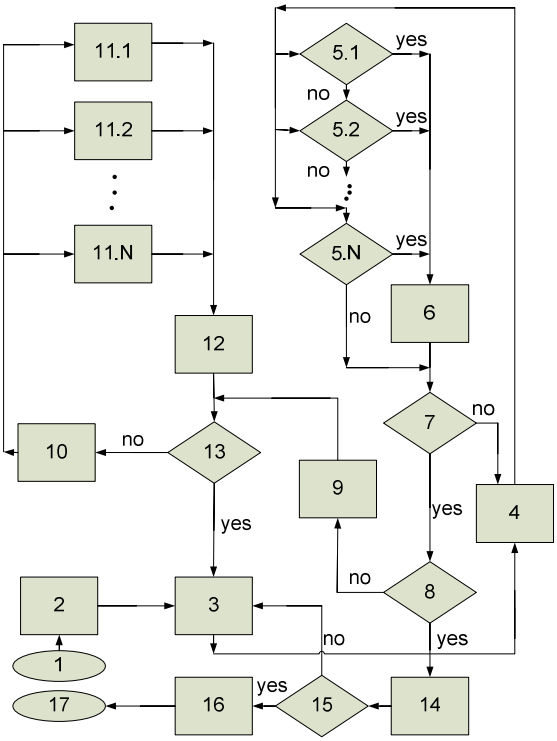

Fig. 1. Block diagram of the event-oriented simulation system

Translation of the algorithm into the language of the process algebra is defined like following. First we should define alphabet of simulation process:

αSIMULATION = {*start*, *init, set_cycle1_parameter, inc_cycle1_iter_cnt, check_event, insert_event_to_list, is_end_of_cycle1, is_list_empty, set_cycle2_parameter, get_event_from_list, serve_event, modify_next_event_time, is_end_of_cycle2, correct_time, is_imitation_finished, deinit, finish*}, where

start – start of the simulating process (1)
init – initialization of the simulating process (2)
set_cycle1_parameter - set parameters of the cycle (3)
inc_cycle1_iter_cnt - increase counter of the cycle (4)
check_event - checking the conditions of implementation of the event (5)
insert_event_to_list - adding the event to the list for processing (6)
is_end_of_cycle1 - check the end conditions of the cycle1(7)
is_list_empty - check if the list of events is empty for processing (8)
set_cycle2_parameter - set parameters of the cycle2 (9)
get_event_from_list - receive events from the list (10)
serve_event - servicing of the event (11)
modify_next_event_time - modification of the moment of the next event (12)
is_end_of_cycle2 - check the end conditions of the cycle2 (13)
correct_time-adjustment of the model time (14)
is_imitation_finished - check the end conditions of simulation(15)
deinit - the simulating process deinitialization (16)
finish – the end of the simulating process (17)

The model's behavior is defined as consequent events changing like following:

SIMULATION = *start* → *init* → CYCLE1_EXEC → if not *is_list_empty* then CYCLE2_EXEC else *correct_time* → if not *is_imitation_finished* then CYCLE1_EXEC else *deinit* → *finish*

CYCLE1_EXEC_AUX = *inc_cycle1_iter_cnt* → if *check_event* then *insert_event_to_list* → if not *is_end_of_cycle1* then CYCLE1_EXEC_AUX

CYCLE1_EXEC = *set_cycle1_parameter* → CYCLE1_EXEC_AUX

CYCLE2_EXEC_AUX = *set_cycle2_parameter* → *get_event_from_list* → *serve_event* → *correct_time* → if *is_end_of_cycle2* then CYCLE1_EXEC else CYCLE2_EXEC_AUX

CYCLE2_EXEC = *set_cycle2_parameter* → CYCLE2_EXEC_AUX

A SIMULATION name refers to the process (in fact, process modeling).

The transact-oriented simulating systems are characterized by:

- the model is a stream of transacts to promote them from one process step to another.
- the state of the model is changed in discrete moments of time, each step of the process is associated with the start of sequence of events.
- the steps are repeated during the whole simulation time.
- the model is as flexible and effective, as event-oriented, but less abstract.

Today transact-oriented simulating systems are widely adopted.

The algorithm description is given in [2].

Translation of the algorithm into the language of the process algebra:

αSIMULATION = {*start, initialization, scan_sources, create, is_all_sources_considered, scan_absorbers, destroy, is_all_absorbers_considered, create_list, is_list_empty, stop_serving, start_serving, shift_time, is_simulation_finished, deinit, finish*}, where

start – start of the simulating process (1)
initialization – initialization of the simulating process(2)
scan_sources – scanning the source of transacts (3)
create – creating transacts with given density (4)

is_all_sources_considered – checking if all the sources of transacts are scanned (5)
scan_absorbers – scanning the absorber of transacts(6)
destroy – destruction of transacts (7)
is_all_absorbers_considered – checking if all the absorbers are considered (8)
create_list – creation of the transaction list (9)
is_list_empty – checking if the list of transacts is empty(10)
stop_serving – stop serving the transacts (11)
start_serving – start serving the transacts (12)
shift_time – shifting of the simulating time (13)
is_simulation_finished – checking the end conditions of modeling (14)
deinit – deinitialization of the simulating process (15)
finish – the end of the simulating process (16)

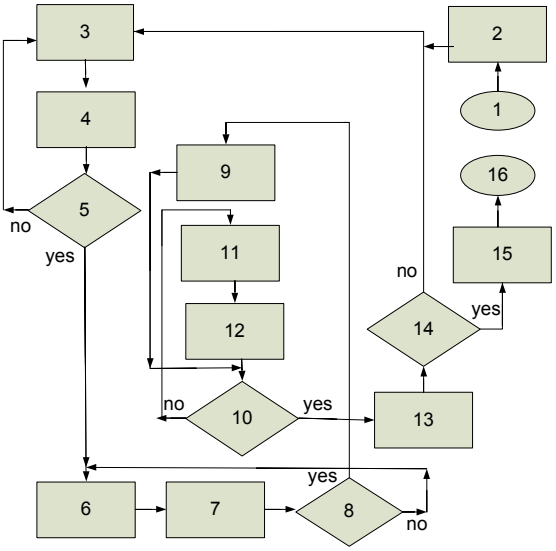

Fig. 2. Block diagram of the transact-oriented simulation system

SIMULATION = *start* → *initialization* → TRANZAKTS_CREATION → TRANZAKTS_DESTROYING → *create_list* → TRANZAKTS_SERVING → *shift_time* → if not *is_simulation_finished* then TRANZAKTS_CREATION → *deinit* → *finish*

TRANZAKTS_CREATION = *scan_sources* → *create* → if not *is_all_sources_considered* then TRANZAKTS_CREATION

TRANZAKTS_DESTROYING = *scan_absorbers* → *destroy* → if not *is_all_absorbers_considered* then TRANZAKTS_DESTROYING

TRANZAKTS_SERVING = if not *is_list_empty* then *stop_serving* → *start_serving*

Comparing these two simulation methods we can see that they have similar alphabets and behavior description which can give us opportunity to describe them as a common formal model. Thus way we can approach by the HLA general purpose architecture [6], which is just parameterized with some specific parameters. This helps us to make such systems decomposition and distributed development easier.

IV. Conclusion

The most obvious area of the approach outlined in the article application is specification, development and computer systems implementation which continuously act and interact with their surroundings.

The basic idea is that using process algebra a system can be decomposed easily into parallel subsystems interacting with each other and with their general surroundings. Parallel decomposition of subsystems gets not more complicated than sequential combination of lines or operators in the conventional programming languages. Such approach includes many of actual ideas of structuring used in the contemporary research on language and programming methodology, such as monitors, classes, modules, packages, critical sections, envelopes, forms, and even ordinary sub-programs. And finally, this approach is a sound basis for avoiding such errors as the divergence, deadlocks, loops, and to prove the correctness of the design and development of computing systems.

References

[1] *Ch.Hoar*. Interacting sequential processes. M.: Mir, 1989 . 264 pp.
[2] Maksimey IV Simulation on the EC. M.: Radio & Communications, 1988. 232 pp.
[3] H. Miller, J. Sanders. Scoping the Global Market: Size Is Just Part of the Story. IT Professional, 1(2):49-54, 1999.
[4] http://en.wikipedia.org/wiki/IEEE_1516

Sergey O. Olishchuk – PhD Student of Kharkov National University of Radio Electronics.

Maxim O. Volk – PhD, Prof. Assistant of Kharkov National University of Radio Electronics.