EXTENDED ALGORITHM FOR TRANSLATION OF MSC-DIAGRAMS INTO PETRI NETS

Sergii Kryvyy, Oleksiy Chugayenko

Abstract: The article presents an algorithm for translation the system, described by MSC document into Petri Net modulo strong bisimulation. Obtained net can be later used for determining various systems' properties. Example of correction error in original system with using if described algorithm presented.

Keywords: MSC, Petri Net, model checking, verification, RAD.

ACM Classification Keywords: D.2.4 Software/Program Verification - Formal methods, Model checking

Conference: The paper is selected from XIVth International Conference "Knowledge-Dialogue-Solution" KDS 2008, Varna, Bulgaria, June-July 2008

Introduction

The growing topicality of modern software systems and the rapidity imposed by pressing time-to-market demands for new approaches to the development process of high-performance and low-cost software. Namely, design productivity should be improved by means of new methodologies implementation due to the increase of the complexity of the systems. Formal methods begin to play a crucial role during design process. The use of formal methods during the initial stages of the development process can help to improve the quality of the later software, even if formal methods are not used in subsequent phases of development.

Protocol design is one of the most critical problems in distributed communication systems. Effective design methodology for protocol design requires formal models which are able to capture the inherent aspects of a system specification and verification tools that allow the designer to verify that a system satisfies its specification, to check the correctness of the system specification, and quickly explore alternative solutions.

Backgrounds

MSC is a modeling technique that uses a graphical interface, which was standardized by ITU (International Telecommunication Union) [1], [2]. It is usually applied to applications of the telecommunication domain, since they have properties of distributed reactive real-time systems. MSC diagrams are widely used in the early design stages of the systems development to capture system requirements. So, MSC is extremely suitable to capture the scenarios that a designer might want the system to exhibit (or avoid). MSC describes message flow between the instances, which present asynchronously communicating objects of the system or system entities like blocks, services or processes of the system. One MSC diagram describes a certain portion of system behavior or a scenario of communication between the instances. The set of the scenarios (MSC–diagrams) are captured as requirements which constitute a complete behavioral description of the system. Let us describe briefly the most basic MSC-constructs.

Instances and Messages. Instance is a basic primitive of MSC, which in graphics is presented as vertical line with its name. Message transmissions, which are acts of communication between instances, are presented by horizontal arrows with possible curve or tilt under angle for reflecting “overtaking” or “intersection” of messages. The beginning of the vector marks a sending of the message and its ending marks receiving of the message. Evens of sending and receiving of the messages are ordered along the instances so that sending of the message always happens earlier than its receiving. There is one more rule in standard MSC’2000 [1] for ordering events along the instances: everything located above happens earlier than that located below (except coregion part, where events are not ordered).

Creation and termination of instances may be specified within MSCs. An instance may be created by another instance. No message events before the creation can be attached to the created instance. The instance stop is the counterpart to the instance creation, except that an instance can only stop itself whereas an instance is created by another instance.

Conditions. Condition is used as for restricting or defining a set of MSC traces through indicating states of the system so for defining the composition of one MSC diagram from the several MSCs. Namely, the standard MSC'2000 [1] defines conditions of two types: setting condition and guarding condition.

Conditions of the first type are those which describe the current global system state (global condition), or some non-global state (nonglobal condition). In the latter case the condition may be local, i.e. attached to just one instance. Instances presenting dynamic objects can be began and finished, so far globality of a state considers dynamically changing set of instances.

Conditions of the second type restrict behavior of the MSC to execution of events in a certain part of MSC depending on the value of the given guarding condition.

Besides of composition role, conditions according to the standard [1] are the means of events synchronization. For example, if two instances share one and the same condition, then for each message between these instance its sending and receiving events shall happen both before or both after setting of the condition. If two conditions are ordered directly sharing the common instance, or indirectly through conditions on other instances, then this order must be respected on all instances that share these two conditions.

General Ordering. General ordering is used to impose additional orderings upon events that are not defined by the normal ordering given by the MSC semantics. For example, it may be used to specify that an event on one instance must happen before an otherwise unrelated event on another instance.

There cannot be both upwards and downwards steps on the same ordering. This means that it may consist of consecutive vertical and horizontal segments. The textual grammar defines the partial order relations by the keywords before and after. They indicate directly in what order the involved events must come in the legal traces.

Inlines. Inline expressions used to create various composition of events inside MSC diagram. They allow creating of alternative, parallel, loop compositions and exceptional and optional regions. Last two are special cases of alternative composition.

Environment and Gates. The gates represent the interface between the MSC and its environment. Any message or order relation attached to the MSC frame constitutes a gate. Due to possibility of MSC nesting, gates can be defined for MSC diagrams, MSC diagram references (reference expressions) and for inlines. For gate connections the associated gate definition must correspond with the actual gate. Message gates are used for message events and order gates are used for causal ordering.

Order gates represent uncompleted order relations where an event inside the MSC will be ordered relative to an event in the environment. Order gates are always explicitly named. Order gates are considered to have direction - from the event ordered first to the event coming after it. Also order gates are used on references to MSC diagrams (MSCs) in other MSCs. The actual order gates of the MSC reference are connected to other order gates or to events.

Message gates define the connection points of messages with the environment. The message gates are used when references to the MSC are put in a wider context in another MSC. A message gate always has a name. The name can be defined explicitly by a name associated with the gate on the frame. The actual message gates on the MSC reference are then connected to other message gates or instances. Similar to gate definitions, actual gates may have explicit or implicit names.

MSC Semantics. MSC is a language with formally defined semantics, which is based on the process algebra. Applying this formal semantics to MSC a process term can be derived for each MSC and each MSC specification. MSC language semantics based on process algebra was defined first for textual representation of MSC diagrams in the form of expressions of process algebra that was called denotation semantics. Operational semantics is defined via transitional rules added to algebraic expressions.

Petri Net. Ordinary Petri net is used as a formal model to define the semantics of MSC system and support analysis [3], [4].

Definition 1. A net is a triple $N=(P,T,F)$, such that P and T are disjoint sets of places and transitions respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is binary incidence relation between places and transitions (flow relation).

On the basis of incidence relation F (flow relation) characteristic function $\overline{F}:(P \times T) \cup (T \times P) \to N$ is introduced, where N is the set of natural numbers.

The sets ${}^\bullet x = \{y \mid yFx\}$ and $x^\bullet = \{y \mid xFy\}$ denote the pre- and post set of arbitrary element $x \in P \cup T$ of net.

The following three conditions are required for the net $N=(P,T,F)$:

C1) $P \cap T = 0$,

C2) $(F \neq 0) \wedge [(\forall x \in P \cup T)(\exists y \in P \cup T): xFy \vee yFx]$,

C3) $(\forall p_1, p_2 \in P)\left(p_1 = p_2 \wedge p^1 = p^2 \Rightarrow p_1 = p_2 \right)$.

Definition 2. A marking of the net $N=(P,T,F)$ is the function $\mu : P \to N$. The equation $\mu(p) = k \in N$ means that place $p \in P$ has k tokens.

Definition 3. Petri net (PN) is a triple (N,μ_0,W), comprising a certain net N, certain initial marking μ_0, and W is weight function (or multiplicity of an arc).

A transition $t \in T$ is enabled at a marking μ of PN (N,μ_0) iff $\forall p \in {}^\bullet t : \mu(p) \geq \overline{F}(p,t)$. Such a transition can be fired, leading to the marking μ' according the following rule: $\forall p \in P : \mu'(p) = \mu(p) - \overline{F}(p,t) + \overline{F}(t,p)$. A sequence of transitions $\sigma = t_1, t_2, \ldots, t_n$ is an occurrence sequence of a PN iff there exist marking $\mu_0, \mu_1, \ldots, \mu$ such that $\mu_0 \vec{t_1} \mu_1 \vec{t_2} \ldots \vec{t_n} \mu$. It is said that μ is reachable from μ_0 by the occurrence of σ. The marking obtained by enabled sequences are said to be reachable.

A PN is said to be safe if any reachable marking has at most one token at each place.

A PN is said to be ordinary if all of its arc weights are 1's.

Definition 4. Marked Petri net is a pair (N,Σ), where N — Petri net and $\Sigma : T \to A$ — markup function over the alphabet A. If Σ is a partial function, unmarked transition are called "λ-transitions" and marked by the "empty symbol" λ.

Algorithm of translation of MSC Document to Petri Net

MSC subset used by algorithm. Presented algorithm processes the MSC 2000 constructions set with the following limitations:

1. Time constraints are ignored.
2. Timers are processed only as separate events.
3. MSC references are allowed only in HMSC.
4. MSC reference expressions are not processed.
5. Loop boundaries are ignored as treated as <1,inf> (or <0,inf> by user's choice).
6. Sequential MSC diagram connection is treated as strong sequence (i.e. all events in the first diagram shall be finished before the second one will start).

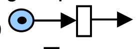
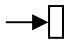
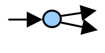
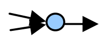
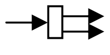
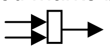
Algorithm assumes that source MSC document is syntactically and statically correct. Relation between MSC diagrams shall be given explicitly by HMSC.

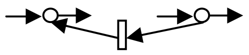
Algorithm description

Stage1 (Building of Trace Graph). During the first stage of the algorithm the Trace Graph will be build. This graph represents the traces, which a structurally possible in original MSC document (save loop iterations, which will be added during the next stage). Trace Graph creation consists of the following steps:

1. HMSC start construction translates to «start» node.
2. HMSC end construction translates to «end» node.
3. HMSC points of alternative branching translates to «alt-in» nodes.
4. HMSC points of branch joining translates to «alt-out» nodes.
5. HMSC par frame translates to «par-in» and «par-out» pair of nodes.
6. HMSC lines translates to edges between corresponding nodes.
7. MSC references in HMSC translates according the following rules (8 — 19).
8. For each MSC diagram created the nodes pair «msc-in» and «msc-out». This pair forms new synchronization zone. All edges, which corresponds to HMSC lines, drawn to this MSC reference, connect to the «msc-in» node. All edges, which corresponds to HMSC lines, drawn from this MSC reference, connect to the «msc-out» node.
9. Nodes, which corresponds to instance create events of MSC diagram, connect with «msc-in» node.
10. Nodes, which corresponds to instance end events of MSC diagram, connect with «msc-out» node.
11. Inline of «exc» and «opt» types translates as corresponding «alt» inlines.
12. Inline of «alt» type translates to «alt-in» and «alt-out» pair of nodes. Each of alternatives in this inline forms new synchronization zone.
13. Inline of «part» type translates to «par-in» and «par-out» pair of nodes. Each of alternatives in this inline forms new synchronization zone.
14. Inline of «loop» type translates to «loop-in» and «loop-out» pair of nodes. Inline body forms new synchronization zone.
15. All edges, which represent MSC lines, drawn to inline, connect to the corresponding «-in» nodes. All edges, which represent HMSC lines, drawn from inline, connect to the corresponding «-out» nodes.
16. Coregions are treated as par inline for all it's events.
17. All other valid MSC events translates to graph nodes; invalid events are skipped.
18. Edges which represent the order, explicitly shows in MSC diagram, added to graph.
19. Gates between MSC diagrams and between inlines resolved and new edges, which corresponds to gated messages and gated order relation, are added to graph.
20. In each synchronization zone edges, which do not correspond to domination relation, are removed. (Nested synchronization zone is treated as one node for outer zone.)

Stage2 (Building of Petri Net). On the second stage Trace Graph is used for resulting Petri Net building. Petri Net creation consists of the following steps:

1. Node «start» translated to  construction. Transition is marked by λ.
2. Node «end» translated to  construction. Transition is marked by λ.
3. Node «alt-in» translated to  construction. Number of out edges corresponds to number of alternatives.
4. Node «alt-out» translated to  construction. Number of in edges corresponds to number of alternatives.
5. Node «par-in» translated to  construction. Number of out edges corresponds to number of alternatives. Transition is marked marks by λ.
6. Node «par-out» translated to  construction. Number of in edges corresponds to number of alternatives. Transition is marked by λ.
7. «msc-in» and «msc-out» nodes are translated to transitions and marked by λ.

8. «loop-in» and «loop-out» pair translated to  construction. Transition is marked by λ. (If user choose <0,inf> loop boundaries, additional pass-through transition with λ mark added.)
9. Rest of graph nodes are translated to transitions marked by names of source MSC document elements.
10. Edges of the Trace Graph are translated to edges in Petri Net, which connects the corresponding nodes. If such edge connects transition with transition, addition place inserted in this edge.

Example

Let's show the algorithm work and usage on simple example. One of the simplest producer-consumer models as shown on the next MSC document with one MSC and one HMSC diagram (figures 1 and 2).

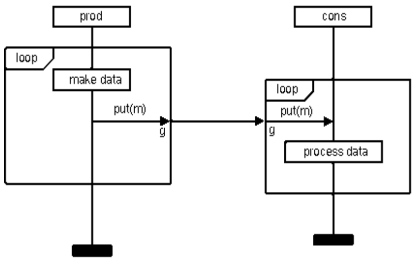


Fig. 1.

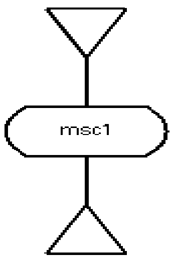


Fig. 2.

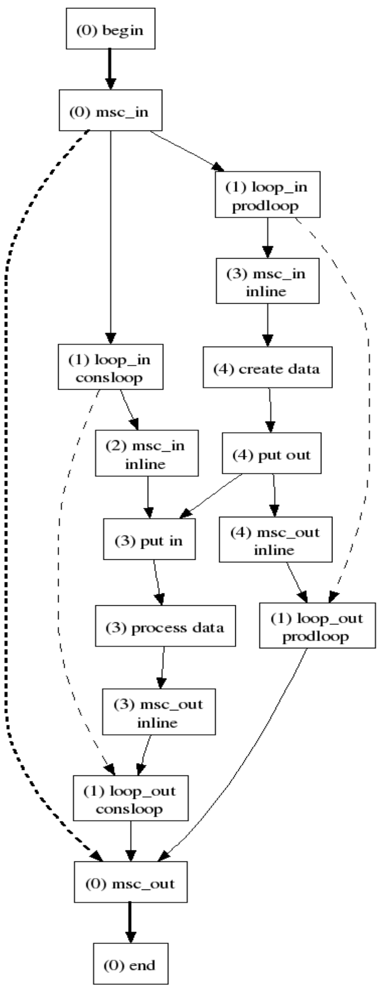


Fig. 3.

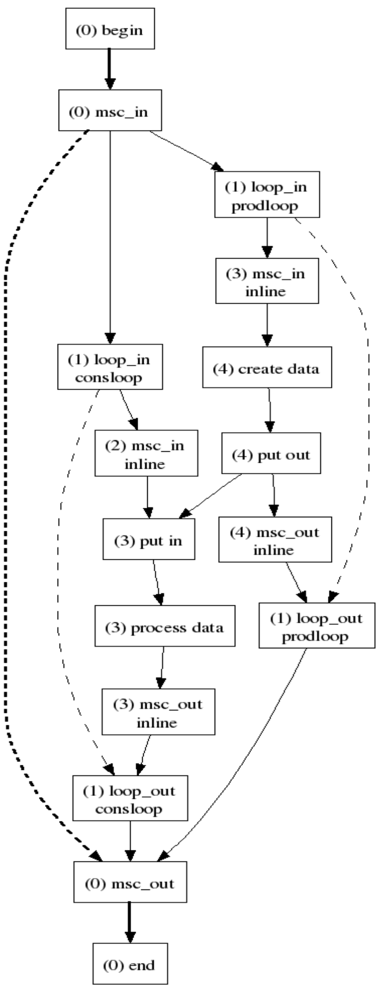


Fig. 4.

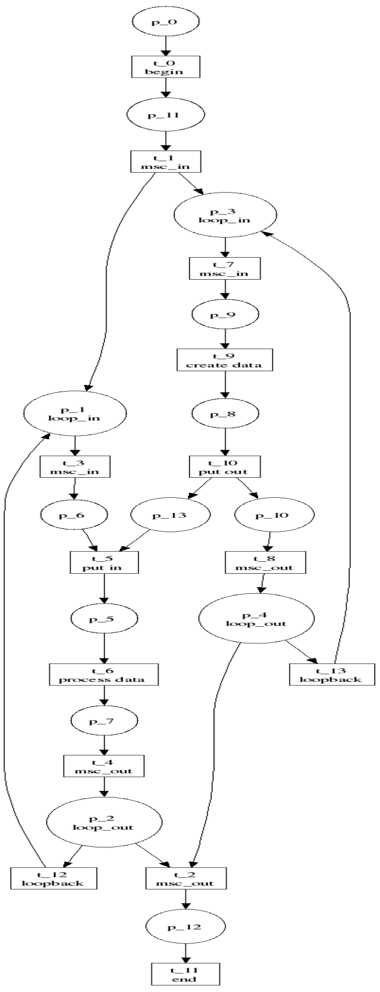


Fig. 5.

This diagram has two instances — producent (prod) and consument (cons). Producent infinitely produces some date (action «make data» in loop) and sends it to consument by message put(m) through gate g. Consument also has a loop, which allows it to infinitely process data («process data» action). After performing steps 1 — 19 we obtain the graph, shown on figure 3 and, after removing the non-dominated edges (step 20), on figure 4.

Then, after performing the Step 2 of algorithm, we obtain PN, shown on figure 5.

Now, let's try to analyze our system. First of all check liveness; to do this, calculate incidence matrix of obtained PN and find it's T-invariants. Incidence matrics is:

```
-1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 -1 0 0 0 0 0 0 0 0 1 0
0 0 -1 0 1 0 0 0 0 0 0 0 -1 0
0 1 0 0 0 0 0 -1 0 0 0 0 0 1
0 0 -1 0 0 0 0 0 1 0 0 0 0 -1
0 0 0 0 0 1 -1 0 0 0 0 0 0 0
0 0 0 1 0 -1 0 0 0 0 0 0 0 0
0 0 0 0 -1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 -1 0 0 0
0 0 0 0 0 0 0 1 0 -1 0 0 0 0
0 0 0 0 0 0 0 0 -1 0 1 0 0 0
1 -1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0 0 -1 0 0
0 0 0 0 0 -1 0 0 0 0 1 0 0 0
```

And truncated solution set of the equation Ax=0 is (0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1), which shows, that transitions in our PN are alive except t_0 (begin MSC document), t_1 (begin MSC diagram), t_2 (begin MSC diagram) and t_11 (end MSC document) — just as expected. So, all events in the source can be executed infinitely. Then, check for PN's boundness. To perform this check, reduce PN as shown in [5] and create it's covering tree (figures 6,7).

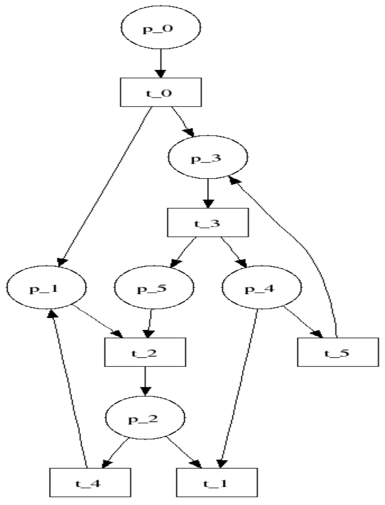


Fig.6.

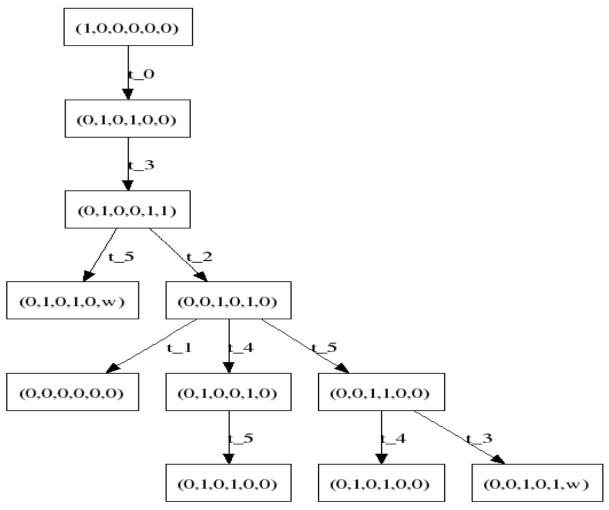


Fig. 7.

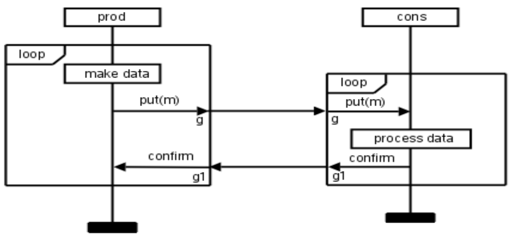


Fig. 8.

As we can see on covering tree, place p_5 of reduced PN is unbounded. It corresponds to place p_13 of original PN and means, that infinite number of messages put(m) can be stacked in original MSC (it can happens if consumer process data slower than producer creates it). To fix this, let's change original MSC by adding synchronization between consumer and producer (figure 8).

Now let's see on reduced PN and covering tree for corrected document (other steps have been omitted to save the article space).

Now we can see (figure 10) that our PN is bounded, so synchronization was enough to fix the found problem.

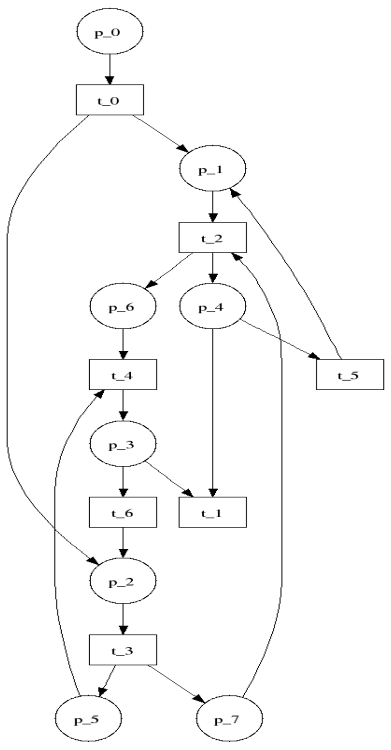


Fig. 9.

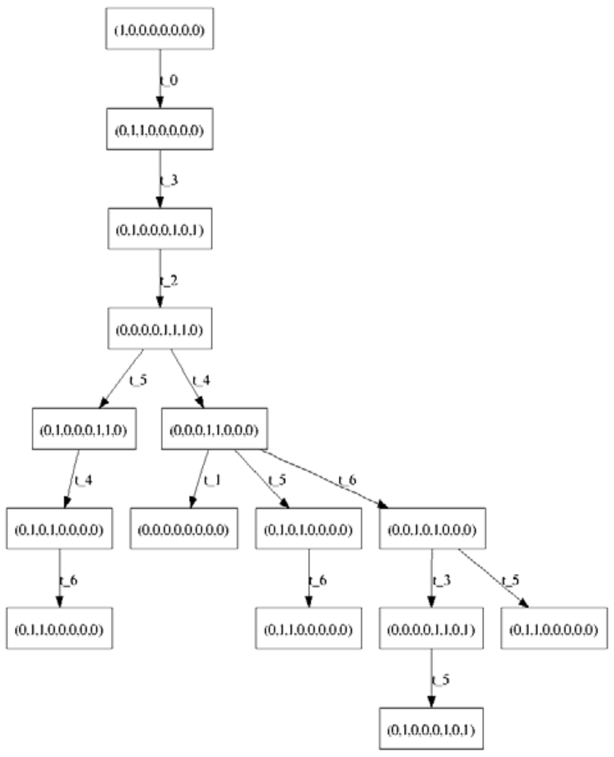


Fig. 10.

Conclusion

For conclusion we note, that the most significant feature of the given algorithm is its increased usability. The input set of MSC elements has been extended. Most of the existing design approaches require decomposing from the beginning the overall functionality into components. Extended input set of MSC elements in the given version of the translation algorithm makes it possible to apply different composing/decomposing techniques for subsequent PNs analysis.

The algorithm of translation MSC diagrams into Petri net, presented in the paper, considers only the subset of MSC language, therefore full implementation of reference MSC expressions and processing time constraints are our further research direction.

References

[1] ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU_TS, Geneva (2000).

[2] ITU-TS Recommendation Z.120 Annex B: Message Sequence Chart Annex B: Formal semantics of Message Sequence Charts. ITU_TS, Geneva (2001).

[3] Kryvyy, S., Matveyeva, L.: Algorithm of Translation of MSC-specified System into Petri Net. Fundamenta Informaticae, Vol.79 (2007), 1–15.

[4] Kryvyy, S., Matveyeva, L., Chugayenko O.: Extension of Algorithm of Translation of MSC specified system into Petri Net. Proc. of the CS&P'2007, 2007 – v2 – p.376—388

[5] Murata T. Petri Nets: Properties, Analysis and Verification. Proc. of the IEEE, 1989 — 77 — N4, p.65—74

Authors' Information

Kryvyi Sergii – *Glushkov Institute of Cybernetics NAS Ukraine, Ukraine, Kiev, 03187, 40 Glushkova Street, Institute of Cybernetics, e-mail: krivoi@i.com.ua*

Chugayenko Oleksiy – *Institute of Cybernetics NAS of Ukraine, Ukraine, Kiev, 03187, 40 Glushkova Street, Institute of Cybernetics, e-mail: avch@avch.org.ua*