THE USE OF FORMAL METHODS FOR MODELLING AND VERIFICATION OF DISTRIBUTED SYSTEMS

Henrikas Pranevicius

Kaunas University of Technology
Studentu 50, Kaunas LT-3028, Lithuania

Tel./Fax: (370 7) 767333
E-mail: Henrikas.Pranevicius@if.ktu.lt
Homepage: http://kat.vik.ktu.lt/hepran/hp.htm

Abstract

The main topic of this paper is the use of piece-linear aggregate mathematical method for formal specification, simulation and validation of distributed systems. The main advantage of this approach is that it integrates two tasks of different nature, i.e. simulation and correctness analysis of specification, on the base of a single specification. Method of verification of aggregate specifications using forward-backward simulation approaches is delivered also. The presented methods are explained using illustrative examples.

1. Introduction

Distributed systems arise in many applications, including telecommunications, distributed information processing, scientific computation and real-time process control.

Two properties are essential for distributed systems:

- computation activity is represented as the concurrent execution of sequential processes,
- processes communicate by passing messages.

The models of computation generally considered to be distributed are process models, in which computational activity is represented as the concurrent execution of sequential processes [LL90]. The process models that are most obviously distributed are ones in which process communicate by message passing: one process sends a message by adding it to a message queue, and another process receives the message, by removing it from queue. These models vary in such details as the length of the message queues and how long a delay may occur between when as message is sent and which it can be received.

A wide variety of message-passing models can be used to represent distributed systems. They can be classified by the assumptions made about four separate concerns: network topology, synchrony, failure and message buffering.

Two kinds of analysis are used for analysis of distributed systems: behaviour and performance analysis. All possible trajectories are analysed during behaviour analysis and it permits to check a correctness of specification. Various validation and verification methods are used for correctness analysis. During performance analysis computer executes developed specification of distributed system. Performance analysis is carried out by simulation means. The main characteristics which are analysed during behaviour and performance analyses are named in Table 1.

Table 1. Analysis methods and analysed characteristics

Kind of analysis	
Behaviour	Performance
Used methods	
Validation and verification	Simulation
Analysed characteristics	
Static and dynamic deadlocks	Lengths of queues
Termination	Transmission time of messages
Invariant properties	Waiting time
Safety	Units utilisation coefficients
Liveness	...
Equivalence	
...	

Various formal methods are used for specification and analysis of distributed systems. This publication is about aggregate approach [Pran91], which permits integrate behaviour and performance analysis on the base of single specification. Such possibility permits to prove that developed specification is correct and to evaluate performance characteristics of an analysed system. For example, analysing computer network protocol it is not enough to prove correctness of its specification but also it is needed correctly to choose parameters of protocols such as timer values, buffer sizes, channel capacities, etc.

The most general definition of a distributed system is a triple $\{S, A, \Sigma\}$, where S – the set of states, A – the set of actions and Σ – the set of behaviours.

Each behaviour is a finite or infinite sequence of the form $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_2 \xrightarrow{a_3} \ldots$, where each s_i is a state and each a_i is an action. A state describes the complete instantaneous state of the system, an action is a system operation that is taken to be indivisible, and a behaviour represents an execution of the system whose a_i action takes the system from state s_{i-1} to state s_i. The set Σ represents the set of all possible system executions.

There are two general approaches describing the set Σ: *constructive* and *axiomatic*. In the constructive approach Σ describes by program which one may be written in a conventional programming language or in terms of a formal model such as I/O automata [Lynch], various modification of Petry nets, calculus of communicating systems [Miln80], aggregate approach [Pran91].

Axiomatic description leads directly to a method of reasoning. If Φ is the set of axioms that describe Σ and C is property expressed in the same form system as Φ, then the system satisfies C if and only if the formula $\Phi + C$ is valid.

2. Aggregate Approach

2.1. The Use of Controlling Sequences for Formal Description of Piece-Linear Aggregates

In the application of the aggregate approach for system specification, the system is represented as a set of interacting piece – linear aggregates (PLA) [Pran82]. The PLA is taken as an object defined by a set of states Z, input signals X, and output signals Y. The aggregate functioning is considered in a set of time moments $t \in T$. The state $z \in Z$, the input signals $x \in X$, and the output signals $y \in Y$ are considered to be time functions. Apart from these sets, transition H and output G operators must be known as well.

The state $z \in Z$ of the piece-linear aggregate is the same as the state of a piece-linear Markov process, i.e.:

$$z(t) = (\upsilon(t),\ z_\upsilon(t)),$$

where $\upsilon(t)$ is a discrete state component taking values on a countable set of values; and $z_\upsilon(t)$ is a continuos component comprising of $z_{\upsilon 1}(t), z_{\upsilon 2}(t), \ldots, z_{\upsilon k}(t)$ co-ordinates.

When there are no inputs, the state of the aggregate changes in the following manner:

$$\upsilon(t) = \text{const},\ \frac{dz_\upsilon(t)}{dt} = -\alpha_\upsilon,$$

where $\alpha_\upsilon = (\alpha_{\upsilon 1}, \alpha_{\upsilon 2}, \ldots, \alpha_{\upsilon k})$ is a constant vector.

The state of the aggregate can change in two cases only: when an input signal arrives at the aggregate or when a continuous component acquires a definite value. The theoretical basis of piece-linear aggregates is their representation as piece-linear Markoff processes. The exact definition of piece-linear processes is given in [NB73].

Aggregate functioning is examined on a set of time moments $T = \{t_0, t_1, \ldots, t_m, \ldots\}$ at which one or several events take place, resulting in the aggregate state alternation. The set of events E which may take place in the aggregate is divided into two non-intersecting subsets $E' = E' \cup E''$. The subset $E' = \{e'_1, e'_2, \ldots, e'_N\}$ comprises classes of events (or simply events) e'_i, $i = \overline{1, N}$ resulting from the arrival of input signals from the set $X = \{x_1, x_2, \ldots, x_N\}$. The class of events $e''_i = \{e''_{ij}, j = 1, 2, 3, \ldots\}$, where e''_{ij} is an event from the class of events e''_i taking place the j-th time since the moment t_0. The events from the subset E' are called external events. A set of aggregate input signals is unambiguously

reflected in the subset E' i.e., $X \to E'$. The events from the subset $E'' = \{e_1'', e_2'', ..., e_f''\}$ are called internal events where $e_i'' = \{e_{ij}'', j = 1, 2, 3, ...\}$, $i = \overline{1, f}$ are the classes of the aggregate internal events. Here, f determines the number of operations taking place in the aggregate. The events in the set E'' indicate the end of the operations taking place in the aggregate.

For every class of events e_i'' from the subset E'', control sequences are specified $\{\xi_j^{(i)}\}$, where $\xi_j^{(i)}$ – the duration of the operation, which is, followed by the event e_{ij}'' as well as event counters $\{r(e_i'', t_m)\}$, where $r(e_i'', t_m)$, $i = \overline{1, f}$ is the number of events from the class e_i'' taken place in the time interval $[t_0, t_m]$.

In order to determine start and end moments of operation, taking place in the aggregate the so-called control sums $\{s(e_i'', t_m)\}$, $\{w(e_i'', t_m)\}$, $i = \overline{1, f}$ are introduced, where $s(e_i'', t_m)$ – the time moment of the start of operation followed by an event from the class e_i''. This time moment is indeterminate if the operation was not started; $w(e_i'', t_m)$ is the time moment of the end of the operation followed by the event from the class e_i''. In case of no priority operations, the control sum $w(e_i'', t_m)$ is determined in the following way:

$$w(e_i'', t_m) = \begin{cases} s'(e_i'', t_m) + \xi_{r(e_i'', t_m)+1}, & \text{if at the moment } t_m \text{ an operation is taking place,} \\ & \text{which is followed by the event } e_i; \\ \infty, & \text{in the opposite case.} \end{cases}$$

The infinity symbol (∞) is used to denote the undefined values of the variables.

The control sum definition presented above is used in simulation. When aggregate models are used for system formalisation and correctness analysis, the control sum may be determined in a simplified way:

$$w(e_i'', t_m) = \begin{cases} < \infty, & \text{if at the moment } t_m \text{ an operation is taking place, followed by the event } e''; \\ \infty, & \text{in the opposite case.} \end{cases}$$

Control sums determine only the possibility conditions for the events after the moment t_m, while the event occurrence moments are not determined.

2.2. The Special Cases of Aggregate Model

In this section it will be shown that it is possible to get various cases of other well-known models from the aggregate model.

Automata without memory:

$$v(t_m) = \varnothing,\ z_v(t_m) = \varnothing,\ E' \neq \varnothing,\ E'' = \varnothing,\ z(t_{m+1}) = H(e_i'),\ y = G(e_i'),\ e_i' \in E'.$$

Automata with memory:

$$v(t_m) \neq \varnothing,\ z_v(t_m) = \varnothing,\ E' \neq \varnothing,\ E'' = \varnothing,\ z(t_{m+1}) = H(z_v(t_m), e_i'),$$
$$y = G(z_v(t_m), e_i'),\ e_i' \in E'.$$

System of differential equations:

$$v(t_m) = \varnothing,\ z_v(t_m) \neq \varnothing,\ E' = \varnothing,\ E'' = \varnothing,\ \frac{dz_{v_i}(t)}{dt} = f[t, z_{v_i}(t_m), x(t)],\ i = \overline{1, k},$$
$$t \in (0, \infty).$$

The general model of aggregate:

$$v(t_m) \neq \varnothing,\ z_v(t_m) \neq \varnothing,\ E' \neq \varnothing,\ E'' \neq \varnothing,\ \frac{dz_v(t)}{dt} = f[t, z_v(t_m), x(t)],\ t \in [t_m, t_{m+1}),$$
$$v(t) = \text{const}, \text{ when } t \in [t_m, t_{m+1}),\ z(t_{m+1}) = H(z_v(t_m), e_i),\ y = G(z_v(t_m), e_i),$$
$$e_i \in E' \cup E''.$$

The piece-linear aggregate:

$$v(t_m) \neq \varnothing,\ z_v(t_m) \neq \varnothing,\ E' \neq \varnothing,\ E'' \neq \varnothing,\ \frac{dz_v(t)}{dt} = -1,\ t \in [t_m, t_{m+1}),\ v(t) = \text{const},$$
$$\text{when } t \in [t_m, t_{m+1}),\ z(t_{m+1}) = H(z_v(t_m), e_i),\ y = G(z_v(t_m), e_i),\ e_i \in E' \cup E''.$$

3. The Use of Simulation Technique for Correctness Analysis of Aggregate Specifications

The most widespread method used for correctness analysis of automata models is simulation technique [NL93, NL95]. In section 2.1 it was showed that aggregate specifications belong to the class of I/O automata. It enables us to use a simulation technique for correctness analysis of aggregate specifications. In this section simulation technique will be used for correctness analysis of alternating-bit protocol. We will use proof strategy based on a hierarchy of automata. This hierarchy represented by description series of a system at different abstraction levels. The process of moving through the abstractions, from the highest level to the lowest, is known as successive refinement. The top level may be a problem specification written in the form of an automaton. Lower levels in the hierarchy look more and more like an actual system that will be used in practice.

n formal specifications of the same system with different abstraction levels are presented in Figure 1, where f_{ij}, $i = \overline{1, n-1}$, $j = \overline{2, n}$ are relations between i-th and j-th specifications.

Figure 1. The sequence of specifications with different abstraction levels

Let A and B be two I/O automata with the same external interfaces. Also A is an automaton with the higher abstraction level and B is an automaton with the lower abstraction level. Suppose f is a binary relation over $states(A)$ and $states(B)$, that is, $f \subset states(A) \times states(B)$. The f is a simulation relation from A to B, provided that both of the following are true:

1. If $s \in start(A)$, then $f(s) \cap start(B) \neq 0$.
2. If s is reachable state of A, $u \in f(s)$ is a reachable state of B and $(s, \pi, s') \in trans(A)$, then there is an execution fragment α of B starting with u and ending with some $u \in f(s')$, such that $trace(\alpha) = trace(\pi)$.

The first condition asserts that any start state of A has some corresponding start state in B. The second condition asserts, that any step of A, and any state of the step, have a corresponding sequence of steps in B. This corresponding sequence can consist of one step, many steps, or even no steps, as long as the correspondence between the states is preserved and the external behaves is the same.

Next, a use of simulation technique for verification on ABP is discussed. We will consider two specifications: the first describes protocol service – higher abstraction level and the second – an algorithm of ABP (see section 3.4) – lower abstraction level.

An aggregate system describing the service of the protocol is depicted in Figure 2, where INFS – the packet, which has to be transmitted, INFR – the packet, which has been received by receiver and ACK – an acknowledgement. State graph of service of ABP is presented in Figure 3.

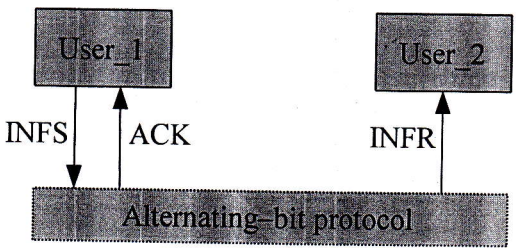

Figure 2. Aggregate system describing the service of ABP

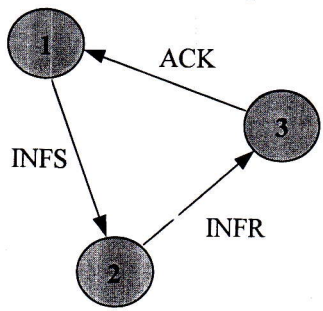

Figure 3. The state graph of service ABP

Figure 4 depicts aggregate system of ABP with lower abstraction level.

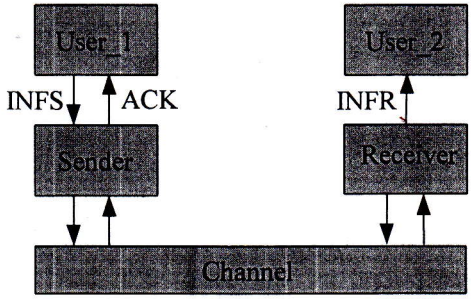

Figure 4. Aggregate system of ABP (lower abstraction level)

Carried out verification of ABP showed that correctness of protocol depends on a timer value. If the timer value is less then a sum of transmission times of packet and acknowledgement, then the protocol does not fulfil a required service. Such situation is illustrated in Figure 5.

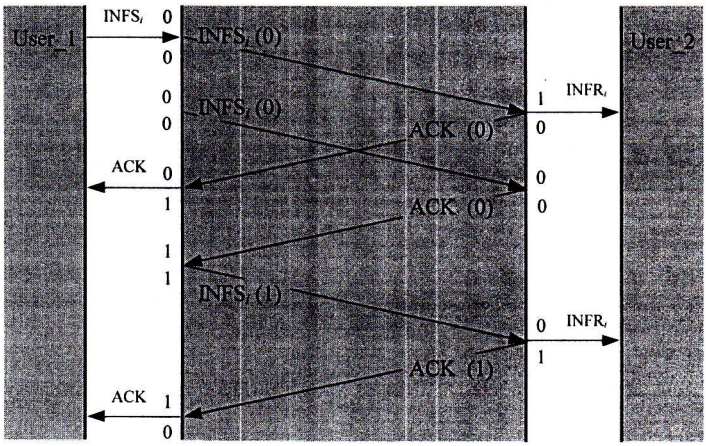

Figure 5. Timing diagram of ABP when timer value is less than a sum of transmission times of packet and acknowledgement

It is possible to formulate two problems, which arise implementing simulation technique for analysis correctness of protocols:

- definition of relation f between states in low and high abstraction levels of specification;
- searching for trajectory(ies) in specification with lower abstraction level for each step in specification with higher abstraction level.

4. Specification and Validation of Timed Protocol with Slot Reuse

4.1. Informal Description of the Protocol

A system comprises of a set of stations and the server, which communicates using slots sent via a unidirectional bus [HP 97]. Stations can receive messages from an environment and forward them to the server, which remove this message from the system. The server can receive messages from the environment and send them to a named destination station; we assume that the environment provides this destination name. When a station receives a message from the server and the message is destined for that station it removes the message from the system, otherwise it return the message to the bus.

Each station and the server identified by a unique Station Number, and comprise of a buffer (B) to hold slots from the bus and a queue (Q) to hold what it receives from the environment.

The bus comprises of the slots, which hold messages or can be empty; each slot is a tuple (Slot Number, OriginatingStation, DestinationStation, Message). Each slot is identified by a unique identifier SlotNumber and Slots circulate in the bus by visiting each station in turn.

4.2. Aggregate Specification of the Protocol

Aggregate system, which describes the protocol is presented in Figure 6. Channels, which are used in aggregate system, are duplex, it means that signals may be transmitted from both ends of the channel. Aggregate system consists of the following aggregates: *Bus*, *Slot_*1, … ,*Slot_M*, *Server*, *Workstation_*1, … , *Workstation_*(*N*–1). Formal descriptions of these aggregates are presented below.

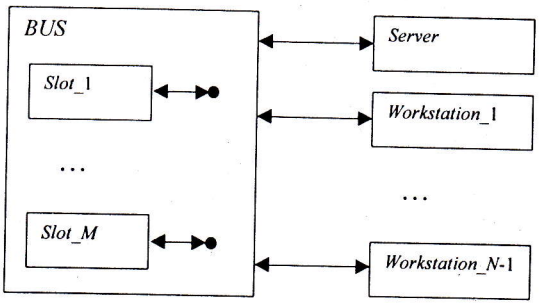

Figure 6. Aggregate scheme of protocol

Formal description of aggregate *Server*

1. *The set of input signals.* $X = \{x_1\}$, $x_1 = (sl, s, r, st)$, where sl – number of the slot which is transmitted, s – number of sending station, r – number of receiving station,

$$st = \begin{cases} 0, \text{ if slot is empty,} \\ 1, \text{if slot transmits packet to server,} \\ 2, \text{if slot transmits packet from server.} \end{cases}$$

2. *The set of output signals.* $Y = \{y_1\}$, $y_1 = (sl, s, r, st)$.
3. *The set of external events.* $E' = \{e_1'\}$, where e_1' denotes an arrival of the signal x_1.
4. *The set of internal events.* $E'' = \{e_1'', e_2''\}$, where e_1'' means that service of slot is ended; e_2'' - message in workstation is formed.
5. *Controlling sequences.* $e_i'' \to \{\mu_i\}$, $i = 1, 2$, where μ_1 –duration of slot service in server; μ_2 – duration of message formation.
6. *Discrete component of state.* $v(t_m) = \{v_1(t_m), v_2(t_m)\}$, where $v_1(t_m)$ – number of the slot which is served in server; $v_1(t_m) = 0$, when there are no slots in server; $v_2(t_m)$ – the number of messages in queue.
7. *Continuous component of state.* $z_v(t_m) = \{w(e_1'', t_m), w(e_2'', t_m)\}$, where $w(e_i'', t_m)$, $i = 1, 2$ – the time instant in which event e_i'' has to occur.
8. *Initial state.* $w(e_1'', t_0) = \infty$, $w(e_2'', t_0) = t_0 + \mu_2$, $v_1(t_0) = 0$, $v_2(t_0) = 0$.
9. *Transition and output operators*:

$H(e_1')$: $v_1(t_{m+1}) = sl$, $w(e_1, t_{m+1}) = t_m + \mu_1$.

$G(e_1')$: Y = ∅.

$H(e_1'')$: $v_2(t_{m+1}) := v_2(t_m) - 1$ if $v_2(t_m) > 0$;

$s(t_{m+1}) := 1$, $r(t_{m+1}) := 1 + \varphi$, $st(t_m) := 2$ if $v_2(t_m) > 0$,

$s(t_{m+1}) := 0$, $r(t_{m+1}) := 0$, $st(t_{m+1}) = 0$ if $v_2(t_m) = 0$,

$v_1(t_{m+1}) = 0$, $w(e_1'', t_{m+1}) := \infty$,

where φ – random value, which define the number of receiver and $\varphi \in \{1, \ldots, N-1\}$.

$G(e_1'')$: $y_1 := (v_1(t_m), s(t_m), r(t_m), st(t_m))$.

$H(e_2'')$: $v_2(t_{m+1}) := v_2(t_m) + 1$.

$G(e_2'')$: Y := ∅.

Formal description of aggregate *Workstation_j, j* = 1, 2, ..., *N*–1.

1 The set of input signals. $X = \{x_1\}$ $x_1 = (sl, s, r, st)$,

2 The set of output signals. $Y = \{y_1\}, y_1 = (sl, s, r, st)$.

3 The set of external events. $E' = \{e_1'\}$, where e_1' means an arrival of the signal x1.

4 The set of internal events. $E'' = \{e_1'', e_2''\}$, where e_1'' means that service of slot is ended in workstation; e_2'' means that new message has arrived to workstation.

5 Controlling sequences. $e_i'' \to \{\mu_i\}$, $i = 1, 2$, where μ1 – duration of slot service; μ2 – duration of message formation.

6 Discrete component of state. $v(t_m) = \{v_1(t_m), v_2(t_m)\}$, where v1(tm) – number of the slot which is served in s-th workstation; v1(tm) = 0, when there are no slots in workstation; v2(tm) – the number of messages in queue.

7 Continuous component of state. $z_v(t_m) = \{w(e_1'', t_m), w(e_2'', t_m)\}$.

8 Initial state. $w(e_1'', t_0) = \infty$, $w(e_2'', t_0) = t_0 + \mu_2$, $v_1(t_0) = 0$, $v_2(t_0) = 0$.

9 Transition and output operators:

$H(e_1')$: $v_1(t_{m+1}) = sl$, $w(e_1'', t_{m+1}) = t_m + \mu_1$.

$G(e_1')$: $Y = \varnothing$.

$H(e_1'')$: $v_2(t_{m+1}) := v_2(t_m) - 1$, $s(t_{m+1}) := j$, $r(t_{m+1}) := 1$,

$s(t_{m+1}) := 1$ if $st(t_m) = 0 \wedge v_2(t_m) > 0$,

$s(t_{m+1}) := 0$, $r(t_{m+1}) := 0$, $st(t_{m+1}) := 0$ if $st(t_m) = 1 \wedge r(t_m) = j$,

$v_1(t_{m+1}) = 0$, $w(e_1'', t_{m+1}) = \infty$.

$G(e_1'')$: $y_1 := (v_1(t_m), s(t_m), r(t_m), st(t_m))$.

$H(e_2'')$: $v_2(t_{m+1}) :=$, $v_2(t_m) + 1$.

$G(e_2'')$: $Y := \varnothing$.

4.3. Results of Validation of Protocol

Specification presented in the previous section was validated using validation subsystem of protocol analysis system PRANAS-2 [HP94]. Some results of validation are presented in Table 2, when number of workstation is 2 and 1 slot is in the bus. Operation "Slot transmission" describes a process of slot transmission through a bus. Operation "Slot in station" describes process of slot processing in either server or workstation. Beside the arrows names of operations are written, which initialise transition to other state. State 383 describes situation, when each station has message for transmission and the slot is transmitted to server. When operation "Slot transmission" ends the state is changed.

Table 2 illustrates validation experiment that after some transitions all messages, which are in stations (state 383), are transmitted to other stations.

Table 2. Results of validation

State Number	Aggregate	Discrete state component				Active operations
		v_1	v_2	v_3	v_4	
383	*Server*	0	1			Message
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	1	0	0	0	Slot transmission
↓	Slot transmission (aggregate *Slot_1*)					
44	*Server*	1	1			Message Slot in station
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	1	0	0	0	
↓	Slot in station (aggregate *Server*)					
77	*Server*	0	0			Message Slot in station
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	2	1	3	2	Slot transmission
↓	Slot transmission (aggregate *Slot_1*)					
127	*Server*	0	0			Message
	Workstation_1	1	1			Message Slot in station
	Workstation_2	0	1			Message
	Slot_1	2	1	3	2	Slot transmission
↓	Slot in station (aggregate *Workstation_1*)					
191	*Server*	0	0			Message
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	3	1	3	2	Slot transmission
↓	Slot transmission (aggregate *Slot_1*)					
260	*Server*	0	0			Message
	Workstation_1	0	1			Message
	Workstation_2	1	1			Message Slot in station
	Slot_1	3	1	3	2	
↓	Slot in station (aggregate *Workstation_2*)					
314	*Server*	0	0			Message
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	1	0	0	0	Slot transmission
↓	Slot transmission (aggregate *Slot_1*)					
29	*Server*	1	0			Message Slot in station
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	1	0	0	0	
↓	Slot in station (aggregate *Server*)					
54	*Server*	1	0			Message
	Workstation_1	0	1			Message
	Workstation_2	0	1			Message
	Slot_1	2	0	0	0	
↓	Slot transmission (aggregate *Slot_1*)					
92	*Server*	1	0			Message
	Workstation_1	0	1			Message Slot in station
	Workstation_2	0	1			Message
	Slot_1	2	0	0	0	

5. References

[HP94] H. Pranevicius, V. Pilkauskas, A. Chmieliauskas. *Aggregate approach for specification and analysis of computer network protocols*. Kaunas, Technologija, 1994, 228p.

[LL90] L. Lamport, N. Lynch. *Distributed Computing: Modes and Methods*. Handbook of Theoretical Computer Science, edited by J. van Leeuwen, Elsevier Science Publishers, 1990.

[Lynch] N.A. Lynch. *Distributed algorithms*. Morgan Kaufmann Publishers, San Francisco, 1996

[Miln80] R. Milner. A *Calculus of Communicating Systems*, Springer–Verlag, 1980.

[NB73] N. Buslenko, V. Kalashnikov, I. Kovalenko. *Lectures on Complex Systems*. Moscow, Soviet. Radio, 1973, (In Russian).

[NL93] N.A. Lynch, F.W. Vaandrager. *Forward and backward simulations – part II: Timed-based systems*. Report CS-R9314, CWI, Amsterdam.

[NL95] N.A. Lynch, F.W. Vaandrager. *Forward and backward simulations – part I: Untimed systems. Information and Computation*, Vol. 121, No.2, 1995, pp.214–233.

[Pran82] H. Pranevicius. *Models and methods for computer system investigation*. Mokslas, Vilnius, 1982, 228p. (In Russian).

[Pran91] H. Pranevicius. *Aggregate approach for specification, validation, simulation and implementation of computer network protocols*. Lectures notes in Computer Science, 502, Springer–Verlag, 1991, pp.433–477.

[HP97] H. Pranevicius. L.Sintonen. *Simulation of high-speed ring access with slot reuse*. In: Information thechnologies, Kaunas University of Technology, 1997, pp.237–244.