Application of Coloured Petri Net for Agent Control and Communication in the ABAsim Architecture

Antonín Kavička[1] and Michal Žarnay[2]

[1] Faculty of Electrical Engineering and Informatics, University of Pardubice,
Nám. Čs.legií 565, CZ-532 10 Pardubice, The Czech Republic
Antonin.Kavicka@upce.cz

[2] Faculty of Management Science and Informatics, University of Žilina,
Univerzitná 8215/1, SK-01026 Žilina, The Slovak Republic
Michal.Zarnay@fri.uniza.sk

Abstract. Petri nets represent a convenient formalism for description of the operational logic of internal agent components within agent-based architectures of simulation models using message-oriented communication paradigm. The approach supports higher flexibility of simulation models as well as formal analysis related to relevant parts of the models. The ABAsim architecture (as an example) already utilizes place/transition Petri nets for description of internal components of agents. Presented modified approach pays attention to an application of non-hierarchical coloured Petri nets (describing behavioural rules of autonomous agents) instead of place/transition ones because of higher modelling capabilities.

Keywords: Coloured Petri net, agent-based simulation, message-oriented communication.

1 Introduction

During the last decade a significant progress was achieved in the area of micro-simulation models reflecting transportation logistic systems [1, 2]. The models are *flexible*, which means that they are namely: (i) composed of reusable components and sub-models, (ii) configurable on a conceptual level and (iii) ready for changes of granularity related to sub-models. A proprietary agent-based architecture (called *ABAsim* *A*gent-*B*ased *A*rchitecture of *sim*ulation models) has been developed and it has been successfully applied within many simulation studies reflecting transportation and logistic systems (an example is represented in [4]). The architecture utilizes simulation model decomposition into individual agents (concentrated on distinct tasks), which are organized within hierarchical structure and communicate by means of sending messages. Each agent is composed of internal components (potentially using internal communication), whereas their logic can be described either

imperatively (with the help of programme routines) or declaratively (e.g. applying formalism of Petri nets) – the respective specification is activated by means of system interpreter during the run of simulation programme.

Up to now, the ABAsim architecture has utilized a subclass of *place*/*transition Petri net* (P/T PN), called *ABA-graph* [3] for description of control agent components denoted as managers. Presently is the attention paid to an application of a subclass of coloured Petri net, named *ABA-CPN*, which brings more flexible approach in defining the component descriptions. It includes, for example, easier and more natural construction of conditional branching and differentiation of various message instances.

The paper is organised as follows: sections two and three shortly describe background of the paper: autonomous agents and ABAsim architecture. Section four explains the ABA-CPN definition in detail, section five provides illustratory example for it and section six discusses conventions for notation of elements in the ABA-CPN followed by conclusion of the paper.

Since it has been too challenging to combine all goals in one example, while keeping its size in reasonable limits, there are two examples. The first accompanies explanations of the definition – it includes all properties from definition, it uses only symbolic names and it has no connection to examples from real world. The second example is built on a simple real case from an ABAsim simulation model of service system, including meaningful labels – it illustrates usage of notation conventions.

2 Paradigm of autonomous agents

Let us remind the generally-respected agent definition [7]: *Agent is an encapsulated computer system situated in some environment and capable of flexible, autonomous action in that environment in order to meet its design objectives*, where the agent features are as follows:

- *Autonomy* – i.e. the agent is able to work autonomously without exogenous interventions, entirely able to control its activities and inner status.
- *Social behaviour* – is based on the agent's interaction with other agents (or with human beings) by means of some communication mechanism or language.
- *Re-activeness* – the agent responds to external influences from its surrounding.
- *Pro-activeness* – the agent acts with initiative and goal-orientation.

The agent realizes, according to its mission, its own life-cycle: *sensing – decision making – acting* (within its life space) using the support of *solving* (focused on making solution proposals) and *communicating* with other agents (eventually with human operators). If the agent detects a problem or a situation beyond its delegated competence, it informs other agents about the need for a corresponding solution.

3 Brief summary of the ABAsim architecture

The agent-based architecture of simulation models of *ABAsim* was mainly developed for simulation of queuing, transportation and logistic systems. Those systems can be considered (from the viewpoint of order processing) strictly *hierarchical*. The order (the customer) entering the system initiates a recursive sequence of suborders, according to the rules of competence redistribution.

The entities (orders/customers and resources) within the frame of the ABAsim architecture are divided into specialized classes with defined behavioural rules. This means that the responsibility for the behaviour of these entities is taken over by their superior subjects (agents). It is necessary in most cases to transfer service resources to the customer (or vice versa), in order to realize the service activity, i.e. frequent and complex transposition processes are typical within such systems.

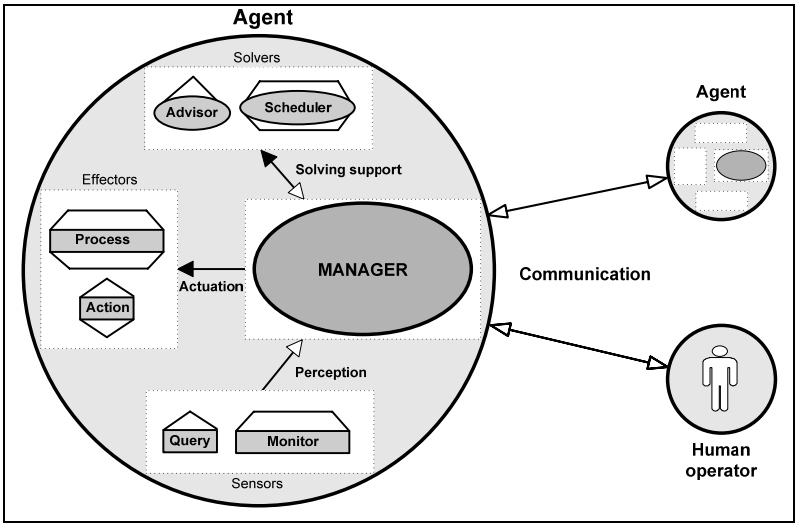

Fig. 1. Agent's decomposition

3.1 Agent components

Each agent can be decomposed into the following groups of *internal components* (presented on fig. 1):

a) The first, control and decision making component (called the manager) is responsible for making decisions and for inter-agent communication. In addition, the manager represents the central agent component because it initiates the work of other internal components and can also communicate with all of them.

b) The group of *sensors* is specialized for mining information from the system's state space. This group is composed of two kinds of components - the *query*

delivers the required forms of information instantly, and the *monitor* scans the state space in some time intervals and continuously brings important information to the manager.

c) The next group, called *solvers*, provides solutions for problems to the manager, which can accept them or ask for alternative ones. The *advisor* represents a passive component, which is able to react only to the manager's requests for delivery of proposals for problem-solving. A typical advisor can be represented e.g. by an optimization algorithm, an artificial neural network, a fuzzy regulator or a human operator. On the other hand, the *scheduler* (focused on a restricted scope of problems) works continuously for the manager, on the basis of either a priori rosters or schedules, which have been created (e.g. related to the allocation of resources), or by making its own dynamic forecast for a defined time interval.

d) The last component group includes *effectors* (actuators), which make changes to system status after receiving corresponding instructions from the manager. No other agent components are allowed to make these changes. An *action*-component makes instant state changes (e.g. switch traffic lights, close a train's doors), while a *process*-component (e.g. a crane's movement) makes them continuously until its task is finished.

The effectors, sensors, and solvers are, for brevity, given the umbrella term of manager's *assistants*, and can be further distinguished as:

- *Continual assistants*, the activities of which fill up some interval in the simulation time (processes, monitors, and schedulers).
- *Instant assistants*, which are active only in discrete instants of simulation time (actions, queries and advisors).

The question arises, how to realize the internal agent components appropriately - they can be described alternatively either

- using *imperative approach* (implementation of program routines constructed in a given source code), or
- by means of *declarative forms* (connected with some kind of symbolic formalism), which are reflected by structured input data and read by a corresponding interpreter; e.g. Petri nets represent effective formalisms appropriate for describing agent internal components.

3.2 Community of agents and its structure

Simulation models for simple real systems could be composed of only one agent; however, the simulation of complex service systems is obviously connected with a *multi-agent approach* using the agents within some organizational structure. Let us remark that the philosophy of the ABAsim architecture was also partly inspired by the paradigm of reactive agents, which is based on a society of reactive rather than proactive agents. The intelligence of such society *emerges* when one observes the whole community and not its separate members (individually of relatively low intelligence).

To summarize the philosophy of the ABAsim operation: The control role is played by mutually communicating managers (supported by sensors and solvers), which initiate the activities of effectors at the correct time instants and under particular conditions.

3.3 Communication mechanism

One way to realize inter-agent communication is to use standard communication languages (e.g. KQML or FIPA-ACL). Another approach is to implement a customized communication mechanism able to reflect, in the best way, the features of the respective architecture.

Communication within the ABAsim architecture is based on a simple, original mechanism applied to *inter-agent* and also *intra-agent* communications. As was already mentioned, inter-agent communication is made by manager components, and intra-agent communications are realized between the managers and their assistants. Both kinds of the ABAsim-communications utilize exclusively the paradigm of sending messages (from this viewpoint, the ABAsim represents *message-oriented architecture*).

The following description characterizes selected kinds of messages used within the ABAsim architecture in a simple way. *Notice*–messages contain some information for the addressee without expecting any answer, *Request*–messages carry specific demands, which are expected to be satisfied or supplied by means of corresponding *Response*–messages. Continual assistants are initiated by *Start*–messages (sent by superior managers), whereas *Finish*–messages (sent by continual assistants) delivered to corresponding managers, indicate completion of an activity related to relevant continual assistant. In addition, managers exploit *Execute*–messages in order to obtain promptly required results from their inferior instant assistants. Finally, *Hold*–messages exclusively mediate the augmentation of simulation time. They involve so-called *time stamps*, which define duration of their deliveries (equal or greater than the current simulation time). The attributes *sender* and *addressee* contain the same values – i.e. the continual assistants send those messages to themselves with some time delay. Thus, after sending *Hold*–message, the continual assistant remains idle and resumes its activity after the message returns. We have to emphasize that the augmentation of the simulation time is realized exclusively by continual assistants, i.e. synchronization of simulation time is based on synchronization of these components.

3.4 ABAsim versus other agent-based architectures

Seeing that general paradigm of autonomous agents influenced the design and development of the ABAsim architecture, it is only natural that the architecture shares some common principles with other agent-based simulation architectures that were inspired by the same paradigm.

Among many, it can be mentioned for example Cougaar architecture [8] (with similar hierarchical organization of agent communities or agent decomposition to

simpler executive units) or architecture HIDES [9], which shares the same view on importance of hierarchical structure of agents reflecting modelled system and supports forming of agent communities responsible for specific tasks. Since its beginning, ABAsim architecture has been oriented to creation of simulation models of complex large-scale service systems, mainly transportation systems, with emphasis on flexibility for simulation model designers, programmers as well as for end-users of simulation models. Since comparison of the ABAsim architecture with outlined or other simulation architectures and detailed explanation of benefits that it introduces is out of scope of this paper, we recommend the reader to pay attention to the following papers [10,11,12].

4 Specification of ABA-CPN

Coloured Petri net (CPN) describing the logic of an agent component can be defined within an environment of a specific software tool, where an analysis of the net is supposed to be carried out. Analysed nets are consequently made available (via respective data files) to a simulation engine of the ABAsim architecture. A relevant interpreter maintains then the evolution of the nets during simulation.

For the needs of simulation models based on the ABAsim architecture, modelling capabilities of the non-hierarchical coloured Petri net can be restricted. This results in a specific subclass of coloured Petri net, called *ABA-CPN*, partially inspired by the mentioned ABA-graph.

The following definition builds upon CPN definitions from [5]. Since it is quite complex, we divide it into four parts that are interleaved with comments and illustrations on a small example depicted on the figure 2 and built just for that purpose.

Definition 1:
ABA-CPN represents a subclass of *CPN* = ($\boldsymbol{\Sigma, P, T, A, N, C, G, E, I}$) that satisfies the following:

(i) $\boldsymbol{\Sigma}$ is a finite set of non-empty types, called *colour sets*.

(ii) The finite set of *places* $\boldsymbol{P} = \{\boldsymbol{p_{in}}\} \cup \{\boldsymbol{p_{out}}\} \cup \boldsymbol{P_S}$, where $\boldsymbol{p_{in}}$ is called *input place*, $\boldsymbol{p_{out}}$ *output place* and $\boldsymbol{P_S}$ is composed of *internal places* and the three sets are mutually disjoint.

(iii) The finite set of *transitions* $\boldsymbol{T} = \boldsymbol{T_D} \cup \boldsymbol{T_A} \cup \boldsymbol{T_S} \cup \boldsymbol{T_B}$, $\boldsymbol{T} \neq \emptyset$, where elements of $\boldsymbol{T_D}$ are denoted as *decision transitions*, elements of $\boldsymbol{T_A}$ as *assistant transitions*, elements of $\boldsymbol{T_S}$ as *sending transitions* and elements of $\boldsymbol{T_B}$ as *standard transitions*, the four sets are all mutually disjoint and $\boldsymbol{T} \cap \boldsymbol{P} = \emptyset$.

(iv) $\boldsymbol{A}$ is a finite set of *arcs* such that $\boldsymbol{P} \cap \boldsymbol{A} = \boldsymbol{T} \cap \boldsymbol{A} = \emptyset$.

(v) $\boldsymbol{N}$ is a *node* function defined from $\boldsymbol{A}$ into $(\boldsymbol{P}\times\boldsymbol{T}) \cup (\boldsymbol{T}\times\boldsymbol{P})$.

(vi) $\boldsymbol{C}$ is a *colour function* defined from $\boldsymbol{P}$ into $\boldsymbol{\Sigma}$.

(ii) The set of places is divided to subsets in order to distinguish specific kinds of places. A token in the input place $\boldsymbol{p_{in}}$ corresponds to an input message and a token in the output place $\boldsymbol{p_{out}}$ corresponds to an output message associated with a relevant

agent component. In the illustration net on the figure 2, the input place $p_{in} = p_1$, the output place $p_{out} = p_9$ and there are seven intern. places: $P_S = \{p_2, p_3, p_4, p_5, p_6, p_7, p_8\}$.

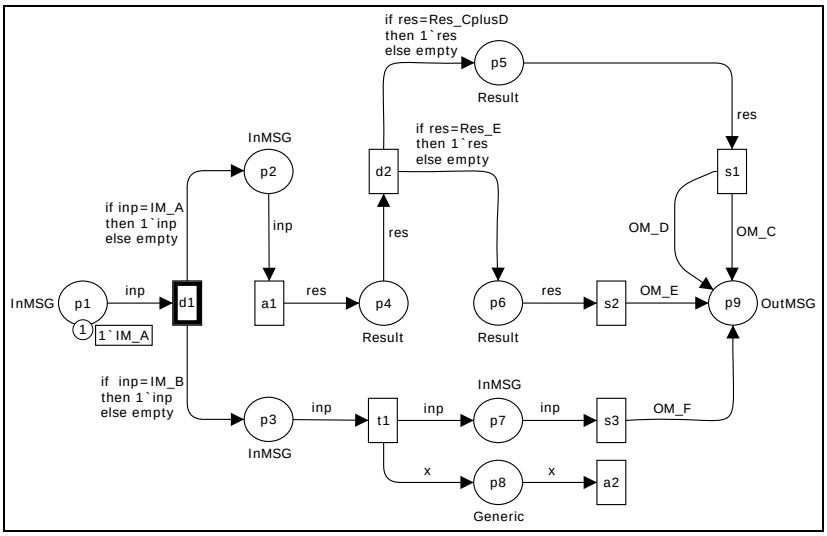

Fig. 2. Illustration net related to definition of ABA-CPN

(iii) The set of transitions is divided to four subsets in order to distinguish various actions in the ABA-CPN application. Decision transitions (involved in T_D) represent points of variant conditional branching (transitions d_1 and d_2 within the illustration net), assistant transitions (folded in T_A) reflect actions of corresponding instant assistants (a_1, a_2), sending transitions (elements of T_S) reflect sending messages to other model components (s_1, s_2, s_3), while only standard transitions (contained in T_B) have no specific meaning in the ABA-CPN application (t_1).

(i) + (vi) In the illustration net, there are four colour sets, i.e. $\Sigma = \{$*InMSG*, *Result*, *Generic*, *OutMSG*$\}$. Examples of colour function are: $C(p_1)$ = *InMSG* and $C(p_6)$ = *Result*. The *InMSG* colour set consists of two values *IM_A* and *IM_B*, the *OutMSG* colour set can have four different values: *OM_C*, *OM_D*, *OM_E* and *OM_F*. Colour set *Result* contains two values *Res_CplusD* and *Res_E*, and finally colour set *Generic* has a single value *e*.

Definition 1 cont'd:

(vii) E is an *arc expression function* defined from A into *arc expressions* (specified in [5]).

(viii) For arcs, there are the following specifications:

a) There is no such pair of arcs a_1, a_2, $a_1 \neq a_2$ with $p(a_1) = p(a_2) \wedge t(a_1) = t(a_2) \wedge ((N(a_1) \in T \times P \wedge N(a_2) \in P \times T)$, where $p(a)$ is the place and $t(a)$ is the transition of $N(a)$,

b) Every arc $a \in A$ belongs to one of the following categories:

- If $t(a) \in T_D \wedge N(a) \in T \times P$, a is called *decision arc* and its arc expression $E(a)$ contains a condition expression for variant branching,

- For all $t(a) \notin T_D$ and all $t(a) \in T_D$: $N(a) \in P \times T$, a is one of the following: *constant arc*, if $E(a)$ is composed of a single constant only, or *elementary variable arc*, if $E(a)$ consists of one variable only.

In the section (viii), the former specification states that self-loops are not admissible, so ABA-CPN represents a *pure net*. The latter specification introduces allowable arc expressions that divide arcs into categories. Arc expressions of all arcs going out of decision transitions (named *decision arcs*) have a condition expression. In the illustration net, it is the case of arcs (d_1, p_2), (d_1, p_3), (d_2, p_5) and (d_2, p_6). Arc expressions of arcs starting from any other transition can contain only one constant (*constant arcs*) or one variable (*elementary variable arcs*). In the illustration net, (p_1, d_1), (p_2, a_1), (a_1, p_4), (p_4, d_2), (p_5, s_1), (p_6, s_2), (p_3, t_1), (t_1, p_7), (p_7, s_3), (t_1, p_8), (p_8, a_2) are elementary variable arcs. Remaining arcs are classified as constant arcs.

Definition 1 cont'd:

(ix) G is a *guard function* defined from T into *guard expressions* (specif. in [5]).

(x) Elements from the sets P and T have the following additional properties (related to use of the formalism):

a) $(\forall\, t \in T: \neg\exists\, a \in A: N(a) = (t, p_{in})) \wedge (\exists_1\, a \in A: N(a) = (p_{in}, t), t \in T)$,

b) $(\exists\, a \in A: N(a) = (t, p_{out}), t \in T) \wedge (\forall\, t \in T: \neg\exists\, a \in A: N(a) = (p_{out}, t))$,

c) $\forall\, p \in P_S: (\exists\, a \in A: N(a) = (t, p), t \in T) \wedge (\exists_1\, a \in A: N(a) = (p, t), t \in T)$,

d) $\forall\, t \in T_B: \exists\, a \in A: N(a) = (p, t), p \in P$,

e) $\forall\, t \in T \setminus T_B: \exists_1\, a \in A: N(a) = (p, t), p \in P$,

f) $\forall\, t \in T_D \cup T_S : \exists\, a \in A: N(a) = (t, p), p \in P$,

g) $\exists_1\, t \in T_D: \exists_1\, a \in A: N(a) = (p_{in}, t)$, where t is denoted as *input transition*,

h) $t \in T: (\exists\, a \in A: N(a) = (t, p_{out})) \vee (\neg\exists\, a \in A: N(a) = (t, p), p \in P)$ is called *output transition*,

i) $t \in T: (\neg\exists\, a \in A: N(a) = (p_{in}, t)) \wedge (\neg\exists\, a \in A: N(a) = (t, p_{out})) \wedge (\exists\, a \in A: N(a) = (t, p), p \in P)$ is named as *internal transition*,

j) $\forall\, t \in T: G(t) = \varnothing$.

In the section (x), properties a), b) and c) deal with places: the *input place* has no incoming and only one outgoing arc (place *p1* in the illustration net); the *output place* can have only incoming arcs (place p_9) and all the other places have at least one incoming and just one outgoing arc.

Properties d) to j) deal with transitions. All transitions have at least one input arc. *Decision*, *assistant* and *sending transitions* have just one incoming arc, *standard transitions* may have more incoming arcs. As for outgoing arcs, they must be present by *decision* and *standard transitions* (case of transitions d_1, d_2, s_1, s_2 and s_3 in the illustration net), while not necessarily by the other transition types (e.g. transition a_2 has no outgoing arc). *Input transition* is just one and it is the one that follows the *input place* (transition d_1 after input place p_1). *Output transitions* are those having at least one outgoing arc to *output place* or not having any outgoing arc (transitions s_1, s_2, s_3 and a_2). All the other transitions are named *internal* (transitions d_2, a_1 and t_1). No transition disposes of a guard.

Definition 1 cont'd:

(xi) Petri net built from all elements of $\boldsymbol{P}$, $\boldsymbol{T}$, and $\boldsymbol{A}$ represents an *acyclic* net structure.

(xii) $\boldsymbol{I}$ is an *initialization function* defined from $\boldsymbol{P}$ into *closed expressions* (specified in [5]).

(xiii) The set of *admissible initial markings* $\boldsymbol{M_0} \subset \boldsymbol{I}(\boldsymbol{P})$ is defined as follows: $\boldsymbol{M_0} = \{{}^{j}\boldsymbol{M_0} \mid \boldsymbol{j} = 1,2, \ldots, |\boldsymbol{C}(\boldsymbol{p_{in}})|\}$, where ${}^{j}\boldsymbol{M_0}$ denotes j-th initial marking in the set, for which holds: $\forall\ \boldsymbol{p} \in \boldsymbol{P}$: $|{}^{j}\boldsymbol{M_0}(\boldsymbol{p})| = 1$, if $\boldsymbol{p} = \boldsymbol{p_{in}}$, and $|{}^{j}\boldsymbol{M_0}(\boldsymbol{p})| = 0$, if $\boldsymbol{p} \neq \boldsymbol{p_{in}}$, and for $i \neq j$, ${}^{i}\boldsymbol{M_0}(\boldsymbol{p}) \neq {}^{j}\boldsymbol{M_0}(\boldsymbol{p})$.

(xiii) The tokens of different colours in ABA-CPN reflect differently filled message forms, which are in the ABAsim architecture utilised for communication purposes. An admissible initial marking of ABA-CPN allows an occurrence of just one token in the whole net which is located in the input place $\boldsymbol{p_{in}}$; all other places dispose of no tokens. This represents a state, where an input message waits in the input place to be processed and there are no other messages being processed. Since the ABA-CPN is constructed to process all relevant input messages for the given manager separately and the input messages are represented by tokens from the colour set of the input place $\boldsymbol{C}(\boldsymbol{p_{in}})$, the set of admissible initial markings $\boldsymbol{M_0}$ contains as many markings as is the number of input messages, i.e. $|\boldsymbol{C}(\boldsymbol{p_{in}})|$, and ${}^{j}\boldsymbol{M_0}$ denotes the j-th initial marking from the set. In the illustration net on the fig. 2, an admissible initial marking is displayed: the input place $\boldsymbol{p_1}$ contains one token of colour *IM_A*, while all other places are empty. This represents a situation of input message *IM_A* coming to the agent to be processed.

5 Example of ABA-CPN

Let us illustrate application of ABA-CPN to a description of manager component involved within an agent named *Resource controller*. The agent represents a part of a model (based on the ABAsim architecture) reflecting simple service system (fig. 3).

The system's customers, coming into the system from its surroundings, are supposed to be consecutively involved in two kinds of serving activities and after their finishing they leave the system. While the first kind of services is carried out (within the process *Service A*) with the help of an immovable service resource (i.e. the customer has to come up to its place), the second one is made by a mobile resource (exploiting the process *Service B*), which is able to move (using the process *Resource transfer*) directly to the customer's spot.

A simulator of the mentioned system is composed of three agents: the *Surroundings* agent, the *Service controller* agent and the *Resource controller* agent. The first one is responsible for a connection between the system and its surroundings (deals with arrivals and departures of individual customers), the second one organizes all concurrently running service activities and finally the third one is competent to assign/release service resources and to make disposals for their transpositions. Such designed simulator can be specified in the form of the *ABAsim model*, the simplified form of which (excluding instant assistants for the sake of simplification) is depicted

on fig. 3. The mentioned picture shows all essential model components and their communication links.

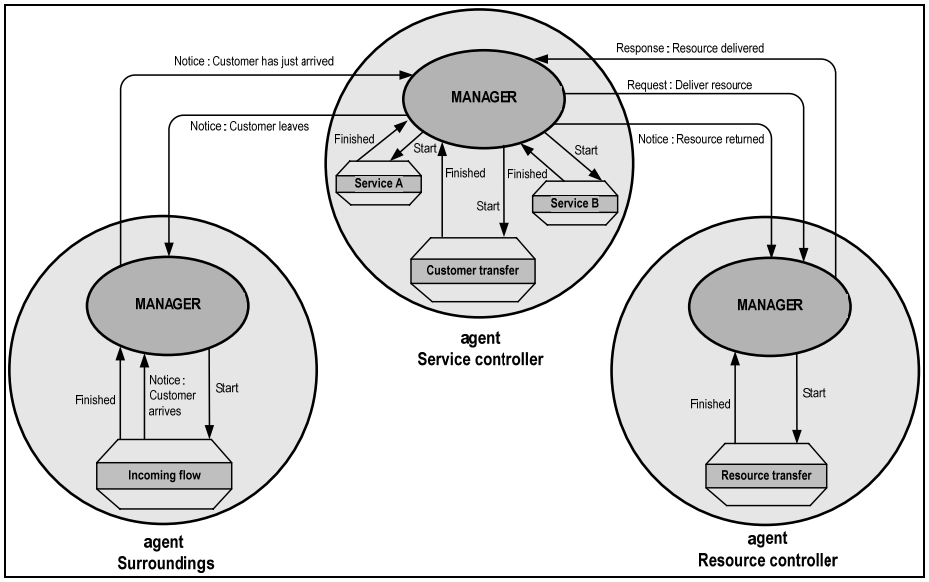

Fig. 3. Simplified ABAsim model (with commun. links) of the elementary service system

The ABA-CPN of the manager component (encapsulated within the *Resource controller* agent) is shown on the fig. 4. It disposes of the following characteristics (according to the ABA-CPN definition):

- Set of places $\boldsymbol{P} = \{\boldsymbol{p_i} \mid \boldsymbol{i} = 1,\ldots,16\}$, where: $\boldsymbol{p_{in}} = \boldsymbol{p_1}$, $\boldsymbol{p_{out}} = \boldsymbol{p_{16}}$, $\boldsymbol{P_S} = \{\boldsymbol{p_i} \mid \boldsymbol{i} = 2,\ldots, 15\}$.
- Set of transitions $\boldsymbol{T} = \boldsymbol{T_D} \cup \boldsymbol{T_A} \cup \boldsymbol{T_S} \cup \boldsymbol{T_B}$, where $\boldsymbol{T_D} = \{\boldsymbol{d_i} \mid \boldsymbol{i} = 1,\ldots, 4\}$, $\boldsymbol{T_A} = \{\boldsymbol{a_i} \mid \boldsymbol{i} = 1,\ldots, 8\}$ (elements of $\boldsymbol{T_A}$ are commented in the table 1), $\boldsymbol{T_S} = \{\boldsymbol{s_i} \mid \boldsymbol{i} = 1,2\}$ and $\boldsymbol{T_B} = \{\boldsymbol{t_1}\}$; the input transition is represented by $\boldsymbol{d_1}$ and output transitions by $\boldsymbol{s_1}$, $\boldsymbol{s_2}$, $\boldsymbol{a_4}$, $\boldsymbol{a_8}$.

Table 1. Characteristics of assistant transitions involved in ABA-CPN presented within fig. 4

Transition	reflects component	Transition	reflects component
$\boldsymbol{a_1}$	instant assistant - advisor *Selection from avail. resources*	$\boldsymbol{a_5}$	instant assistant - action *Resource assign. to applicant*
$\boldsymbol{a_2}$	instant assistant - action *Resource release*	$\boldsymbol{a_6}$	instant assistant - action *Dequeue applicant*
$\boldsymbol{a_3}$	instant assistant - query *Queue for released resource?*	$\boldsymbol{a_7}$	instant assistant - query *Resource transfer to applicant?*
$\boldsymbol{a_4}$	instant assistant - action *Enqueue applicant*	$\boldsymbol{a_8}$	instant assistant - action *Update transfer statistics*

- The set of admissible initial markings $\boldsymbol{M_0} = \{{}^1\boldsymbol{M_0}, {}^2\boldsymbol{M_0}, {}^3\boldsymbol{M_0}\}$ represents three different input messages that can reach the manager component of the *Resource_controller* agent and is defined as follows:
  ${}^1\boldsymbol{M_0}(\boldsymbol{p_1}) = \{REQ\_Deliver\_resource\}$ and ${}^1\boldsymbol{M_0}(\boldsymbol{p_j}) = \emptyset$, $\boldsymbol{j} = 2, \ldots, 16$;
  ${}^2\boldsymbol{M_0}(\boldsymbol{p_1}) = \{NTC\_Resource\_returned\}$ and ${}^2\boldsymbol{M_0}(\boldsymbol{p_j}) = \emptyset$, $\boldsymbol{j} = 2, \ldots, 16$;
  ${}^3\boldsymbol{M_0}(\boldsymbol{p_1}) = \{FIN\_Transfer\}$ and ${}^3\boldsymbol{M_0}(\boldsymbol{p_j}) = \emptyset$, $\boldsymbol{j} = 2, \ldots, 16$.

  Occurrence graph of the illustrative ABA-CPN for the initial marking ${}^1M_0(p_1)$ is presented on fig. 5 (using the *CPN Tools* software). It shows that processing of input message *REQ_Deliver_resource* can be ceased in three terminal states: either produce one output message (state space nodes 13 and 14) or no output message (node 7). The output message can be one of these two: *START_Transfer* (node 14) or *RESP_Resource_delivered* (node 13).

6 Conventions for the notation of ABA-CPN

In order to carry out unambiguous transformation of the ABA-CPN into relevant data structures of simulation model program and to enable its correct evolution using a specialized interpreter, there are certain conventions for the notation of the ABA-CPN elements. ABA-CPN is constructed within the environment of the *CPN Tools* software (developed at the University of Aarhus, Denmark [6]). Notation conventions are as follows (see an example on fig. 4):

a) places are denoted as $\boldsymbol{p_i}$ for $\boldsymbol{i} = 1..\boldsymbol{n}$, where $\boldsymbol{n} = |\boldsymbol{P}|$; $\boldsymbol{p_1}$ denotes *input place* and $\boldsymbol{p_n}$ corresponds to *output place* of the net,
b) *decision transitions* are denoted as $\boldsymbol{d_i}$ for $\boldsymbol{i} = 1..\boldsymbol{m}$, $\boldsymbol{m} = |\boldsymbol{T_D}|$, (as $\boldsymbol{d_1}$ we denote *input transition*), *standard transitions* are denoted as $\boldsymbol{t_i}$ for $\boldsymbol{i} = 1..\boldsymbol{k}$, $\boldsymbol{k} = |\boldsymbol{T_B}|$, *assistant transitions* dispose of description $\boldsymbol{a_i}$ for $\boldsymbol{i} = 1..\boldsymbol{l}$, $\boldsymbol{l} = |\boldsymbol{T_A}|$ and *sending transitions* are associated with notation $\boldsymbol{s_i}$ for $\boldsymbol{i} = 1..\boldsymbol{r}$, $\boldsymbol{r} = |\boldsymbol{T_S}|$,
c) descriptions of *constant arcs* or *elementary variable arcs* use declared constants, elements of colour sets and variables, each arc with one symbol only,
d) *decision arcs* typically dispose of an expression in the form 'if *var* = *const* then *case1* else *case2*', where *var* or *const* represent relevant variable or constant, *case1* or *case2* reflect elementary notations composed of one variable or one constant or the key word *empty*.

Conventions for denoting variables and constants in arc expressions follow the goal that the designer should be able to control reactions to relevant "content" of a token (in this case content of a relevant message form). In addition, using the conventions enables to manage "movements" of token instances within the net.

For purpose of the following explanation, let us consider $\boldsymbol{s}$ as a string, whereas $\boldsymbol{i}$-th character of the string is denoted as $\boldsymbol{s^i}$.

Proposal of conventions related to denoting variables and constants in arc expressions of arcs adjacent with a transition $\boldsymbol{t} \in \boldsymbol{T}$, is as follows:

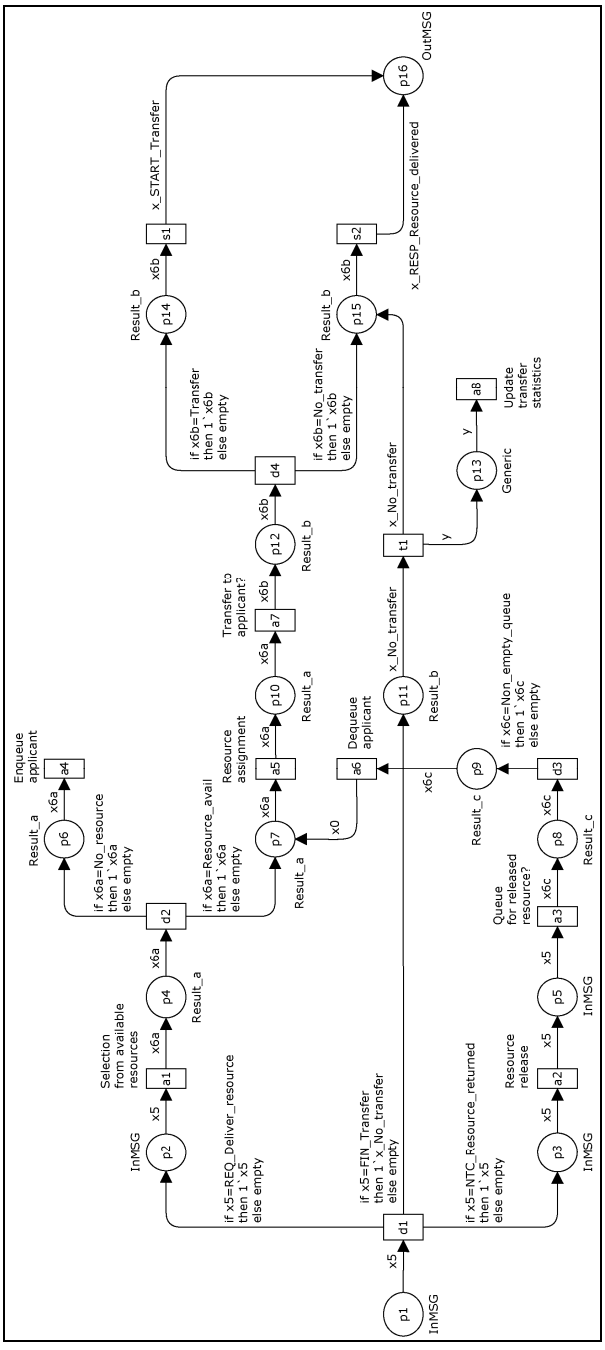

Fig. 4. Coloured Petri net reflecting a manager related to Resource controller agent

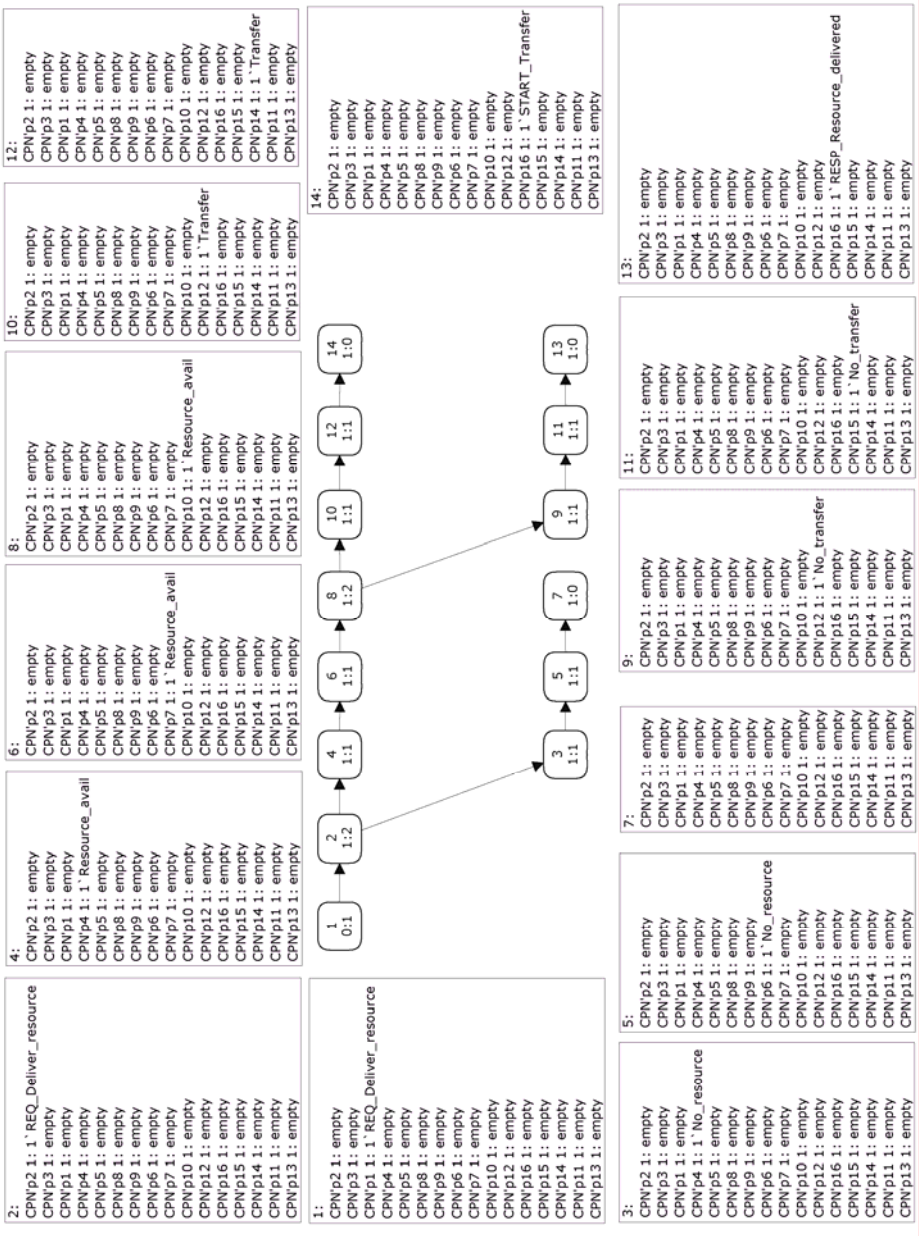

Fig. 5. Occurrence graph for $M_0(p_1)$=REQ_Deliver_resource, $M_0(p_i)$=∅, i = 2,…,14

a) For firing transition $\boldsymbol{t} \in \boldsymbol{T}$, it holds that token instance removed from the place $\boldsymbol{p} \in \boldsymbol{P}$: $\boldsymbol{N}(\boldsymbol{a}) = (\boldsymbol{p}, \boldsymbol{t})$, $\boldsymbol{a} \in \boldsymbol{A}$ is consequently placed (itself or its identical copy) to a place $\boldsymbol{q} \in \boldsymbol{P}$: $\boldsymbol{N}(\boldsymbol{a}) = (\boldsymbol{t}, \boldsymbol{q})$, $\boldsymbol{a} \in \boldsymbol{A}$ under following conditions:

- Token instance from the place $\boldsymbol{p}$ is removed by means of a constant or a variable (let's call it ***input*** for the sake of this explanation) contained within expression of the arc ($\boldsymbol{p}$, $\boldsymbol{t}$).
- Positioning token instance to the place $\boldsymbol{q}$ is mediated by a constant or a variable (let's call it ***output***) encapsulated within expression of the arc ($\boldsymbol{t}$, $\boldsymbol{q}$).
- For the mentioned identifiers of constants or variables, it holds: $\boldsymbol{input}^{\mathbf{1}} = \boldsymbol{output}^{\mathbf{1}}$, i.e. the first character of both identifiers is the same.

The presented convention enables to affect the concrete "path" of a token instance within the net. It means in fact that designer of a corresponding net (included within the frame of a simulation model based on the ABAsim architecture) can determine a particular passing of a message instance (carrying specific data) through the net. Such a feature extends behavioural rules of classical coloured Petri nets: instead of disappearing of "old" and appearing of "new" tokens during firing of transitions the tokens representing messages can be understood as preserved.

b) The second character within string identifier (if it has more than one character) of a variable (not a constant) involved in a relevant arc expression is utilized for determining of data that is contained within $\boldsymbol{j}$-th item of a given token $\boldsymbol{c}$. Firing of decision transition $\boldsymbol{t} \in \boldsymbol{T_D}$ follows the next stages:
   - Token instance $\boldsymbol{c}$ from the place $\boldsymbol{p} \in \boldsymbol{P}$: $\boldsymbol{N}(\boldsymbol{a}) = (\boldsymbol{p}, \boldsymbol{t})$, $\boldsymbol{a} \in \boldsymbol{A}$ is removed by means of a variable (with string identifier ***input***) contained within expression of ($\boldsymbol{p}$, $\boldsymbol{t}$).
   - The second character of the ***input*** string is identified as $\boldsymbol{j} = \boldsymbol{int}(\boldsymbol{input}^{\mathbf{2}})$, where function ***int*** transforms character $\boldsymbol{input}^{\mathbf{2}} \in \boldsymbol{M}$ (set of characters '1',…,'9') to a corresponding result from interval of integers $\langle 1, \ldots, 9 \rangle$.
   - The corresponding data content of the $\boldsymbol{j}$-th item of $\boldsymbol{c}$-token is obtained – let us denote it as $\boldsymbol{val}^{c,j}$ – it is consequently elaborated within expressions of decision arcs, which go out of decision transition $\boldsymbol{t}$.

   In case that the second character of the string identifier $\boldsymbol{s}^{\mathbf{2}}$ is equal to '0', it indicates that all data items associated with an instance of relevant token in the simulation program are during the operation represented by the adjacent transition reset to initialisation values.

c) Places are denoted as $\boldsymbol{p_i}$ for $\boldsymbol{i} = 1..\boldsymbol{n}$, where $\boldsymbol{n}=|\boldsymbol{P}|$; $\boldsymbol{p_1}$ denotes *input place* and $\boldsymbol{p_n}$ corresponds to *output place* of the net.

d) The third character within the string identifier $\boldsymbol{s}^{\mathbf{3}}$ of variables is utilized only in the cases that there is more than one identifier in one net with the first two characters being the same, while they are related to tokens of different colours.

e) Naming convention for constant identifiers for their second and every additional character differs from variable identifiers. There is no further rule for that, which means that constant identifiers follow only the convention in the point a).

Let us demonstrate proposed conventions (using the ABA-CPN from fig. 4) for construction of identifiers related to constants and variables on the cases of two transitions.

- For the input decision transition $\boldsymbol{d_1}$, it holds that arc expressions associated with all adjacent arcs dispose of identifiers related to constants and variables, the first

characters of which are equal to 'x' ('$x5$','$x\_No\_transfer$'). It means that the message represented by a token instance removed from the place p_1 is consequently bound to a new token positioned to a relevant place q. The second character of the variable identifier associated with elementary variable arc (p_1, d_1) is equal to '5', i.e. interpreter of ABA-CPN decides about variant branching according to the current content of the 5-th data item of the message encapsulated by a token c that is being currently removed from the place p_1. For example, in the case of $val^{c.5} = REQ\_Deliver\_resource$ the message is bound to a new token that is placed to p_2.

- On the other hand, situation on the transition t_1 is different: firing it does not use only the message instance in the token removed from p_{11} (subsequently placed to p_{15}), but creates another (new) instance as a copy of it and puts it to p_{13}. This behaviour is based on the fact that the first character of the constant identifier '$x\_No\_transfer$' (associated with arc (p_{11}, t_1)) is not equal to the first character of the variable identifier 'y' linked to arc (t_1, p_{13}).

7 Conclusions

In this paper, we described the ABA-CPN, subclass of coloured Petri nets, used for formalization of control and communication of agents within the ABAsim architecture. The ABA-CPN is *structurally bounded*, not *reversible* and not *live* Petri net from definition, since it is acyclic (point (xi) of Def. 1) and there is no source transition allowed (points (x) d) and e) of Def. 1).

State space of the ABA-CPN usually contains at least one dead marking, which can be of two types. The first type is caused by existence of output place p_{out} with no outgoing arc (point (x) b) of Def. 1). In this type of dead marking, only the place p_{out} contains one or more tokens and all the other places are empty. The second type is caused by existence of standard transition $t \in T_B$ or assistant transition $t \in T_A$ with no outgoing arcs, what may lead potentially to a dead marking with no tokens in the net. All dead markings represent admissible end states of processing of initial message (initial marking M_0) in the ABA-CPN. The former type of dead marking symbolises sending of message (result of processing) from current agent to another agent, the latter type stands for consumption of the message form and no need of further communication.

As for liveness of individual transitions, the only transition occurring in all sequences is the input transition $t \in T_D$. If the net is designed correctly, it contains no transition that would be dead in all occurrence sequences for all admissible initial markings. Analysis of the liveness property is the principal benefit from use of the ABA-CPN. It is used for checking of correct construction of a concrete ABA-CPN. If the occurrence graph contains dead markings related to non-admissible states, it indicates a mistake in structure of the constructed net.

Current development concentrated on descriptions of agent components within the ABAsim architecture of simulation models prefers utilizing a specific subclass of coloured Petri net (ABA-CPN) to a subclass of P/T Petri net due to higher modelling capabilities of CPN. For example, construction of conditional branching and

differentiation of various instances of messages is more natural and feasible using formalism of CPN than formalism of P/T PN.

At the present time, the development of a software application (interpreter of ABA-CPN) is carried out. Within the ABAsim simulation kernel, it is supposed to maintain the evolution of the mentioned ABA-CPN. Constructed and analyzed ABA-CPN (within the *CPN Tools* environment) is at interpreter's disposal using XML-formatted file. The interpreter of the ABA-CPN is currently intensively tested.

Acknowledgments. This work has been supported by the Czech National research program under project MSM 0021627505 "Theory of transportation systems" and by the grant of the Scientific Grant Agency VEGA 1/4057/07 in the Slovak Republic.

References

1. Kavička, A., Klima, V., Adamko, N.: Simulations of transportation logistic systems utilizing agent-based architecture, International Journal of Simulation Modelling, DAAAM International, Vienna, 1 (2007) 13-24
2. Adamko, N., Kavička, A., Klima, V.: Agent based simulation of transportation logistic systems, DAAAM International Scientific Book 2007, Chapter 36, B. Katalinic (Ed.), DAAAM International, Vienna (2007) 407- 422
3. Kavička, A.: Petri net with decision transitions applied within ABAsim architecture of simulation models. In MOSIS'03 – Proceedings of the 37th conference Modelling and simulation of systems, MARQ, Ostrava (2006) 373-380
4. Kavička, A., Klima, V., Adamko, N.: Analysis and optimization of railway nodes using simulation techniques, In COMPRAIL 2006 – Proceedings of 10th Computer system design and operation in the railway and other transit system, WIT-Press, Southampton (2006) 663-672
5. Jensen, K.: Coloured Petri nets – basic concepts. Springer Verlag, Berlin (1997)
6. CPN Tools home page. [online]. [cited on 29 February 2008] Available at: <http://www.daimi.au.dk/ CPNTools/>
7. Jennings, R.: An agent-based approach for building complex software systems, *Communications of the ACM*, Vol. 44 (2001) 35-41
8. Helsinger, A., Thome, M., Wright, T.: Cougaar: A Scalable, Distributed Multi-Agent Architecture, *Proceedings of Systems, Man and Cybernetics, IEEE International Conference*, Cambridge (2004) 1910-1917
9. Henoch, J., Ulrich, H.: HIDES: Towards an Agent-Based Simulator, *Proceedings of the Workshop on Agent Based Simulation,* SCS European Publishing House, [cited on 24 September 2008] Available at: *<http://www.ifor.math.ethz.ch/publications/oldpublications/ 2000_towardsagentbasedsimulator.pdf >*
10. Daniel Moldt, D., Wienberg,F.: Multi-Agent-Systems Based on Coloured Petri Nets, Proceedings of the 18th International Conference on Application and Theory of Petri Nets (1997) 82-101
11. Fernandes, J., M., Bello, O.: Modeling of Multi-agent System Activities through Colored Petri Nets: an Industrial Production System Case Study. In Proc. of the16 Int. Conf. on Applied Informatics, Anaheim, CA, (1998) 17-20.
12. Weyns, D., Holvoet, T.: A colored Petri-net for a multi-agent application, Proceedings of MOCA'02 (Moldt, D., ed.), vol 561, DAIMI PB (2002) 121-141