4

Modeling and Analysis Methods for Multi-agent Systems

Jose R. Celaya[1] and Alan A. Desrochers[2]
[1]*Decision Sciences and Engineering Systems Department*
[2]*Electrical, Computer, and Systems Engineering Department*
Rensselaer Polytechnic InstituteTroy, NY 12180
USA

1. Introduction

Multi-agent systems have been studied for the past few decades. At this point in time, several multi-agent systems frameworks have been defined in order to apply the multi-agent system concept to different applications. In a multi-agent system, several agents communicate and interact in order to solve a complex problem. A multi-agent system can be studied as a computer system that is concurrent, asynchronous, stochastic and distributed. These characteristics of multi-agent systems make them also a discrete-event dynamic system, and these have been studied under several analytical methodologies, particularly Petri nets. Petri nets have a well-defined mathematical structure that can be leveraged to provide formal analysis of discrete-event dynamic systems. From the discrete-event dynamic system point of view, multi-agent systems lack analysis and design methodologies. This chapter is concerned with the development of analytical methods for modeling and analysis of multi-agent systems, as well as the definition and assessment of system properties. The study of system properties is becoming more important due to the fact that we are faced more and more with handling large complex dynamic systems. Computer simulation is generally used to assess system properties and to verify that the system is achieving its design objectives. An important challenge in this field is the development of analytical methods to assess key properties of such systems. Such methods could be used to provide a preliminary analysis of the multi-agent system, providing design and operation feedback before the development of expensive simulation models. Furthermore, they will provide insight into design methodologies for multi-agent systems that will ensure that the system design under such methodologies complies with the required properties, hence being a dependable multi-agent system.
The communication and interaction among agents is critical for the overall multi-agent system design and for the proper functioning of the system. Several interaction frameworks have been defined and they range from collaboration among agents, through competition for resources requiring some level of negotiation in the multi-agent system. This work focuses particularly at the interaction level, and studies the interactions between different agents in the system.
The mission of the chapter is to present a methodology for modeling and analysis of multi-agent systems at the agent interaction level. This will be carried out by studying the discrete-

Source: Multiagent Systems, Book edited by: Salman Ahmed and Mohd Noh Karsiti,
ISBN 978-3-902613-51-6, pp. 426, February 2009, I-Tech, Vienna, Austria

event characteristics of the interactions and using the Petri net methodologies as the modeling and analysis tool. Properties known to be important in the discrete event systems and Petri net domains will be used to study multi-agent systems. Here, properties like boundedness and liveness will be analyzed and demonstrated for multiagent systems, as they relate to deadlock avoidance in the Petri net domain. Furthermore, these properties will be related to characteristics of the interaction mechanism. If modeled properly, a deadlock found in a Petri net domain will mean that the interaction mechanism in use in the multi-agent system is prone to deadlocks in the interaction among agents.

In particular, this chapter will include methodologies for mapping multi-agent systems into Petri net models. These methodologies will present the right level of detail/abstraction in order to map all the important behaviors of the interaction framework into the resulting Petri net models. Petri net analysis methods such as the reachability graph and the analysis of the network invariants will be used for the assessment of properties. Furthermore, Petri net synthesis techniques will allow the ability to provide more or less detail to the models, giving the ability to add detail/abstraction to the behavior of the multi-agent system to be modeled.

1.1 Methodology

The problem addressed in this chapter can be considered in two parts: properties, and methodologies for modeling and analysis. The following is the methodology followed for this work.

Properties: If a multi-agent system is regarded as a discrete-event system and modeled using Petri nets, then properties known to be important in the discrete-event systems and Petri net domains could be used to study multi-agent systems. If we consider models of multi-agent systems as discrete-event systems, an important question to consider is which properties in the Petri net domain are important? As a starting point there are properties we would like to analyze and demonstrate for multi-agent systems. Examples of these are boundedness and liveness as related to deadlock avoidance in the Petri net domain. Other properties exist that are related to performance evaluation. These properties from the Petri net domain could be related to characteristics of the communication and interaction of the multi-agent system. If modeled properly, a deadlock found in a Petri net domain will mean that the interaction mechanism in use in the multi-agent system is prone to deadlocks in the interaction among individual agents.

Methodologies for modeling and analysis: Considering that a multi-agent system can be regarded as a discrete-event system, Petri nets can be used as a modeling tool. This will require methodologies for mapping multi-agent systems into Petri net models. These methodologies will require the right level of detail/abstraction in order to map all the important behaviors of the communication and interaction framework into the resulting Petri net models. Having Petri net models of multi-agent systems will allow us to use the existing analysis methodologies for Petri nets. Important properties of discrete-event systems could be obtained with Petri net analysis methods such as the reachability graph and the analysis of the network invariants. New analysis techniques related directly to the multi-agent system domain can be designed or tailored from existing Petri net techniques. Furthermore, Petri net synthesis techniques allow us to provide more or less detail to our models giving the ability to add detail/abstraction to the behavior of the multi-agent system that we would like to model.

1.2 Related work

Petri nets and Petri net extension methodologies have been used to model systems with more than one agent. Murata et al. [1] presented an algorithm to construct predicate/ transition models of robotic operations. Basically, robot actions were considered as firing transitions and the model was used for the planning of concurrent activities of multiple robots (agents). Even though the robotic system considered does not have direct interaction between the agents, the model used shows the ability of the Petri net-like models to capture interactions between the agents that are not evident in the design process. In a similar way, Xu et al. [2] proposed a methodology based on predicate/transition nets for multiple agents under static planning of activities. In addition, they proposed a validation algorithm for plans with parallel activities. The verification is done based on reachability graphs due to the fact that agents actions are modeled as transitions. Petri nets also have been used to model specific multi-agent system frameworks but the resulting models have not been used to provide a study of the properties of the multi-agent system. Ahn et al. [3] proposed a multi-agent system architecture for distributed and collaborative supply-chain management. The suggested architecture is aimed at discovering the structure of the supply-chain and predicting future demands based on local information sharing among the agents. A Petri net model is presented but no structural analysis of the model and no verification of the coordination activities were performed. The advantages of having a Petri net model were not exploited. The work of Leitao et al. [4], proposed a Petri net model approach to formal specification of holonic control systems for manufacturing. They developed a Petri net submodel for each of the four types of holons (agents) suggested in the ADACOR (Adaptive Holonic Control Architecture for Distributed Manufacturing Systems) architecture. There was no attempt to study the structural properties of the Petri net model in order to assess some sort of dependability in the proposed architecture. Formal modeling and specification of the multi-agent systems interaction framework has only been attempted in the holonic manufacturing system considering the contract net protocol as the interaction framework. The work presented by Hsieh in [5] proposed a new model called a *collaborative Petri* net and addressed the question of deadlock and undesirable state avoidance under the contract net protocol. Finally, multi-agent system survivability[1] and fault tolerance using Petri net models have been used in the mobile-agent area. Lyu et al. [6] used a *Stochastic Petri net* model to assess survivability and fault tolerance of mobile agents systems. They use the model for design and verification of their proposed agent architecture.

2. Introduction to Petri nets

Petri nets are a graphical and mathematical modeling tool used to describe and analyze different kinds of real systems. Petri nets were first introduced by Carl Adam Petri in 1962 in Germany [8], and evolved as a suitable tool for the study of systems that are concurrent, asynchronous, distributed, parallel and/or stochastic. Performance evaluation has been a very successful application area of Petri nets. In addition, Petri nets have been successfully used in several areas for the modeling and analysis of distributed-software systems, distributed-database systems, flexible manufacturing systems, concurrent and parallel programs and discrete-event dynamic systems (DEDS) to mention just a few [9][8][10][11]. A

[1] Survivability is the agent's ability to recognize, resist and recover from attacks [7].

multi-agent system is a kind of DEDS that is concurrent, asynchronous, stochastic and distributed. From the DEDS point of view, multi-agent systems lack analysis and design methodologies. Petri net methods are used in this work to develop analytical methodologies for multi-agent systems.

Petri nets are often used in the modeling and analysis of DEDS. They include explicit conditions under which an event can occur; capturing also the relations between concurrent and asynchronous events. As a result, Petri nets are suitable for studying complex and general DEDS [10][11].

This section presents an introduction to Petri nets. Petri nets are defined followed by important properties and analysis methodologies. Finally, an example of a manufacturing application is presented.

2.1 Petri nets definition

Definition 1 *The following is the formal definition of a Petri net [9][8][12][13]. A Petri net is a five-tuple*

$$(P, T, A, W, M_0) \tag{1}$$

where:

P is a finite set of places
T is a finite set of transitions
$A \subseteq (P \times T) \cup (T \times P)$ *is a set of arcs*
$W : A \rightarrow \{1, 2, 3, \ldots\}$ *is a weight function*
$M_0 : P \rightarrow Z_+$ *is the initial marking*

The meanings of *places* and *transitions* in Petri nets depend directly on the modeling approach. When modeling, several interpretations can be assigned to places and transitions. For a DEDS a transition is regarded as an event and the places are interpreted as a condition for an event to occur.

Table 1 presents several typical interpretations for transitions and places. A simple Petri net example is presented in figure 1. This example is used later to define additional Petri net characteristics.

Input Place	Transitions	Output Places
Preconditions	Event	Postconditions
Input data	Computation Step	Output data
Input signal	Signal processor	Output signal
Resources needed	Task or job	Resource released
Conditions	Clause in logic	Conclusion
Buffers	Processor	Buffer

Table 1. Modeling interpretations of transitions and places [8].

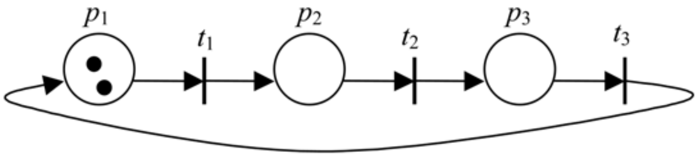

Fig. 1. Petri net example.

Places, transitions and arcs: Places are represented with circles and transitions are represented with bars. The arcs are directed from places to transitions or from transitions to places. The places contain *tokens* that travel through the net depending on the firing of a transition. A place p is said to be an input place to a transition t if an arc is directed from p to t. Similarly, an output place of t is any place in the net with an incoming arc from transition t. In the example (figure 1) p_1 is an input place of t_1, and p_2 is an output place of t_1.
Transition firing: A transition can fire only if it is *enabled*. For a transition t to be enabled, all the input places of t must contain at least one *token*[2]. When a transition is fired, a *token* is removed from each input place, and one *token* is added to each output place. In this way the *tokens* travel through the net depending on the transitions fired.
Definition 2 (Marking) *The marking m_i of a place $p_i \in P$ is a non-negative quantity representing the number of tokens in the place at a given state of the Petri net. The marking of the Petri net is defined as the function $M : P \rightarrow Z_+$ that maps the set of places to the set of non-negative integers. It is also defined as a vector $M_j = (m_1, m_2, \ldots, m_{|P|})$ were $m_i = M(p_i)$, which represents the j_{th}* state *of the net. M_j contains the marking of all the places and the initial marking is denoted by M_0.*
In the example of figure 1 only transition t_1 is enabled. When t_1 fires, one token is removed from place p_1 and one token is added to place p_2. Figure 2 shows the evolution of the Petri net in the previous example. Figure 2 a) presents the initial marking of the net $M_0 = [M(p_1), M(p_2), M(p_3)] = [2, 0, 0]$, only transition t_1 is enabled. Figure 2 b) presents the net with marking $M_1 = [1, 1, 0]$ after t_1 is fired. Here, transitions t_1 and t_2 are enabled and they can be fired. Finally, figure 2 c) represents the net after t_2 is fired. In this case transitions t_1 and t_3 are enabled with marking $M_2 = [1, 0, 1]$.

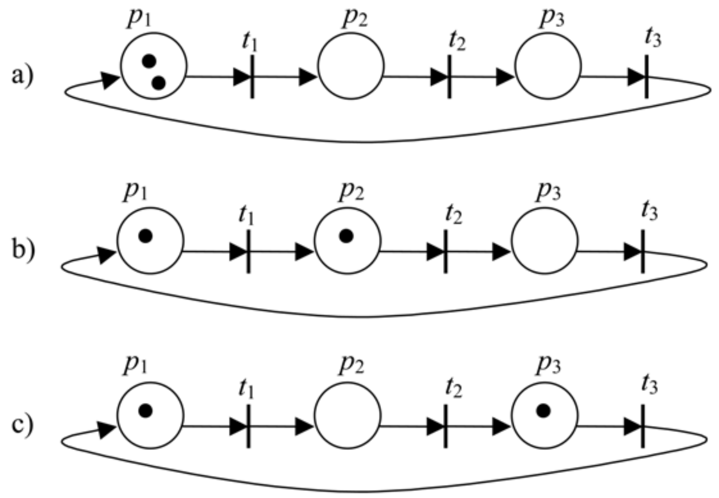

Fig. 2. Petri net evolution after firing transitions t_1 and t_2.

[2] Assuming the weights W of the Petri net are equal to one. When the weights are not indicated they are assumed to be one. The weight on an arc coming to a transition from one of the incoming places indicates the minimum number of tokens needed in the incoming place in order for that transition to be enabled. When the transition fires, it will remove from the incoming place the amount of tokens indicated by the weight of the arc.

The marking of the Petri net represents the state of the net. As described above, the transitions change the state of the Petri net in the same way an event changes the state of a DEDS.

Definition 3 (Reachability graph) *The reachability graph has the marking of the Petri net (or state of the Petri net) as a node. An arc of the graph joining M_i with M_j represents the transition when firing takes the Petri net from the marking (state) M_i to the marking M_j .*

The *reachability graph* of the Petri net in figure 1 is presented in figure 3.

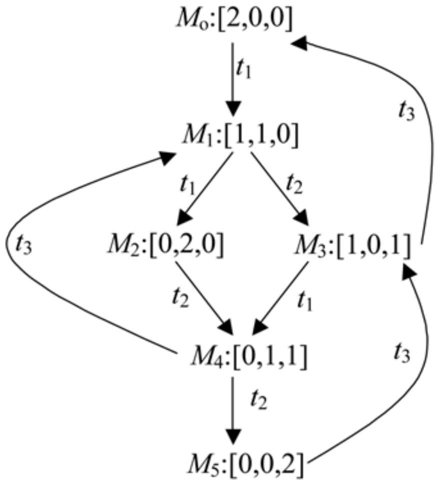

Fig. 3. Reachability graph.

2.2 Properties

This section covers some of the most important *properties* of Petri nets such as *Reachability, Liveness, Boundedness* and *Reversibility*. These properties are essential for the analysis of Petri net models. Furthermore, they are required characteristics for the use of Petri nets in performance evaluation [8][10][11].

These are properties that could be applied to multi-agent systems models. Examples of these properties are boundedness and liveness since they are related to deadlock avoidance in DEDS. Other properties are going to be relevant to multi-agent systems particularly to the communication, interaction, and single agent architectures. It is unknown how the available properties of Petri nets relate to models of multi-agent systems. This is a research question addressed in this study. In addition, the definition of new properties might be required to capture behaviors particular to multi-agent systems. A complete description of the available Petri net properties can be found in [8]. The analysis methods developed in this research will focus on the following properties.

Definition 4 (Reachability) *A marking M_j is said to be reachable from marking M_i if there exists a sequence of transitions that takes the Petri net from state M_i to M_j . The set of all possible markings that are reachable from M_0 is called the reachability set and is defined by $R(M_0)$.*

The concept of *reachability* is essential for the study of the dynamic properties of a Petri net. The reachability set can be obtained from the reachability graph presented previously, e.g., figure 3 [8][11].

Definition 5 (Liveness) *A Petri net is said to be live for a marking M_0 if for any marking in $R(M_0)$ it is possible to fire a transition.*
The *liveness* property guaranties the absence of deadlock in a Petri net. This property can also be observed from the reachability graph. If the reachability graph contains an absorbent state[3], then the Petri net is not live at that state and it is said to have a deadlock [8][11].
Definition 6 (Boundedness) *A Petri net is said to be* bounded *or* k-bounded *if the number of tokens in each place does not exceed a finite number k for any marking in $R(M_0)$. Furthermore, a Petri net is* structurally bounded *if it is bounded for any finite initial marking M_0. A Petri net is said to be safe if it is* 1-bounded *[8].*
Definition 7 (Reversibility) *A Petri net is* reversible, *if for any marking in $R(M_0)$, M_0 is reachable. This means that the Petri net can always return to the initial marking M_0 [8][11].*
For the example in figure 1 we have a reachability set $R(M_0) = \{M_1 = [1, 1, 0], M_2 = [0, 2, 0], M_3 = [1, 0, 1], M_4 = [0, 1, 1], M_5 = [0, 0, 2]\}$. The Petri net is live, reversible and *2-bounded* for the marking $M_0 = [2, 0, 0]$.

2.3 Structural analysis

This section considers the structural analysis of Petri nets by using invariant analysis as described in [8][13]. Basically, the liveness and boundedness of the net will be assessed by using *P-invariants* and *T-invariants*. These invariants are obtained from the incidence matrix of the net and they give information regarding token conservation and transition firing sequences that leave the marking of the net unchanged. These concepts are used to assess the overall liveness and boundedness of the net.
Definition 8 (Incidence matrix) *Let $a_{ij}^{+} = w(i, j)$ be the weight of the arc that goes from transition t_i to place p_j and $a_{ij}^{-} = w(j, i)$ be the weight of the arc from place p_j to transition t_i. The incidence matrix A of a Petri net has $|T|$ number of rows and $|P|$ number of columns. It is defined as $A = [a_{ij}]$ where $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$.*
The example presented in figure 1 shows an ordinary Petri net (all the weights are equal to 1) and the following is its corresponding incidence matrix.

$$A_1 = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \\ 1 & 0 & -1 \end{bmatrix}$$

Definition 9 (Net-invariants) *Let A be the incidence matrix. A P-invariant is a vector that satisfies the equation*

$$Ax = 0 \tag{2}$$

and a T-invariant is a vector that satisfies the equation

$$A^T y = 0. \tag{3}$$

[3] If the net is not live for marking M_0 then at least one marking from $R(M_0)$ will not have any enabled outgoing transitions. If the reachability graph is considered as the state graph of the net, then an absorbent state is that from which the marking it is representing does not have any outgoing transitions enabled. As a result, when the net reaches an absorbent state, it will remain in it indefinitely.

2.3.1 Boundedness assessment

The P-invariants of the incidence matrix are used in Theorem 1 to make an assessment of the boundedness of the Petri net. A Petri net model is covered by P-invariants if and only if, for each place s in the net, there exists a positive P-invariant x such that $x(s) > 0$.

Theorem 1 *A Petri net is structurally bounded if it is covered by P-invariants and the initial marking M_0 is finite [13].*

2.3.2 Liveness assessment

The liveness of the Petri net model is assessed on Theorem 2 by means of the T-invariants of the incidence matrix. A Petri net model is covered by T-invariants if and only if, for each transition t in the net, there exists a positive T-invariant y such that $y(t) > 0$. This is a necessary condition but not sufficient. The liveness assessment by the use of T-invariants is still an open problem [8].

Theorem 2 *A Petri net that is finite is live and bounded if it is covered by T-invariants [13].*

3. Modeling of multi-agent systems with indirect interaction

The methodology presented here consists of defining a simple multi-agent system based on the abstract architecture for intelligent agents (M). The abstract architecture is modeled as a discrete-event system using Petri nets (N) and structural and reachability analysis provides an assessment of the interaction properties. Deadlock avoidance in the multiagent system is considered as a key property, and it is evaluated using the liveness and boundedness properties of the Petri net model.

The purpose of this work consists of the definition of an abstract architecture for multiagent systems with indirect interaction, analogous to the abstract architecture for intelligent agents. The proposed architecture allows the description of agent-to-agent interactions via changes in the environment and serves as an initial description of the discrete-event dynamics of the multi-agent system. In addition, this work presents an algorithm (algorithms 1 and 2) to obtain a Petri net model of a multi-agent system by making use of the multi-agent system's abstract architecture. Finally, a methodology to ensure that the multi-agent system is deadlock free is presented; it is based on the analysis of the properties of the Petri net model.

Here, the abstract architecture of intelligent agents is used as a starting point; particularly the abstract architecture for purely reactive agents. The level of abstraction of agents modeled by the abstract architecture makes it a good candidate for the study of multiagent systems as discrete-event dynamic systems. The study of interaction frameworks is first approached by studying the simplest means of interaction among agents (indirect interaction); assuming the agents have the perception/action capabilities and agents can interact by changing each others environment.

Modeling approach based on interaction among agents: The interactions between agents can be either a direct agent-to-agent interaction or an indirect interaction. The typical structure of a multi-agent system was presented in [14] and is reproduced in figure 4. It shows how the agents interact among each other and how they operate over a metalevel (multi-agent level) environment. Arrows define direct agent interactions from agent to agent; the indirect interactions are based on the environment. In the indirect interaction, an agent modifies another agent's environment triggering a reaction. The indirect interaction occurs in the cases when two or more agents share a subset of the environment. It should be noted that the overall multi-agent system acts over a meta-level environment. An agent that is part of the multi-agent system has its own environment that is somehow related to the meta-level environment of the multi-agent system. This meta-level environment of the multi-agent system is referred to in the literature as being an open environment [15]. A complex problem will provide an open environment, which is dynamic, has components that are unknown in advance, its structure changes over time and might be heterogeneous in its implementation [15]. By focusing on the interactions among agents as described above, it is natural to regard a multi-agent system as a discrete-event system.

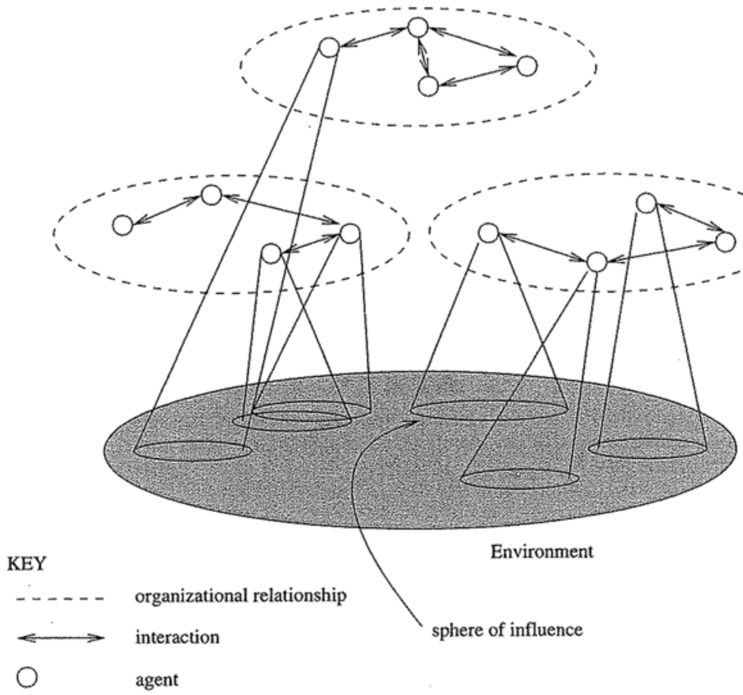

Fig. 4. Structure of a multi-agent system [14].

Following the goal of defining a methodology for modeling multi-agent systems, there is a need to define a modeling methodology that makes no use of the context of the system or at least abstracts itself from it. Basically, looking at an abstract architecture as a means of modeling single agents and multi-agent systems. The proposed modeling approach, uses the abstract architecture to obtain the Petri net models.

3.1 Petri net models from the abstract architecture

The artificial intelligence research considers three difierent paradigms for intelligent agents: a) reactive and b) deliberative; and c) hybrids between them. The abstract architecture models how an agent behaves with respect to changes in its environment. Here, an agent has its own environment and this environment is defined by the nature of the agent. The goals, objectives and the general purpose of the agent define its environment. This abstract architecture is based on the reactive paradigm of perception and action. A purely reactive agent has a perception of the environment and it is used in the decision mechanism that provides an action in the agent. A reactive agent can also have an internal state as a decision mechanism for the actions to be undertaken. An agent with perception and internal state capabilities has more computational power than an agent without them and its computational power is now comparable with that of the Belief-Desired-Intention architecture as described in [14].

Abstract model for purely reactive agents: In a purely reactive agent, the perception part records the changes in the state of the environment. The action part computes the actions to be taken in order to react to changes in the environment. The agent environment changes based on the actions applied by the agent, as well as actions by other agents, and it may be dynamic in that it may change by itself.

The environment consists of a set of states $S = \{s_1, s_2, \ldots\}$. The agent can undertake a set of actions $A = \{a_1, a_2, \ldots\}$ and perceive a set of percepts $P = \{p_1, p_2, \ldots\}$. For a purely reactive agent, the behavior of the agent can be represented as the function *action*: $P \to A$ and *perception* : $S \to P$. The deterministic behavior of an environment can be represented by the function *environment* : $S \times A \to S$.

Petri net modeling of multi-agent systems: A Petri net is defined as a five-tuple (P, T, A, W, M_0) where: P is a finite set of *places*, T is a finite set of *transitions*, $A \subseteq (P \times T)$ [$(T \times P)$ is a set of *arcs*, $W : A \to \{1, 2, 3, \ldots]$ is a *weight* function, and $M_0 : P \to Z_{+}^{|P|}$ is the initial *marking*. When modeling, several interpretations can be assigned to places and transitions. For a discrete-event dynamic system a transition is regarded as an event, and the places are interpreted as a condition for an event to occur.

There are properties we would like to show for multi-agent systems. Examples of these properties are boundedness and liveness since they are related to deadlock avoidance in discrete-event systems. Other properties are going to be relevant to multi-agent systems particularly to the communication, interaction and individual agent architectures. The reachability, liveness and boundedness properties of Petri nets are going to be used in the analysis presented in this chapter.

3.1.1 Obtaining Petri net models from the abstract architecture

Places model the environmental state of the agent. Having a token in a place representing state s_i means that the agent is currently in such a state.

Transitions model the actions of an agent. The environmental state is changed by actions so, for the Petri net model having tokens move from one place to another by firing transitions, this agrees with the execution process of the abstract architecture.

Algorithm 1 (Petri net submodel for agent *i*) *Let S_i be the set of environmental states of agent i, and $s_{ij} \in S_i$ be the jth environmental state of agent i. Similarly, let A_i be the set of actions of agent i, and $a_{ik} \in A_i$ be the kth action of agent i.*

1. *Add a place for each element of the environment S_i and label each place using notation P_{ij} for s_{ij}.*
2. *Add a transition for each action in A_i and label each transition using notation T_{ik} for a_{ik}.*
3. *For each instance of the function environment : $S_i \times A_i \to S_i$ say $s_{ij} \times a_{ik} \to s_{il}$: a) add an arc leaving from place P_{ij} and ending in transition T_{ik}; b) add an arc leaving from transition T_{ik} and ending in place P_{il}; c) add a weight of 1 to each arc.*
   If an arc from transition T_{ik} to place P_{il} already exists, add a new transition and label it T'_{ik}; perform this step using T'_{ik} instead of T_{ik}.
4. *Add a token in the place representing the initial state of the environment.*

Algorithm 2 (Petri net model of the multi-agent system) *The Petri net sub-models of each of the individual agents in the system should be joined based on their indirect interactions. In general, this indirect interaction will be in such a way that an agent i action will change an environment state of agent j. This communication act can be regarded as a regular action in the construction of the complete model. There will be arcs added from the places modeling the environmental states of agent j to the transition modeling the communication in agent i.*

3.1.2 Analysis of the Petri net model

The Petri net model of the multi-agent system can be analyzed to assess system properties like deadlock. Inspection of the reachability graph of the Petri net model can indicate if the model is live and bounded. On the other hand, liveness and boundedness properties can also be assessed using invariant analysis [13]. Basically, the liveness and boundedness of the net can be assessed by using the P-invariants and T-invariants obtained from the incidence matrix which give information regarding token conservation and transition firing sequences that leave the marking of the net unchanged.

Let $a_{ij}^{+} = w(i, j)$ be the weight of the arc that goes from transition t_i to place p_j and $a_{ij}^{-} = w(j, i)$ be the weight of the arc from place p_j to transition t_i. The incidence matrix A of a Petri net has $|T|$ number of rows and $|P|$ number of columns. It is defined as $A = [a_{ij}]$ where $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$. Furthermore, a P-invariant is a vector that satisfies $Ax = 0$ and a T-invariant is a vector that satisfies $A^T y = 0$.

A Petri Net model is covered by P-invariants if and only if for each place s in the net, there exists a positive P-invariant x such that $x(s) > 0$. Furthermore, a Petri net is structurally bounded if it is covered by P-invariants and the initial marking M_0 is finite. In addition, a Petri net model is covered by T-invariants if and only if for each transition t in the net, there exists a positive T-invariant y such that $y(t) > 0$. Furthermore, a Petri Net is live and bounded if it is covered by T-invariants. This is a necessary condition but not sufficient.

3.2 Multi-agent system modeling and analysis example

This section presents a description of a simple multi-agent system consisting of two physical agents that work together at a task. This system will be used throughout this chapter to illustrate how the Petri net and abstract architecture models can be applied to multi-agent systems.

This system consists of two agents referred to as *agent A* and *agent B*. They move objects from one end of a path to the other. Figure 5 shows 4 scenarios in the operation of the system. Both agents are capable of moving objects and agent A moves faster than agent B. The objective of agent A is to go to the end of the path, pick an object, and return to the start of the path (figure 5a). Since it is faster than agent B it reaches the objects first (figure 5b). At the time they intersect in the path, agent A gives its object to agent B and returns to the end of the path to pick another object (figures 5c and 5d). On the other hand, the objective of agent B is to go in the direction of the end of the path, intersect with agent A and get its object. Once the agent has an object, it returns to the start of the path, places the object down and starts all over again.

The *meta-level environment* of the system consists of a single path with a set of objects on one side and no objects on the other side.

There is *indirect interaction* between the agents via changes in their environments. The exchange of the object between the agents should be regarded as a result of a change in the environment of both agents. The two agents are regarded as *purely reactive*, which implies they do not have a record of history and they do not have an internal state. The decisions they make are about which actions to undertake and those decisions are directly influenced by the location of the agent in the path and whether the agent has an object or not. It should be noted also, that the goal of the overall multi-agent system is a little different from the goal of each specific agent.

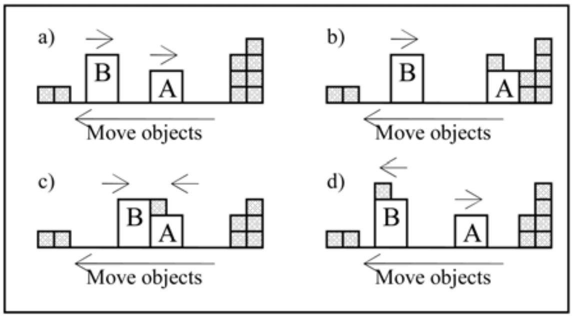

Fig. 5. Multi-agent system example for modeling and analysis.

3.2.1 Abstract architecture description

Under normal conditions, agent A will be walking to the end of the path to pick the object to be moved. The object is going to be taken away by the other agent at the intersection. Let *M* be the multi-agent system with agent A and B modeled with the abstract architecture. Agent A has environmental states S_A, a set of actions A_A, an action function $action_A$, and an environment evolution function $environment_A$. On the other hand, Agent B has environmental states S_B, a set of actions A_B, an action function $action_B$, and an environment evolution function $environment_B$. Tables 2 and 3 describe the possible environmental states (S_A) and the actions (A_A) that agent A can undertake.

State (S_A)	**Description**
s_1	Agent has object
s_2	Agent has no object and it is not at the end
s_3	Agent has no object and it is at the end

Table 2. Environmental states of agent A of *M*.

Actions (A_A)	**Description**
a_1	Walk to the end
a_2	Pick an object
a_3	Walk to start

Table 3. Actions for agent A of *M*.

Table 4 presents the mapping of the environment ($environment_A$) describing how it will be changing as the agent undertakes actions. It should be noted that the notion of exchanging the part with agent B at the intersection has not been considered explicitly in the description of the $environment_A : S \times A \rightarrow S$ for agent A. The agents decision mechanism is described by the $action_A$ function as presented in (4).

$$action_A(s) = \left\{ \begin{array}{ll} a_1 & \text{if } s = s_2 \\ a_2 & \text{if } s = s_3 \\ a_3 & \text{if } s = s_1 \end{array} \right\} \quad (4)$$

Agent B will be walking toward the end of the path until it intersects with agent A which is on its way back to the beginning of the path carrying an object. At the intersection point, agent B takes over the object of agent A and proceeds to return to the beginning of the path to drop the object and start the cycle again. The abstract architecture of agent B is presented in Tables 4 and 5, and the *action*$_B$(*s*) function is described in (5).

A_A	S_A	S_A	Description
a_1	s_2	s_3	Walking towards the end of the path. Eventually will reach the end with no object.
a_2	s_3	s_1	Now the agent has an object.
a_3	s_1	s_1	Walking to the start of the path. Eventually will lose object to agent B.

Table 4. Environment function for agent A of *M*.

State (S_B)	Description
s_1	Agent has no object
s_2	Agent has object and it is not at the start
s_3	Agent has object and it is at the start
s_4	Agent is at the intersection with agent A

Table 5. Environmental states for agent B of *M*.

Actions (A_B)	Description
a_1	Walk to the end
a_2	Take over object from agent A
a_3	Walk to start
a_4	Put object down at start

Table 6. Actions for agent B of *M*.

$$action_B(s) = \left\{ \begin{array}{ll} a_1 & \text{if } s = s_1 \\ a_2 & \text{if } s = s_4 \\ a_3 & \text{if } s = s_2 \\ a_3 & \text{if } s = s_3 \end{array} \right\} \quad (5)$$

The mapping of the environment of agent B (*environment*$_B$) is presented in Table 7. The interaction with agent A (in the exchange of the part) is implicitly modeled by action a_2 although there is no indication in its abstract architecture that it will change the environment of agent A.

3.2.2 Petri net model

The Petri net model of the multi-agent system was obtained following the procedure described in algorithm 1 (*Petri net submodel for an agent*) and algorithm 2 (*Petri net model of the multi-agent system*). Basically, algorithm 1 is executed once for each agent in the system.

A_B	S_B	S_B	**Description**
a_1	s_1	s_4	Will walk toward the end until intersection with agent A
a_2	s_4	s_2	Object exchange, now it has an object
a_3	s_2	s_3	Will walk towards start until it reaches it
a_4	s_3	s_1	Will drop the object at start

Table 7. Environment function for agent B of *M*.

Furthermore, once all the individual agents' submodels are obtained, algorithm 2 is used to join the submodels of those agents that engage in indirect interaction.

Petri net submodel for agent A: The Petri net submodel for agent A presented in figure 6 is obtained as follows. Step 1 of the algorithm is concerned with the places of the Petri net. For each of the environmental states of agent A as described in table 2, a place is added to the new submodel following the described notation. In this step, three places are added in total, e.g., place p_{A2} which models the environmental state s_2. In this state, agent A does not have an object and it is not at the end of the path. The transitions of the model are added in the second step of the algorithm. For this agent, a total of three transitions are added which model the three actions agent A can undertake, e.g., transition t_{A2} models action a_2 which in turn is the *pick an object* action of agent A as described in table 3. The arcs of the Petri net are added in Step 3 of the algorithm. These arcs are related directly to the function that describes the evolution of the environment of the agent. This function maps the Cartesian product of *environmental states* and *actions* into the environmental states resulting from the agent undertaking a particular action. From the Petri net model point of view, this means a Cartesian product of places and transitions that is mapped into a set of places. The addition of arcs revolves around the transitions/actions of each instance of this mapping as described in table 4, e.g., $f(s_2, a_1) = s_3$ indicates that two arcs should be added to transition t_{A1}, an incoming arc from place p_{A2} and an outgoing arc to place p_{A3}. The last step of the algorithm consists of assigning the initial condition or current state to the model. A token is added to the place representing the current environmental state of the agent. A complete list with description of places and transitions is presented in table 8.

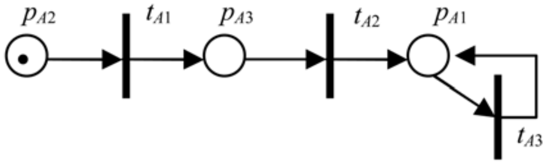

Fig. 6. Petri net model of agent A.

The Petri net submodel for agent A is not pure since place p_{A1} and transition t_{A3} form a self-loop. A Petri net is said to be *pure* if there is no place *p* that is both the input and output place to a transition *t* [8]. This self-loop is an artifact of the abstract architecture model of the agent. A token in place p_{A1} indicates that the agent is currently at environmental state S_1, which in turn means that *agent A has an object*. Furthermore, transition t_{A3} models action a_3 which indicates that *agent A walks to the start of the path*. This self-loop models agent A walking towards the start of the path while carrying an object. It does not model what will happen once the agent reaches the start of the path. It must be noted that this is not a deficiency in the Petri net construction model, but a choice made in generating the abstract architecture description for this agent. This description assumes that when agent A is holding an object and walking towards the start of the path, the other agent will intersect with it and take over the object.

From the Petri net analysis point of view it can be seen that a token will eventually reach place p_{A1} and remain there indefinitely. This is consistent with the abstract architecture description since it does not model the capabilities of the agent once it reaches the start of the path having an object to drop there. As a result, the dynamics of this agent by itself reach a stationary state. A stationary state in this context means that the agent will keep doing the same activity and that the Petri net model indicates that the dynamics eventually get trapped at one state. It can be seeen from the reachability graph in figure 7 that the agent reaches state M_2 and remains there. The only transition enabled at state M_2 is transition t_{A3} and once it fires, the system remains in state M_2.

The states of the reachability graph are described in figure 7 as well. The initial marking M_0 describes the initial condition where agent A has no object and it is not at the end of the path. From the reachability graph it can be concluded that the net is bounded and live for $M_0 = [0, 1, 0]$. Even though the subnet is live, it will remain in state M_2 once such a state is reached, as a result, the subnet is not reversible.

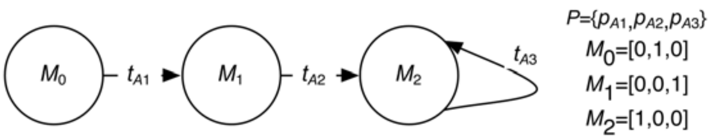

Fig. 7. Reachability graph of agent A.

The incidence matrix A_A of the Petri net submodel of agent A is presented in equation 6. The order of the places in the matrix is $P = \{p_{A1}, p_{A2}, p_{A3}\}$ and the order for transitions is $T = \{t_{A1}, t_{A2}, t_{A3}\}$. The incidence matrix A_A is a $m \times n$ matrix, where $m = |T|$ and $n = |P|$. Let x be a P-invariant of the subnet which satisfies equation $Ax = 0$, $x_1 = [1, 1, 1]^T$ is the only P-invariant of A_A with integer elements. Furthermore, let y be a T-invariant of the subnet which satisfies equation $A^T y = 0$, $y_1 = [0, 0, 1]^T$ is the only T-invariant of A_A with integer elements.

$$A_A = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 1 & -1 & 0 \end{bmatrix} \tag{6}$$

The Petri net submodel of agent A is not covered by positive T-invariants. As a result, the necessary condition for liveness is not met and it can be concluded that the subnet is not structurally live. On the other hand, the subnet is covered by positive P-invariants, as a result, the subnet is bounded for finite initial markings.

Petri net submodel of agent B: The Petri net submodel of agent B is presented in figure 8. This submodel is obtained in a similar way as that of agent A, following the steps presented in algorithm 1 and the abstract architecture description of the agent presented in the previous section. The first step of the algorithm results in the addition of four places to the model based on the set of environmental states described in table 5. The four transitions are added in step 2, representing the set of actions in table 6. The four instances of the environment evolution function presented in table 7 result in the eight arcs of the model. The token was added to place p_{B1} to indicate that the agent does not have an object.

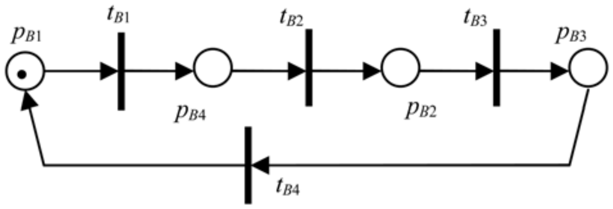

Fig. 8. Petri net model of agent B.

The reachability graph is presented in figure 9. The initial marking M_0 describes the initial condition where agent B has no object. From the reachability graph it can be concluded that the net is bounded and live for $M_0 = [1, 0, 0, 0]$.

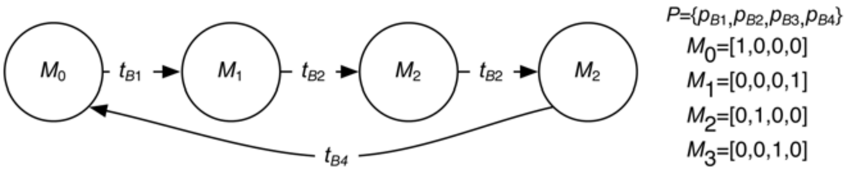

Fig. 9. Reachability graph of agent B.

The incidence matrix A_B of the Petri net submodel of agent B is presented in equation 7. The order of the places in the matrix is $P = \{p_{B1}, p_{B2}, p_{B3}, p_{B4}\}$ and the order for transitions is $T = \{t_{B1}, t_{B2}, t_{B3}, t_{B4}\}$. The incidence matrix A_B is a $m \times n$ matrix, where $m = |T|$ and $n = |P|$. Let x be a P-invariant of the subnet which satisfies equation $Ax = 0$, $x_1 = [1, 1, 1, 1]^T$ is the only P-invariant of A_B with integer elements. Furthermore, let y be a T-invariant of the subnet which satisfies equation $A^T y = 0$, $y_1 = [1, 1, 1, 1]^T$ is the only T-invariant of A_B with integer elements.
The Petri net submodel of agent B is covered by positive T-invariants. As a result, the necessary condition for liveness is met so it can be structurally live. On the other hand, the subnet is also covered by positive P-invariants, as a result, the subnet is bounded for finite initial markings.

$$A_B = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix} \tag{7}$$

Petri net model of the complete multi-agent system: Let $N = (P, T, A, W, M_0)$ be the Petri net model of the complete system with places $P = \{p_{A1}, p_{A2}, p_{A3}, p_{B1}, p_{B2}, p_{B3}, p_{B4}\}$, transitions $T = \{t_{A1}, t_{A2}, t_{A3}, t_{B1}, t_{B2}, t_{B3}, t_{B4}\}$, and $M_0 = [0, 1, 0, 1, 0, 0, 0]$. Figure 10 shows N and the interpretation of places and transitions is presented in Table 8.
This model was obtained by joining the Petri net submodels of the two agents, by following the methodology presented in algorithm 2. The premise of the algorithm is to identify agents engaging in indirect interaction in order to join their models. Furthermore, a transition firing from one of the agents will modify the other agent's environmental state. For this example, transition t_{B2} which models an action of agent B, modifies the environmental state of agent A. The interpretation in terms of the system's description is

that when agent B *takes over the object from agent A*, the environmental state of agent A changes from *having an object* to *not having an object and not being at the end of the path.* As a result, two additional arcs are added to the submodels in order to construct the overall multi-agent system model; an incoming arc to transition t_{B2} from place P_{A1}, and an outgoing arc from transition t_{B2} to place p_{A2}.

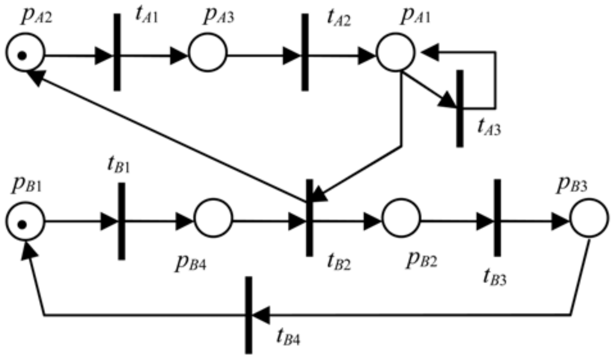

Fig. 10. Petri net model *N*.

Element	**Name**	**Description**
Places (P)	p_{A1}	A token in this place indicates the environment is at state s_1 for agent A
	p_{A2}	A token in this place indicates the environment is at state s_2 for agent A
	p_{A3}	A token in this place indicates the environment is at state s_3 for agent A
	p_{B1}	A token in this place indicates the environment is at state s_1 for agent B
	p_{B2}	A token in this place indicates the environment is at state s_2 for agent B
	p_{B3}	A token in this place indicates the environment is at state s_3 for agent B
	p_{B4}	A token in this place indicates the environment is at state s_4 for agent B
Transitions (T)	t_{A1}	Execution of action a_1 for agent A
	t_{A2}	Execution of action a_2 for agent A
	t_{A3}	Execution of action a_3 for agent A
	t_{B1}	Execution of action a_1 for agent B
	t_{B2}	Execution of action a_2 for agent B
	t_{B3}	Execution of action a_3 for agent B
	t_{B4}	Execution of action a_4 for agent B

Table 8. Description of Petri net model *N*.

The tokens in places p_{A2} and p_{B1} represent the initial conditions of agent A and B respectively. The tokens travel through the net representing different environment states for the agents as the system executes. The systems execution scenarios presented in figure 5 can