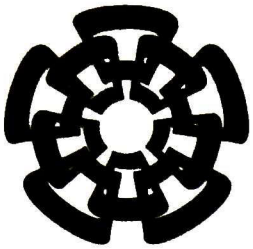


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Modelado y cómputo del comportamiento de agentes interconectados usando redes de Petri híbridas temporizadas

Tesis que presenta:

Gema Berenice Gudiño Mendoza

para obtener el grado de:
Doctora en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis:

Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Agosto 2014.

Modelado y cómputo del comportamiento de agentes interconectados usando redes de Petri híbridas temporizadas

Por:

Gema Berenice Gudiño Mendoza

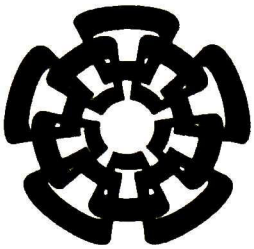
Maestra en ciencias en Ingeniería Eléctrica

CINVESTAV 2005-2007

Director de Tesis:

Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Agosto 2014



Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

Modeling and computing networked agents' behavior using timed hybrid Petri nets

A thesis presented by:

Gema Berenice Gudiño Mendoza

to obtain the degree of:
Doctor in Science

in the subject of:
Ingeniería Eléctrica

Thesis Advisor:

Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, August 2014.

Modeling and computing networked agents' behavior using timed hybrid Petri nets

By:

Gema Berenice Gudiño Mendoza

Master of Science in Electrical Engineering

CINVESTAV 2005-2007

Thesis Advisor:

Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, August 2014

Modelado y cómputo del comportamiento de agentes interconectados usando redes de Petri híbridas temporizadas

RESUMEN

Esta tesis trata sobre el modelado, cómputo y simulación del comportamiento de sistemas de agentes interconectados. Se propone un marco de trabajo para el análisis de sistemas multi-agentes, cuyo enfoque está basado en la representación interna del estado híbrido de cada agente a través de modelos de redes de Petri híbridas temporizadas (RPHT). La propuesta abarca tres etapas importantes: el modelado de agentes y redes de agentes, el cálculo de la evolución del modelo de agente y la simulación de redes de agentes idénticos.

Para el modelado se desarrolló una metodología que permite la estructuración de la red y una arquitectura para el diseño de agentes reactivos multi-rol en la cual pueden distinguirse variables de estado continuas y discretas. Una parte de esta metodología se enfoca en la descripción detallada de la evolución del estado del agente representándolo como RPHT; siendo esta metodología la base para el análisis a través de simulación de la red de agentes.

Respecto a la parte de cómputo, se propone una caracterización matemática la cual permite calcular el comportamiento no lineal de la parte continua de una RPHT mediante programación lineal mixta. La caracterización permite abordar modelos con estructuras de redes cíclicas y con conflictos estructurales; para esta última característica dos políticas pueden ser aplicadas: prioridades y compartimiento.

Posteriormente, se definió una estrategia para simulación del modelo híbrido de una red de agentes idénticos interconectados, la cual permite desarrollar una herramienta de software, donde se explota la característica de agentes idénticos para calcular fuera de línea todos los estados de un agente a través de la contribución antes descrita y posteriormente este cálculo se utiliza en la simulación de la red de agentes evitando así un cómputo repetitivo durante la simulación.

Finalmente, mediante la estrategia de simulación se desarrolló un software para un caso de estudio relacionado con redes de intercambio de energía. El software consiste de dos módulos: uno para el cálculo de los estados posibles de la parte continua de la RPHT implementado en Matlab y el otro para la simulación de la red de agentes, implementado en Java y JADE.

Modeling and computing of networked agent's behavior using timed hybrid Petri nets

ABSTRACT

This thesis deals with modeling, computing and simulation of networked agent systems behavior. A framework for the multi-agent systems analysis is proposed; it is focuses on agent's hybrid state representation through timed hybrid Petri nets (THPN) models. The proposal has three important stages: agent and agent network modeling, calculation of the agent model state evolution, and identical agent network simulation.

For the modeling part, a methodology that allows structuring the network and an architecture for designing reactive multi-role agents has been developed, in which state variables continuous and discrete can be distinguished. One part of this methodology is focused on the description in detail the agent's state evolution using THPN; this methodology is the basis for the analysis through simulation of agents network.

Regarding the computing issue, a mathematical characterization to compute the non linear behavior of the continuous part of a THPN through mixed linear integer programming (MLIP) is proposed. Cyclic structures and structural conflicts can be included in the characterization; on this last feature two policy resolutions can be applied: priorities and sharing.

Thereafter, a scheme for the simulation of hybrid identical agent networked systems has been defined; this scheme allows developing a software tool; where the characteristic of identical agents is exploited by computing off-line all the possible states for only one agent using the contribution described above; then this information is used in the agents network simulation, allowing avoiding the repetitive computing through the simulation.

Finally, through the simulation scheme, a software for a case study related to energy interchange networks has been developed. The software has two main parts: one that computes the possible states for the continuous part of the THPN, which has been implemented in Matlab, and the other one for the agent network simulation, which has been has been implemented in Java and JADE.

ACADEMIC ACKNOWLEDGMENTS

I would like to extend my thanks to Dr. Manuel Silva and Dr. José Angel Bañares, for their invaluable and constructive suggestions on this project during the research staying at the University of Zaragoza, Spain (May-Jul 2012). Their willingness to give their time so generously has been very much appreciated.

Special thanks should be given to Prof. Hassane Alla for his professional guidance, constructive recommendations and invaluable support during the research staying at the University of Grenoble, France (Jun-Jul 2013). Also for his collaboration in the development on the submitted work: "A Linear Characterization of the Switching Dynamic Behavior of Timed Continuous Petri Nets with Structural Conflicts".

I would also like to express my deep gratitude to Dr. Ernesto López Mellado my thesis advisor, for their patient guidance, enthusiastic encouragement, and professional assistance on the complete development of this work.

Finally, I wish to thank Conacyt for its financial support.

AGRADECIMIENTOS

Me gustaría agradecer a mis compañeros del laboratorio de computación por su compañía y consejo.

A mis padres y familia por su apoyo incondicional.

Y a mi esposo por su apoyo recibido tanto profesional como personalmente.

CONTENTS

3.1. Properties of the system

The proposed architecture is a multipurpose agent system, which is a modular representation where the dynamics of each entity of the system is represented by the interaction of continuous and discrete variables [Gudiño-Mendoza,2011]. Some relevant properties represented by the architecture are:

- Closed system; it is not possible to introduce additional agents. The number of agents is fixed.
- The topology is fixed; each agent knows its neighbors all the time.
- The communication between the agents is one-hop nearest neighbors and bidirectional.
- Each agent in the system is an autonomous agent with computation, actuation, sensing, and communication capabilities.
- No agent is controlled by another one.

3.2. General structure of an agent and a networked agent system

An agent A_i can be defined as a 3-tuple $<P_i, EI_i, S_i>$. Where:

- P_i is the vector of perceptions; the messages received from others agents and the signals received from the environment are mapped into this vector, $P_i = [P_{iD} \mid P_{iC}]$, where $P_{iD}(j) \in \mathbb{N}$ and $P_{iC}(j) \in \mathbb{R}$.
- EI_i is the vector of external information; each agent maintains and evolves the information perceived through this vector, this information contains the outside agent's perception, $EI_i = [EI_{iD} \mid EI_{iC}]$, where $EI_{iD}(j) \in \mathbb{N}$ and $EI_{iC}(j) \in \mathbb{R}$.
- S_i is the vector state of the agent; it contains all the basic internal variables to describe the agent, $S_i = [S_{iD} \mid S_{iC}]$, where $S_{iD}(j) \in \mathbb{N}$ and $S_{iC}(j) \in \mathbb{R}$.

The networked agent system can be described as a directed graph $G = \{A, E\}$. Where:

$A = \{A_1, A_2, A_3, \ldots\}$ is the set of agents in the system.

$E \subset A \times A$ defines the topology of the agent network. The pair $(A_i, A_j) \in E$ if the agent A_j can receive information from the agent A_i.

3.3. Agent internal architecture

From an external point of view, the agent processes information coming from sensors and messages sent from other agents; then it produces actions that may affect the environment and sends messages to other networked agents. The scheme of Figure 3.1 describes the main components of an autonomous agent and their relationships. Every module performs a key function in the behavior of the agent.

INTRODUCTION

The multi-agents based approach has been widely held for addressing complex distributed systems. In these kinds of applications the distribution emerges naturally by conceiving the devices as agents that interact with their neighbors in a given topology. According to the applications, the agents have to perform complex individual tasks that involve the information exchange with other agents in order to achieve a collective task.

In general, networked agents are reactive and multirole entities whose reasoning is based mainly on an internal state and the current external information about the environment and from other agents. These agents would be identical but capable of perform different roles. State representation gathers a set of integer and real variables that characterize the status of every cognitive process in the agent.

Several agent architectures have been proposed for representing the agent behavior; they differ according to: their field of application, agents functioning, or the kind of analysis for the networked agent system. Some works considers the agents as intentional systems [Ingrand,1992], [Bratman,1998], [Fisher,1996]; in others works the agents are modeled as abstract machines [Feber,1996], [Yu,2006], [Coleri,2002]; other research group considers the agents as dynamical systems [Bos,1999], [Bosse,2007], [Fregene,2001], [Negenborn,2006]. On the other hand, several analytical approaches have been developed focusing on the analysis and control schemes for abstract machines such as timed hybrid Petri nets (THPN). These works are based on mixed linear integer programming (MLIP) [Balduzzi,1999], [Balduzzi,2000], [Balduzzi,2001], [Dotoli,2008], [Dotoli,2009], [Nishi,2009], [Julvez,2004].

In this thesis we addressed the study of networked agents looking for an analytical framework allowing a more formal analysis of their collective behavior. The proposal includes three contributions regarding the modeling, computing, and simulation of identical interactive networked agents, which exhibit a hybrid dynamics.

The modeling issue is addressed by proposing a modeling framework that includes a) guidelines for structuring the network and b) an architecture for designing reactive agents in which the types of state variables are distinguished [Gudiño-Mendoza,2011]. Later, a methodology for detailing the agents' behavior is developed in which the agent's state evolution is represented by timed Hybrid Petri nets (THPN). At difference of the above mentioned related works, our modeling proposal is focused in establish the agent's state behavior as THPN, where the agent's dynamics is driven by continuous and discrete events, [Gudiño-Mendoza,2013]. The proposed approach is illustrated

through a case study dealing with sensor networks. This was though as a basis for the analysis and simulation of these kind of systems.

The computing issue is addressed by an analytical approach, it is a mathematical characterization based on MLIP of the timed continuous Petri nets (TCPN) models issued from the switching behavior the THPN. This proposal extends previous works by addressing TCPN models including a) a wider set of resolution policies for structural conflicts, and b) cyclic structures [Gudino-Mendoza,2014].

Finally, the simulation issue is addressed through a simulation scheme based on the mathematical characterization for networked identical agents. This scheme allows developing a software tool for executing the hybrid model of a whole agent network. The off-line computation of all possible states for one agent using the characterization for the TCPN is performed; then this information is used to compute the overall networked agent system state's during the network simulation. Unlike the algorithmic simulation proposed in [David,2010], the analytical characterization allows obtaining all the needed information for one agent before the simulation is performed; this information is then used for all the agents avoiding the repetitive computing. The simulation scheme is tested using a distributed energy household network system case study.

This thesis is organized as follows. Chapter 1 introduces the basic concepts of networked agent systems, Petri nets, linear representation for continuous and discrete events, and sensor networks. Chapter 2 describes several works related to the research proposals. Chapter 3 presents a general framework for modeling agent networked systems focusing in the agent's hybrid state representation. Chapter 4 presents a modeling technique based on TCPN for specifying the agent's evolution. Chapter 5 shows TCPN behavior characterization using MLIP. And finally, Chapter 6 presents a scheme for the simulation of identical networked agent system.

Chapter 1

NETWORKED AGENTS, PETRI NETS, LINEAR MODELING, AND SENSOR NETWORKS

Abstract. This section is devoted to cover all the primary notions in which is based the developed work. Section 1.1 presents some basic concepts about networked agents in which the first two modeling proposals presented in Chapter 3 and Chapter 4 are based on. Section 1.2 and Section 1.3 present notions in timed continuous and hybrid Petri nets for proposals from Chapter 4, Chapter 5 and Chapter 6. Section 1.4 describes how to model some discrete and continuous behaviors using linear restrictions with continuous and discrete variables, which are used for proposal in Chapter 5. And the last section presents notions about sensor networks, the two study cases from Chapter 3 and Chapter 4.

1.1. Networked Agents

Agents are a suitable paradigm through which exploit the possibilities presented by massive distributed systems. A collection of these agents seem to be a natural metaphor for understanding and building a wide range of complex systems.

Agents [Negenborn,2004] are problem solvers that have abilities to act, sense, reason, learn, and communicate with each other in order to solve a given problem. Agents have an information set containing their knowledge (including information from sensing and communicating), and an action set containing their skills.

A networked agent system [Wooldrige,2002] consists of a number of agents, which interact with another one, typically by exchanging messages. In the most general case, the agents in a networked agent system will be representing or acting on behalf of users or owners with very different goals.

1.2. Timed continuous Petri nets

Timed continuous Petri nets (TCPN) is a model limit case of time discrete Petri nets. An autonomous continuous Petri net can be defined [David,2010] as 5-uple $R =< P, T, Pre, Post, M_0 >$ such that: P is a set of places, T is a set of transitions, both sets finite and not empty, $P \cap T$ are disjoint, $Pre: P \times T \rightarrow \mathbb{Q}_+$, $Post: P \times T \rightarrow \mathbb{Q}_+$ and M_0 the initial marking. ${}^\circ T_j$ will denote the set of input places to T_j, m_i the marking of place P_i, and $Matrix_{i,j}$ the corresponding value of *Matrix* in row i and column j the same idea applies for vectors.

In a TCPN an additionally relation is added $Spe: T \times V$ where V is the *maximal firing speed* (*mfs*) associated to each transition. The set P of places may be divided into two subsets: $P^+(\mathbf{m}_k)$ the set of places P_i such that $m_i>0$ and $P^0(\mathbf{m}_k)$ the set of places P_i where $m_i=0$. A macro-marking is the union of all markings m_k with the same $P^+(\mathbf{m}_k)$ of marked places.

The evolution of a TCPN is divided is several invariant behavior states (IB-states), each one of them has a duration (dt_k) and a constant vector of *instantaneous firing speeds* (*ifs*) v_k, where $0 \leq v_k \leq V$, and a macro-marking. The evolution marking of these nets is given as for the discrete model by the evolution marking equation: $\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + \boldsymbol{W} \cdot \boldsymbol{v}_k \cdot \Delta t_k$, where $\boldsymbol{W} = \boldsymbol{Post} - \boldsymbol{Pre}$ and $\boldsymbol{m}_{k+1} \geq 0$. A change of IB-State is because a place becomes empty, so the necessity of a nonnegative marking forces to change in vector $\boldsymbol{v}_k$. To compute *ifs* (v_j) for a transition T_j in a TCPN without conflicts three conditions of enabling are necessary to consider:

- Strongly enabled, it means $\forall P_i \in {}^\circ T_j, m_i > 0$ *or* ${}^\circ T_j = \emptyset$
  - $v_j = V_j$
- Not enabled, it means $\exists P_i \in {}^\circ T_j, m_i = 0$ *and* $I_i = 0$, where $I_i = \sum_{T_k \in {}^\circ P_i} \boldsymbol{Post}_{i,k} \cdot v_k$
  - $v_j = 0$
- Weakly enabled, it means $\exists P_i \in {}^\circ T_j, m_i = 0$ *and* $I_i > 0$

- $v_j = \min_{P_i \in {}^\circ T_j} \left(\frac{I_i}{Pre_{i,j}}, V_j \right)$

1.2.1 Conflicts in TCPN

An structural conflict in a TCPN can be defined as $SC = \langle Pc, Tc \rangle$, where $Pc = \{P_i\}$ is one place in conflict, and $Tc = \{T_1, T_2, ..., T_s\} \subseteq {}^\circ Pc$ are the transitions involved in the conflict (Figure 1.1). To establish the resolution by priority, the notation $T_1 < T_2 < ... < T_s$ means that T_1 is the transition with more priority, then T_2 and so on until T_s; the input flow from *Pc* must satisfy first to transition T_1, the remaining flow is applied now to T_2 and so on until T_s. To establish the resolution by sharing the notation $[\alpha_1 T_1, \alpha_2 T_2, ..., \alpha_s T_s]$ is used, this notation indicates that the following relation must be satisfied $v_1/\alpha_1 = v_1/\alpha_2 = ... = v_s/\alpha_s$, where α_I is a real number. An actual conflict occurs when in a structural conflict $m_c = 0$ and the feeding speed I_c is not enough for firing all its output transitions, that is, at least one of these transitions must slow down its *ifs* due to *Pc*; this case is illustrated through an example in section 1.2.3.

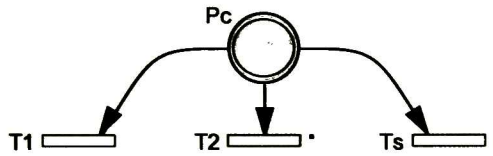


Figure 1.1 TCPN with structural conflict

Below we present some expected behaviors considering different kind of structures and conflict resolution policies; these behaviors all fully covered by the proposal described in Chapter 5.

1.2.2 Expected behaviors in TCPN with conflicts

Table 1.1 shows three possible behaviors of a TCPN with a structural conflict. In each case the policy to solve the conflict is indicated. The TCPN's structure in Figure 1.2 serves for illustrating the three cases.

Cases	*Examples*	$T_4 < T_5 < T_6$	$[T_4, T_5, T_6]$
A. There is not actual conflict	A.1 M_0=[0 m_2 0], V=[2 3 1 2 2 2]	v=[2 3 1 2 2 1]	v=[2 3 1 2 2 1]
	A.2 M_0=[0 m_2 m_3], V=[2 3 1 2 2 2]	v=[2 3 1 2 2 2]	v=[2 3 1 2 2 2]
	A.3 M_0=[0 0 0], V=[2 9 2 3 2 2]	v=[2 9 2 3 2 2]	v=[2 9 2 3 2 2]
B. There is an actual conflict.	B.1 M_0=[0 0 0], V=[2 3 1 2 2 2]	v=[2 3 1 2 1 0]	v=[2 3 1 1 1 1]
C. There is an actual conflict.	C.1 M_0=[0 0 0], V=[0.5 3 1 2 2 2]	v=[0.5 3 1 0.5 2 0.5]	v=[0.5 3 1 0.5 1.25 1.25]
	C.2 M_0=[0 0 0], V=[2 3 1 2 2 0.5]	v=[2 3 1 2 1 0]	v=[2 3 1 1.25 1.25 0.5]
	C.3 M_0=[0 0 0], V=[0.5 3 0.4 2 3 2]	v=[0.5 3 0.4 0.5 2.5 0]	v=[0.5 3 0.4 0.5 2.1 0.4]

Table 1.1 Solutions for cases A-C

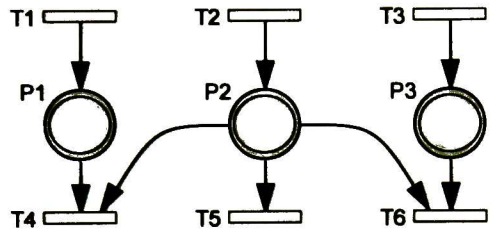


Figure 1.2 TCPN with structural conflict cases A-C

Case A is exhibited when the place in conflict (P_2) is marked (A.1 and A.2) or when the input flow to *Pc* is enough to satisfy the flow for the transitions in the conflict (A.3), for these cases the solution of *ifs* is the same (because the rule does not need to be applied) the *ifs* for the transitions in the conflict is established according to its own *mfs* (A.2 and A.3) or limited by an input flow (A.1), in this case v_6 is limited by I_3=1.

Case B describes a situation in which the resolution rule is in fact fully applied to all the transitions, *Pc* limits the input flow to all transitions in *Tc*; in this case, for all transitions in the conflict, the minimum input flow is given by I_2=v_2=3. In the resolution by priority, T_4 has the highest priority, it goes to its *mfs* 2; then it remains only 1 for T_5 and 0 for T_6. In sharing all the transitions goes to the same instantaneous firing speed 1, the constraint v_4=v_5=v_6 is satisfied.

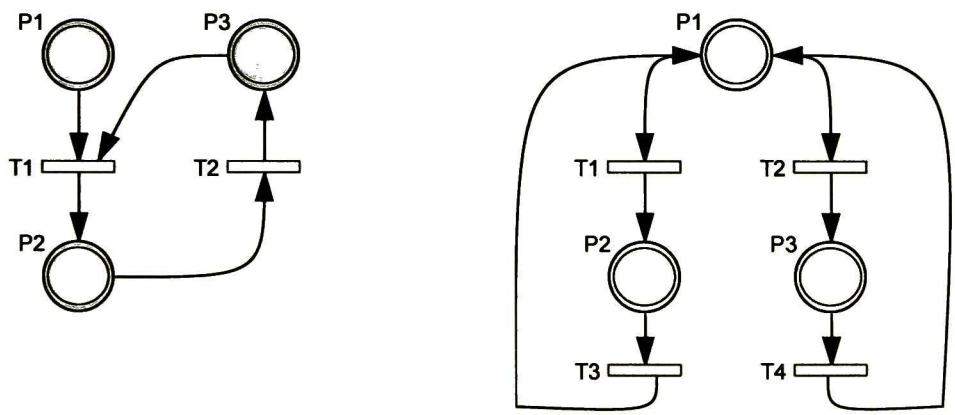
Case C illustrates a problem that arises when some transitions in the conflict are limited by other input flow (C.1, C.3) or by its own *mfs* (C.2), *Pc* limits the input flow to some transitions in *Tc*; this is because, first, the *ifs* for these transitions must be set before, and then the remaining flow must be assigned to the other transitions in the conflict in compliance with the resolution rule. For case C.1 in resolution by priority, the *ifs* for T_4 is limited by I_1; this transition has the highest priority, thus v_4=0.5. For the remaining flow 2.5, v_5=2 is assigned, and then v_6=0.5. In the same case, in the resolution by sharing, I_2=3 must be shared between the three transitions in the conflict, but T_4 cannot consume its part because it is limited by I_1; hence v_4=0.5 and the remaining flow 2.5 must be shared between the other two transitions. The constraint to fulfill in this case must be v_5=v_6; then their *ifs* are 1.25. In case C.2 T_4 is limited by its own *mfs*, so in this case in the resolution by priority establishes v_4=2, v_5=1 and v_6=0. In resolution by sharing, now T_6 cannot use its part because V_6=0.5 then v_6=0.5 and the constraint to satisfy is v_4=v_5; thus both transitions fires at 1.25. Finally in C.3, T_4 is limited by I_1=0.5; in resolution by priority it results in v_4=0.5 and v_5=2.5 and v_6=0, and applying resolution by sharing the flows T_4 and T_6 are limited by the input flows I_1=0.5 and I_3=0.4 respectively, consequently v_4=0.5, v_6=0.4 and the remaining flow from I_2 can be assigned to v_5=2.1, here non restriction must be satisfied.

All the situations described in the above cases may appear during the evolution of a TCPN model, then it is necessary a strategy to manage the application of the conflict resolution policies. Using some discrete and continuous variables it is possible to represent this behavior.

1.2.3. Expected behaviors in TCPN with diverse structures

When the TCPN has circuits or a cyclic structure (Figure 1.3), sometimes there exist some dependencies when *ifs* are calculated. Three cases are presented in Table 1.2 to show the main problems faced with these structures. For Case D (Figure 1.3(a)) there is a dependency: the feeding of T_2 depends on the firing of T_1 and also T_1 needs $v_2>0$ to be fired; in this case T_1 cannot be fired because P_3 never will be fed. This is an important consideration when *ifs* are calculated, in the formulation of the solution must be established that v_2 cannot be fired because P_3 never will be fed; in fact P_1 and P_3 form an unmarked P-component.

In Case E.1 (Figure 1.3(b)), T_3 is strongly enabled, then $v_3=1$; for the resolution policy $T_1<T_2$, first $v_1=1$, then the remaining flow is equal to zero, so $v_2=0$ and $v_4=0$; for the resolution policy $[T_1,T_2]$ the following behavior is observed: the strongly enabled transition T_3 creates an input flow $I_1=1$, this flow is shared between T_1 and T_2, so $v_1=0.5$ and $v_2=0.5$; this causes $v_4=0.5$, now $I_1=1.5$, again the flow is shared and $v_1=0.75$ and $v_2=0.75$, so T_4 increases its flow to 0.75; with this $I_1=1.75$, and $v_1=0.875$ and $v_2=0.875$; this behavior leads to have *ifs* of 1 for all the transitions. In Case E.2, for resolution $T_1<T_2$ again T_3 is strongly enabled so $v_3=2$ and $I_1=2$, this flow allows the firing of T_1 at its *mfs* $v_1=1$ and the remaining flow is assigned to T_2, $v_2=1$ and then $v_4=1$; now $I_1=3$, with this flow both transitions in the conflict can be fired at their maximal speed. For the resolution by sharing at the beginning the input flow $I_1=2$ is shared between T_1 and T_2 leading to $v_1=v_2=1$; it makes $v_4=1$ and now $I_1=3$, with this input flow T_1 and T_2 can fire at the maximal speed. In fact in Case E.2, there is no actual conflict.



(a). Case D. No-conflict (b) Case E. Structural conflict

Figure 1.3. TCPN with structural conflicts cases D and E

Cases	*Examples*	*Solution*	
D. No-conflict	V=[1 3]	v=[0 0]	
E. Structural conflict		$T_1<T_2$	$[T_1,T_2]$
	E.1 M_0=[0 m_2 0], V=[2 2 1 1.5]	v=[1 0 1 0]	v=[1 1 1 1]
	E.2 M_0=[0 m_2 0], V=[1 2 2 1]	v=[1 2 2 1]	v=[1 2 2 1]

Table 1.2. Solutions for cases D and E

1.3. Timed hybrid Petri nets

A marked autonomous hybrid Petri net (HPN) [David,2010] can be defined as a 6-uple $R =< P, T, Pre, Post, M_0, h >$; h: $P \cup T \rightarrow \{D, C\}$, called "hybrid function", indicates for every node whether it is a discrete node (sets P^D and T^D) or a continuous one (sets P^C and T^C); and M_0: $P \rightarrow \mathbf{R}_+$ or $\mathbf{N}$ is the initial marking.

In the definitions of *Pre*, *Post*, and M_0, its domain depends of the kind node of node, $\mathbf{N}$ corresponds to the case where $P_i \in P^D$, and $\mathbf{Q}_+$ or $\mathbf{R}_+$ corresponds to the case where $P_i \in P^C$. *Pre* and *Post* functions must meet the following criterion: if P_i and T_j are such that $P_i \in P^D$ and $T_j \in T^C$, then $Pre(P_i,T_j) = Post(P_i,T_j)$ must be verified. The enabling condition for the proper evolution of the HPN is defined for each kind of transition: A discrete transition in a HPN is enabled if each inning place P_i to T_j has the following condition: $m_i \geq Pre(P_i,T_j)$; a continuous transition in a HPN is enabled if each inning place P_i to T_j has the following condition: $m_i \geq Pre(P_i,T_j)$, if $P_i \in P^D$; and if $P_i \in P^C$, the enabling conditions described in section 1.2 for TCPN are applied.

A timed HPN (THPN) is a pair *(R, tempo)* where:

- *R* is a marked HPN.
- *tempo* is a function from the set *T* of transitions to the set of positive or zero rational numbers
  - if $T_j \in T^D$, $d_j = tempo(T_j)$ = timing associated with T_j;
  - if $T_j \in T^C$, $U_j = 1/tempo(T_j)$ = flow rate associated with T_j.

The fundamental equation to evolve marking of a THPN is the same than that for a TCPN in which vector entries elements $\boldsymbol{m_{k+1}}$, $\boldsymbol{m_k}$ may be positive real or integer numbers, and $\boldsymbol{W}$ may have rational or integer numbers (It depends if elements are continuous or discrete). In a more explicit way the fundamental equation for a THPN can be rewritten as:

$$\begin{bmatrix} m^D_{t_2} \\ m^C_{t_2} \end{bmatrix} = \begin{bmatrix} m^D_{t_1} \\ m^C_{t_1} \end{bmatrix} + \begin{bmatrix} W^D & 0 \\ W^{CD} & W^C \end{bmatrix} \cdot \left(\begin{bmatrix} n^D_{t_2} - n^D_{t_1} \\ 0 \end{bmatrix} + \int_{t_1}^{t_2} \begin{bmatrix} 0 \\ v^C(u) \end{bmatrix} du \right)$$

Where $0 \leq t_1 \leq t_2$, and n^D_t is the vector whose components correspond to the number of firings of each discrete transition from time *0* to time *t*, and $v^c(u)$ is the vector whose components correspond to the instantaneous firing speed for continuous transitions.

For the calculus of the instantaneous firing speed, first it is necessary to obtain the maximal firing speed (V_j). It is the product of the flow rate by the *D-enabling degree*: $V_j = U_j\, D(T_j, m)$, where

$$D(T_j, m) = \min_{P_i \in {}^\circ T_j \cap P^D} \left\lfloor \frac{m_i}{Pre(P_i, T_j)} \right\rfloor$$

1.4. Linear constraints to represent continuous and discrete events

When we analyze the behavior of a system, the objective is to find the simplest model; for this purpose linear models are often used because they are easier to solve. However, many systems can exhibit complex behaviors and the use of real variables is not enough for representing such behaviors; in this case expressions including binary variables allow representing non linear situation as linear, At a first look, this artifice makes appear the problem as non linear; but there exist powerful algorithms based in linear programming techniques to solve this kind of problems.

In order to compute the behavior of a TCPN, it is necessary to calculate the *ifs*; these velocities are constant for periods of time, but they may change abruptly due to changes in the marking of places; these jumps make the system non linear. Additionally, other important issue in TCPN is computing *ifs* for transitions involved in a structural conflict. Every conflict has a policy resolution (sharing or priorities), then solving the conflict requires including in the model some conditional expressions (if-then).

The basic non-linear statements used in this work, transformed to linear constraints are listed below. For more details the reader can consult [William,1999], [Glover,1975], [Floudas,2004] or [Bemporad,1999].

Nonlinear	***Is equivalent to***
$\lambda_1 V \lambda_2$	(1) $\lambda_1 + \lambda_2 \geq 1$
$y_1 = y_2 \cdot y_3$	(2) $y_3 = \sum_{i=0}^{\lceil log_2(Max(y_2)) \rceil} 2^i \cdot \lambda_i$
$y_1 = \lambda_1 \cdot y_2$	(3) $y_1 \leq Max(y_2) \cdot \lambda_1$
	(4) $y_1 \geq Min(y_2) \cdot \lambda_1$
	(5) $y_1 \leq y_2 - Min(y_2) \cdot (1 - \lambda_1)$
	(6) $y_1 \geq y_2 - Max(y_2) \cdot (1 - \lambda_1)$
$[y_1 = x_1] V [y_1 = x_2]$ ... $V [y_1 = x_n]$	(7) $y_1 = \sum_{i=1}^{n} x_i \cdot \lambda_i$

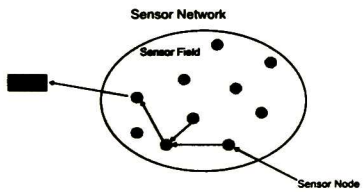
	(8) $\sum_{i=1}^{n} \lambda_i = 1$
$[y_1 \leq 0] \rightarrow [\lambda_1 = 1]$	(9) $y_1 \geq \varepsilon + (Min(y_1) - \varepsilon) \cdot \lambda_1$
Where	$Min(y_i) \leq y_i \leq Max(y_i)$ $y_i, x_i \in \mathbb{R}$ $\lambda_i \epsilon \{0,1\}$

Table 1.3. Logic statements expressed in linear constraints

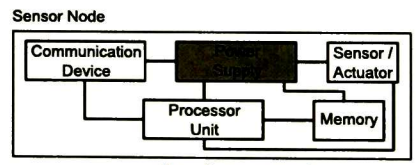
1.5 Sensor networks

A generic model [Anastasi,2009] for Sensor Networks (Figure 1.4(a)) consists of sensing devices distributed in a wide area, which are interconnected and coordinate together for transmitting sensor data to a sink node (or base station) that will produce meaningful information. Sensor networks are usually composed of a large number of wireless identical sensors; this characteristic facilitates the design, analysis, and operation of this kind of systems. A basic sensor node (Figure 1.4(b)) is composed by five main components [Holger,2005]:

- A processor unit to process all the relevant data.
- Some memory to store programs and intermediate data.
- Sensors and actuators, which are the actual interface to the physical world: devices that can observe or control physical parameters of the environment.
- A communication device for sending and receiving information over a wireless channel. Usually these modes operate in a mutual exclusion way.
- Usually no tethered power supply is available; some form of batteries is necessary to provide energy. Sometimes, recharging capabilities allows obtaining energy from the environment (e.g. solar cells).



(a) Sensor network



(b) Sensor node

Figure 1.4. Sensors

Each of these components has to operate balancing the trade-off between the reduction of energy consumption as possible and the need to achieve their tasks. Energy consumption of a sensor node must be tightly controlled. The battery has small capacity and recharging by energy scavenging is complicated and volatile.

The crucial observation for proper operation is that most of the time a wireless sensor node has nothing to do. Hence, it is best to turn it off. Therefore, completely turning off a node is not possible, but rather, its operational state can be adapted to the tasks at hand. Introducing and using multiple states of operation with reduced energy consumption in return for reduced functionality is the core technique for energy-efficient wireless sensor node.

The main task of a sensor node is to sense information and transmit it to a sink node. In order to accomplish this task, one solution possible can be for each node to transmit its data directly to the sink node. However, if the network is deployed in a wide region, this one hop transmission is costly and nodes die very quickly. Another approach is to use multi-hop forwarding. In the common multi-hop configuration, sensor nodes form a routing tree, each node forwards data of its own and its children to its parent in the tree.

Chapter 2

AGENT ARCHITECTURES AND BEHAVIORAL ANALYSIS OF PETRI NETS

Abstract. In this chapter is presented the background and the motivation for the research addressed in this thesis. First some important agent-based system architectures in Section 2.1 is revised; this works correspond to the related work for proposals from Chapter 3 and Chapter 4. Section 1.2 mentions the most important works related to the proposal presented in Chapter 5 and an algorithmic approach for the simulation of TCPN is the reference for Chapter 6.

2.1. Agent-Based architectures

There are several proposals for agent-base systems architectures. Three approaches are addressed: Agents as intentional systems, agents as abstract machines and agents as dynamical systems. All of them proposes a way to model internally an agent and how interact with the environment and other agents.

2.1.1. Agents as intentional systems

In this approach, agents are endowed with: beliefs, desires, and intentions (BDI). Its behavior is predicted and explained through these mental states. Three of the most important works in this are described in the following lines.

[Ingrand,1992] presents a generic architecture: the Procedural Reasoning System (PRS). It consists of a database containing the systems current beliefs about the world, a set of current goals, a library of plans or procedures (Sequences of actions) and an intention structure (Plans chosen for execution at run time).

In proposal [Bratman,1998] a high-level specification of the practical-reasoning component for a resource-bounded rational agent is presented. This architecture has four key symbolic data structures: a plan library, and explicit representations of beliefs, desires, and intentions. Additionally, the architecture has: a reasoner, for reasoning about the world; a means-ends analyzer, for determining which plans might be used to achieve the agent's intentions; an opportunity analyzer, which monitors the environment in order to determine further options for the agent; a filtering process; and a deliberation process.

The proposal [Fisher,1996] called INTERRAP, combines the BDI style and a layered architecture. It models the agent's ability to interact, react, deliver and cooperate (With other agents). INTERRAP describes an agent by its world's interface, control unit and a knowledge base.

2.1.2. Agents as abstract machines

In proposal [Feber,1996] a general theory of action in networked agent systems not based on agent's mental states is presented. It relies on a clear distinction between influences, which are produced by agents' behavior, and the reaction of the environment. This theory describes complex interactions between situated agents. It provides specific tuples and functions to describe and evolve the internal state of an agent and a networked agent system.

In work [Yu,2006] an ADL (Architecture Description Language) for a multi-agent system (ADLMAS) is proposed. It adopts Object-Oriented Petri nets (OPN) as a formal theory basis. ADLMAS can visually and intuitively depict a formal framework for networked agent systems from the agent level and society level, describe the static and dynamic semantics, and analyze, simulate

and validate networked agent system and interactions among agents with formal methods. It studies a networked agent system from the point of view of software architecture.

In [Coleri,2002] a modelling methodology for a sensor network is proposed. They present how to model TinyOS (Event-based operating system for sensors) through a hybrid automata. Once an agent is modeled, some properties like the reachability problem can be verified using HyTech (a tool for the automated analysis of embedded systems). A simulation of a sensor network is provided and an energy conservation analysis over this example.

2.1.3. Agents As dynamical systems

In proposal [Bos,1999] a networked agent system is seen as a collection of distributed control system or decision makers operating on a dynamic system. A visual scheme and generic equations are proposed to describe the internal agent state and its communication with other agents.

In [Bosse,2007] is presented and integrative approach to simulate and analyze complex systems integrating quantitative, numerical and qualitative logical aspects within one expressive temporal specification language: Leadsto. Also, it proposes techniques for analysis of hybrid systems.

HICA [Fregene,2001] combines concepts from hybrid control and networked agent systems to build agents which are especially suitable for multimode control purposes, applicable to diverse domains. The HICA agent essentially consist of an intelligent agent built around a hybrid control system core which is intended to provide a 'bare-bones' structure to facilitate controller synthesis. The HICA agent combines deliberative planning/coordination with reactivity.

In [Negenborn,2006] a networked agent model predictive control for transportation networks like road traffic networks, power distribution networks or water distribution networks is presented. These transportation networks are considered at a generic level, at which commodity is brought into the network at sources, flows over links to sinks, and is influenced in its way of flowing by elements inside the network. This work considers both continuous and discrete elements, as control goals includes avoiding congestion of links, minimizing costs of control actions, maximizing throughput, etc.

2.1.4. Discussion and proposals

The most part of these works are generic tools to be used in a wide range of applications [Ingrand,1992], [Bratman,1998], [Fisher,1996], [Feber,1996], [Yu,2006], [Bos,1999], [Bosse,2007] and [Fregene,2001]. They can be seen as descriptive ways to formally specify an agent's internal state and their interrelationships with other agents and its environment. Modeling is their primary objective, some of them go beyond and present simulations through an example.

The first proposal of this thesis (Chapter 4) takes inspiration of the works [Ingrand,1992], [Bratman,1998] and [Fisher,1996] in the sense of present in a modular way, the internal structure of an agent. The modeling framework proposed differs from these works in the representation of the evolution of an agent; it is not based in logic aspects as in them, its evolution is established through generic functions to express continuous and discrete events.

Trying to find a way to specify formally these generic functions (second proposal Chapter 4), several works regarding to represent an agent with continuous and discrete dynamics where study: [Feber,1996], [Yu,2006], [Bos,1999], [Bosse,2007], [Fregene,2001], [Coleri, 2002] and [Negenborn,2006].

Formalism like [Feber,1996] introduce its own abstract machines, and [Yu,2006] uses the formalism of Petri nets. The advantages of use a proprietary formalism is that they can be more suitable to the problem, and tools like Petri nets are necessary to extend and adapt; but some previews studies in these nets, allows further analysis and simulation.

Works like [Bos,1999], [Bosse,2007] and [Fregene,2001] allows to specify an agent as a set of variables evolving through differential equations, in the same way as the formalism of Petri nets there is interesting theory to introduce analysis techniques in the networked agent systems. Works like [Coleri,2002] and [Negenborn,2006] are completely focused in address a specific problem; the first one analyzes energy consumption and the other one controls the flow between agents. In particular [Negenborn,2006] it's an interesting work; here the hybrid dynamic of each agent is modeled with mixed linear integer programming. The behavior of a THPN, formalism taken for the second proposal; it is governed by discrete and continuous events and calculate the evolution of these events is not an easy task. The third proposal starts here, as a way to characterize this net's behavior. The developed work has been in TCPN, the continuous part of the net. Focused in this line, the following section presents several works regarding this.

2.2 Algorithms and mathematical formulations to describe TCPN's behavior

2.2.1 Speed calculator algorithms

In [David,2010] five algorithms of speed calculation for one IB-state are presented, which serve as an algorithmic tool for TCPN simulation. The calculation of the *ifs* must be performed by solving a linear programming problem (LPP) in each IB-state. The set of constraints for the LPP depends on if the net has or has not conflicts.

The constraints in the LPP are the following:

- C1. Speed limits. *mfs*

- C2. Non-negative balances. For empty places their marking must not be negative, so the output flow must be equal or less than the input flow.
- C3. Transitions whose instantaneous speed is zero and surely firable transitions. Some weakly enabled transitions may not be fired because all its input places could not receive flow, so the firing velocity for these transitions must be set to zero. In nets with structural conflicts, in order to obtain this information it is required to solve several times a LPP. This restriction is needed as mentioned before in Section 1.2.3, Case E.1.

Solving the LPP implies maximize $\sum_{T_j \in T} v_j$ given C1,C2 and C3.

If the net has not conflicts, then the set of constraints is obtained with Algorithm 1 (See below). The following four algorithms require several passages (several computations of a LPP) in order to obtain the *ifs*. All the algorithms receive as input data the incidence matrix, the *mfs*, and the marking at the beginning of the IB-state. For TCPN with conflicts is necessary to provide the conflict resolution rule. The output of each algorithm is a vector containing the *ifs*.

The general characteristics are summarized in Algorithms 1-5. Also an additional iterative algorithm is used to obtain the information for all IB-states. At the beginning, this algorithm divide the complete net according to some rules and the *ifs* of each subnet is calculated by one of the 5 algorithms according to its structure and conflict resolution if the subnet requires it.

Algorithm 1: Finding *ifs* in TCPN without structural conflicts

Input: W, *mfs*, M_0

Output: ifs

1. Determine C3 for not firable transitions according to the flow propagation through the net according to the current marking.
2. Solve the LPP.

Algorithm 2: Finding *ifs* in TCPN with structural conflicts solved by priorities

Input: W, *mfs*, M_0, conflict resolution

Output: ifs

1. Determine C3 for not firable transitions in several passages based on the priority resolution rule established.
2. In each passage solve a LPP to determine if a transition with less priority can be fired in the next passage.

Algorithm 3: Finding *ifs* in TCPN with one structural conflict of two transitions solved by sharing

Input: W, *mfs*, M_0, conflict resolution

Output: ifs

1. In a sharing between two transitions T_1 and T_2, speed vectors are calculated successively assuming that $T_1<T_2$ and later assuming that $T_2<T_1$.

Algorithm 4: Finding *ifs* in TCPN with structural conflicts and without a circuit solved by sharing

Input: W, *mfs*, M_0, conflict resolution

Output: ifs

1. With this kind of structure it is possible to calculate the speeds dividing the net in three subnets: transitions and place in conflict, the upstream subnet from the transitions in the conflict and the downstream subnet from the transitions in the conflict.
2. First, the *ifs* for the upstream subnet is calculated, then the *ifs* for the transitions in the conflict, and finally the *ifs* for the transitions in downstream net. For the upstream and downstream subnets any previous algorithm can be used. And for the transitions in the conflict a procedure iterative is given to share the input flow between the transitions in the conflict.

Algorithm 5: Finding *ifs* in TCPN with structural conflicts solved by sharing and priorities

Input: W, *mfs*, M_0, conflict resolution

Output: ifs

1. Each net is divided into groups (subnets). The *ifs* of each group are calculated separately using any of the previous algorithms.

2.2.2. First order hybrid Petri nets

First-Order Hybrid Petri Net (FOHPN) is a formalism very similar to Timed Hybrid Petri Nets in which the continuous part dynamics is piecewise constant like in TCPN. However it has two slightly differences: a) it is used mainly to solve some optimization problems, by allowing the transitions to take values between an interval to compute the *ifs*. b) the continuous part of the net is only disabled by the discrete part.

2.2.2.1. Controlling FOHPN

In [Dotoli,2008] and [Balduzzi,2000] several linear programming models are proposed to control the continuous part of the FOHPN when it has a fixed discrete marking (Operational mode). The authors provide a technique to maximize flows in the continuous transitions and minimize stored flows in the places; several examples for the manufacturing area are presented. In this work the solution found is a piecewise optimal solution. When there are conflicts in the net a more general schema than the resolution established in [David,2010] is proposed; furthermore the optimization function itself gives the policy used en each IB-state.

2.2.2.2. FOHPN Decomposition technique

In [Nishi,2009] a general decomposition method for transition firing sequence problems in FOHPN is proposed. The dynamics of the net is formulated as a MLIP problem, and this problem is solved

with a decomposition method. Regarding the modelling there are two interesting features: a) the complete behavior is represented through linear equations, and b) the time in the model evolves in fixed discrete time intervals; usually a small value is chosen. At the end of this stage the obtained results, the marking evolution and the firing speeds, are numerous, and an additional procedure is necessary to identify each IB-state. This MLIP is used to optimize time to reach a desired marking.

2.2.3 TCPN Event-Driven control

In [Julvez,2004] a strategy to control a TCPN as an event-driven mixed logical dynamical system is proposed. The main difference with the preview work (besides of the kind of nets) is that the time evolution is driven by the occurrence of events in the TCPN; this reduces the number of steps to reach an optimal sequence. Also this work presents several optimization criterions, for example time to reach a desired marking and throughput maximization. When there are structural conflicts ($<\{P_c\},\{T_1,T_2\}>$) in the net the resolution policy is equal sharing($[T_1,T_2]$). For this case an example with two transitions is presented.

2.2.4 Discussion and proposal

In Table 2.1 a brief summary of the works described before as well as our proposal is presented.

Related Work	***Objective***	***Solution***	***Optimization criteria***	***Time Modeling***	***Class of net***	***Policy Resolution***
[David,2010]	Compute *ifs*, marking, IB-state duration	LPP			TCPN	Priorities (Any structure) Sharing (2 transitions any structure, 2 or more transitions with a structure without a circuit)
[Dotoli,2008]	Optimization in each IB-state.	LPP	Optimal time, throughput, maximize flows		FOHPN	Ruled by the optimization function
[Nishi,2009]	Global Optimization	MLIP	Optimal time	Discrete-Time	FOHPN	Ruled by the optimization function
[Julvez,2004]	Global Optimization	MLIP	Optimal time, throughput, steady state	Continuous-Time	TCPN	Equal sharing
Proposal	Compute *ifs*, marking, IB-state duration	MLIP		Continuous-Time	TCPN	Priorities (Simple Petri nets) Sharing (Simple Petri nets)

Table 2.1. Comparative: Related work and proposal

Most of the related works do not include a fixed policy resolution when there is a structural conflict, with the exception of [Julvez,2004] where the considered policy is equal sharing for two transitions

in conflict; however the situations in which there exist more than two transitions, such as the case C analyzed in Section 1.2.2, are not addressed.

The proposal described in Chapter 5 presents a method for computing firing speed, the marking and the IB-state duration in TCPN. It addresses a wider class of conflict resolution policies than those handled in related works. Structurally these petri nets can be simple petri nets, in which each transition may be involved in at most one conflict. Both sharing and priorities policies involving any number of transitions are dealt. Although this proposal has common features with works described above, such as the use of MLIP and time management, the proposed method copes with more general structures that include conflict resolution policies. Furthermore, the method allows the systematic statement of the MILP.

Chapter 3

MODELING FRAMEWORK FOR NETWORKED AGENT SYSTEMS

Abstract. In this chapter a general architecture [Gudiño-Mendoza,2011] for a subclass of networked agent systems is presented. First, properties of the modeled system are presented. Then the general structure of an agent is defined as well as the structure of the multi-agent system. Furthermore the agent internal architecture and its internal state representation are presented. Later communication between agents is described. And finally, a case study illustrates the proposed schemes.

3.1. Properties of the system

The proposed architecture is a multipurpose agent system, which is a modular representation where the dynamics of each entity of the system is represented by the interaction of continuous and discrete variables [Gudiño-Mendoza,2011]. Some relevant properties represented by the architecture are:

- Closed system; it is not possible to introduce additional agents. The number of agents is fixed.
- The topology is fixed; each agent knows its neighbors all the time.
- The communication between the agents is one-hop nearest neighbors and bidirectional.
- Each agent in the system is an autonomous agent with computation, actuation, sensing, and communication capabilities.
- No agent is controlled by another one.

3.2. General structure of an agent and a networked agent system

An agent A_i can be defined as a 3-tuple $<P_i, EI_i, S_i>$. Where:

- P_i is the vector of perceptions; the messages received from others agents and the signals received from the environment are mapped into this vector, $P_i = [P_{iD} \mid P_{iC}]$, where $P_{iD}(j) \in \mathbb{N}$ and $P_{iC}(j) \in \mathbb{R}$.
- EI_i is the vector of external information; each agent maintains and evolves the information perceived through this vector, this information contains the outside agent's perception, $EI_i = [EI_{iD} \mid EI_{iC}]$, where $EI_{iD}(j) \in \mathbb{N}$ and $EI_{iC}(j) \in \mathbb{R}$.
- S_i is the vector state of the agent; it contains all the basic internal variables to describe the agent, $S_i = [S_{iD} \mid S_{iC}]$, where $S_{iD}(j) \in \mathbb{N}$ and $S_{iC}(j) \in \mathbb{R}$.

The networked agent system can be described as a directed graph $G = \{A, E\}$. Where:

$A = \{A_1, A_2, A_3, \ldots\}$ is the set of agents in the system.

$E \subset A \times A$ defines the topology of the agent network. The pair $(A_i, A_j) \in E$ if the agent A_j can receive information from the agent A_i.

3.3. Agent internal architecture

From an external point of view, the agent processes information coming from sensors and messages sent from other agents; then it produces actions that may affect the environment and sends messages to other networked agents. The scheme of Figure 3.1 describes the main components of an autonomous agent and their relationships. Every module performs a key function in the behavior of the agent.

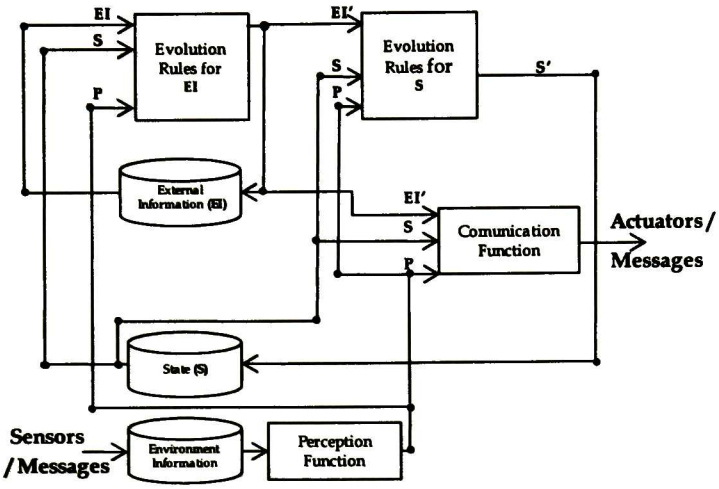


Figure 3.1. Agent Architecture

Environment Information. This repository stores the information received from other agents or from sensors. The perceived information is local; the agent cannot know the overall system state.

Perception Function. This module maps the information stored in *Environment Information* into a hybrid vector *P*.

Evolution Rules for EI (External Information). This module contains a set of rules to update the vision that the agent has from the outside.

Evolution Rules for S (Agent's state). This module performs the evolution of the agent's state *S*. The state is influenced by the hybrid vectors *EI'*, *P*, and the current state *S*. These rules and the *Evolution Rules for EI* are fixed; they do not change during the evolution of the system.

Communication Function. This function performs pertinent actions on the environment and/or sends messages to other agents according to values of the hybrid vectors *IE'*, *S*, and *P*.

The interaction between the modules is performed as follows. An agent can receive information from outside through sensors or messages. This information is stored in the repository *Environment Information*. Through *Perception Function*, the received perceptions are mapped in a vector named *P*, allowing translating the received information into continuous and discrete variables.

The module Evolution Rules for *EI*, takes as input current values in *EI*, *S* and *P* to evolve the vector *EI* to *EI'*. According to the new values in *EI'* and the current values of *S*, the state evolves to *S'* through *Evolution Rules* for *S*. In this new state, actions and messages can be sent to the outside though the *Communication Function* module Agent internal state representation

State representation and updating is an important feature for the agent operation exhibiting intelligent behavior, since most of the decision making functions are based on agent's state.

In general, the state evolves according to the dynamics of the diverse internal behaviors represented for achieving the assigned tasks. It can be viewed as a state equation in which a complex function $f = [f_D f_{DC} f_{CD} f_C]$ computes the next state:

$$S' = \begin{bmatrix} S_D' \\ S_{DC}' \\ S_{CD}' \\ S_C' \end{bmatrix} = \begin{bmatrix} f_D(EI_D, S_D, P_D) \\ f_{DC}(EI, S, P) \\ f_{CD}(EI, S, P) \\ f_C(EI_C, S_C, P_C) \end{bmatrix}$$

f_D : Represents the pure discrete dynamic of the agent. $F_D : EI_D \times S_D \times P_D \rightarrow S_D$. $S_D(j) \in \mathbb{N}$.

F_{DC}: Represents the discrete dynamic of the agent influenced by continuous and discrete variables. $F_{DC} : EI \times S \times P \rightarrow S_{DC}$. $S_{DC}(j) \in \mathbb{N}$.

F_{CD}: Represents the continuous dynamic of the agent influenced by continuous and discrete variables. $F_{CD}: EI \times S \times P \rightarrow S_{CD}$. $S_{CD}(j) \in \mathbb{R}$.

F_C: Represents the pure continuous dynamic of the agent. $F_C : EI_C \times S_C \times P_C \rightarrow S_C$. $S_C(j) \in \mathbb{R}$.

Figure 3.2 presents in more detail the module *Evolution Rules for S*, showed in Figure 3.1. In this module the agent state is evolved. *EI'*, *S* and *P* are the inputs to the module. Each vector is decomposed in its continuous and discrete parts. All the discrete parts (V_D) are evaluated in the function f_D to produce the discrete part of S' (S_D'). *Pre* is a verification vector of the discrete elements and *Post* is an output vector of the discrete elements. With the function f_{DC} evolves all the discrete states S_{DC}, influenced by the continuous and discrete variables (V_C, V_D). *Pre* verifies the discrete part and EV_C evaluates the continuous elements to produce *g*, a continuous function to specify the behavior of the continuous part.

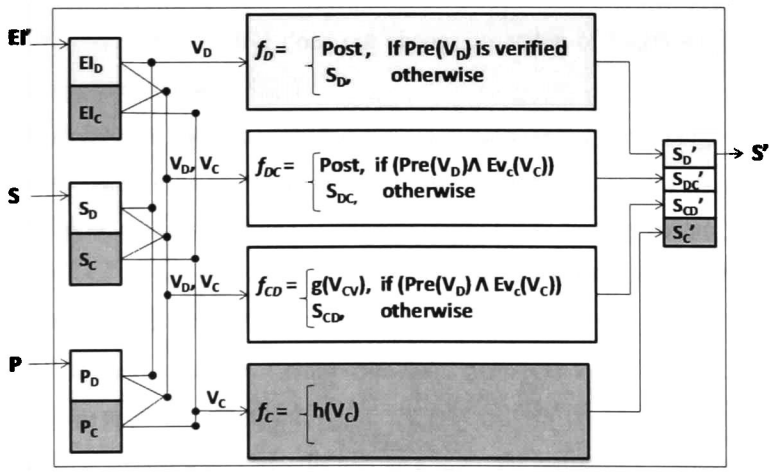


Figure 3.2. Module Evolution Rules for S

f_{CD} updates S_{CD}, continuous part influenced by the continuous and discrete variables (V_C, V_D). Finally f_C is a continuous function to evolve the pure continuous part of the state S_C.

3.4. Interfacing

A message sent between the agents can be specified by a 4-tuple $<A_i,A_j,TM,SI>$ where:

- A_i is the agent who transmits the message and A_j is the agent who receives the message, both A_i, $A_j \in A$.
- $TM \in \{0,1\}$; where 0 means agent A_i request a task from agent A_j, and 1 means that agent A_i informs an status to agent A_j.
- SI is a vector containing the information sent, $SI=[SI_D, SI_C]$, where $SI_D(j) \in \mathbb{N}$ and $SI_C(j) \in \mathbb{R}$.

3.5. A case study

In this section an illustrative example related to a sensor network application is dealt. This example is based on that presented in [Franceschelli,2011]. The purpose is to solve the consensus on the average value measured by agents in a sensor network through a gossip algorithm based on broadcast. The network topology is fixed, thus each node knows its out-degree at any time. Each node can be then either transmitting information, receiving information, or in an idle state during the process of computing the average of the sensed value of sensors.

The network operates iteratively as follows:

- One transmitter node I is chosen randomly and it broadcasts its state value $Sx_i(t)$ to all neighbor nodes to whom agent I is able to communicate. Also, it broadcasts a value $Sz_i(t)$ and the number of all nodes that receive information from node I ($\delta_{out,i}$). The transmitter node I does not change its value of $Sx_i(t)$ while it resets to 0 the companion variable $Sz_i(t)$.
- The receiver node j update their $Sx_j(t)$ by computing the average between their own value and the received state value. Furthermore, it corrects its updated value by a fraction of their companion variable $Sz_j(t)$ by adding up several terms, designed to preserve the average of the network at each iteration while converging to the average of the initial measurements.
- For the idle nodes that neither transmit nor receive information, both variables Sx and Sz remain unchanged.

Besides the previous specification we consider an additional feature: the battery level of each node is computed as a function of the agent activity (messages transmitted and received).

Now we can define a model for the system based on the proposed framework. Two kinds of basic agents are defined, one to model the transmitter, receiver, and idle nodes (Agent A), and other to choose the transmitter node (Agent B).

Agent A has the vectors: Perception (P_A), External Information (EI_A) and State (S_A). These elements are shown in Table 3.1. In the perception vector P_A, the discrete variables are: Pb, Pe_j and Pa.

Vector	*Variable*	*Domain*	*Description*
P_A	Pb	{0,1}	Perform Broadcast
	Pe_1	{0,1,2}	Status of Agent 1
	...	{0,1,2}	...
	Pe_n	{0,1,2}	Status of Agent n
	Pa	$Pa \in E$	Transmitter agent's index
	Px	$\mathbb{R}$	Sx_{Pa} value
	Pd	$\mathbb{R}$	Sz_{Pa}/*Agent Pa* outdegree
EI_A	Eid	$\{j \mid (I,j) \in E\}$	Neighbors of agent
	Eie_1	{0, 1, 2}	Status of agent 1 $\in$ Eid
	...	{0, 1, 2}	...
	Eie_n	{0, 1, 2}	Status of agent n $\in$ Eid
S_A	Sx	$\mathbb{R}$	Sensed value (Average)
	Sz	$\mathbb{R}$	Companion variable
	Sb	$\mathbb{R}$	Agent's Battery

Table 3.1. Vectors for Agent A

- Pb indicates if the agent is a transmitter node (it will emit the broadcast).
- Pe_j indicates the received status of agent j; agents j are all the receiver nodes to whom agent i is able to communicate. For this variable: 0 means that the agent j has not executed the calculation, 1 means that the agent j has performed the calculation, and 2 means that agent j is out of battery.
- Pa is the index of the transmitter node, as a reference for the receiver nodes.
- The continuous variables are: Px and Pd; these variables are the sent values Sx and $Sz/\delta_{out,I}$ from the transmitter to the receivers.
- External information vector EI_A has only discrete values.
- Eid maintains a list of all its neighbors (It remains unchanged during the execution since the topology never change).
- Eie_j maintains the status of agent j. This information is relevant for the transmitter node.
- The state vector S_A is composed by the continuous values Sx, Sz, and Sb for the value of level battery.

Some status variables like Pe_j and Eie_j are not modeled in [Franceschelli, 2011], but they are necessary to coordinate the behavior of each agent.

Table 3.2 shows the relationship between the messages and the conditions to generate the messages sent by Agent A. The first message is sent by the transmitter to request a calculus and the two

following messages are sent by the receivers to indicate their status to the transmitter. The last message is sent by the transmitter to the Agent B to indicate that all the agents have performed the computation and a new transmitter can be chosen again.

Message	*Condition*
$\{i,j,0,\{i,Sx,Sz,\|Eid\|\}\}$, Where $j \in Eid$	$Pb = 1$
$\{I,Pa,1,\{1\}\}$	$Pa <> 0 \wedge Sb > 0$
$\{I,Pa,1,\{2\}\}$	$Pa <> 0 \wedge Sb = 0$
$\{I,B,1,\{1\}\}$	$(Eie_1 = 1 \parallel Eie_1 = 2) \wedge \ldots \wedge (Eie_n = 1 \parallel Eie_n = 2)$

Table 3.2. Message Function for Agent A

The vector perception changes according to the received messages. This relation is shown in Table 3.3. The evolution rules for EI_A are given in Table 3.4.

Variable	*Value*	*Condition*
Pb'	1	$\{B,i,0,\{1\}\}$
	0	*Otherwise*
Pe_j'	v	$\{j,i,1,\{v\}\}$
	0	*Otherwise*
Pa'	a	$\{a,i,0,\{a,xa,za,da\}\}$
	0	*Otherwise*
Px'	xa	$\{a,i,0,\{a,xa,za,da\}\}$
	0	*Otherwise*
Pd'	za/da	$\{a,i,0,\{a,xa,za,da\}\}$
	0	*Otherwise*

Table 3.3. Perception Function for Agent A

Variable	*Value*
Eid'	Eid
Eie_1'	Pe_1
...	...
Eie_n'	Pe_n

Table 3.4. Evolution of External Information for Agent A

The evolution rules for S are given in Figure 3.3. Observe that the state is continuous, but the influence is continuous and discrete. The agent's battery level has been expressed as a function (g_1) of its own battery when the agent is receiver, and as a function (g_2) of its own battery level and $\delta_{out,I}$ when the agent is the transmitter.

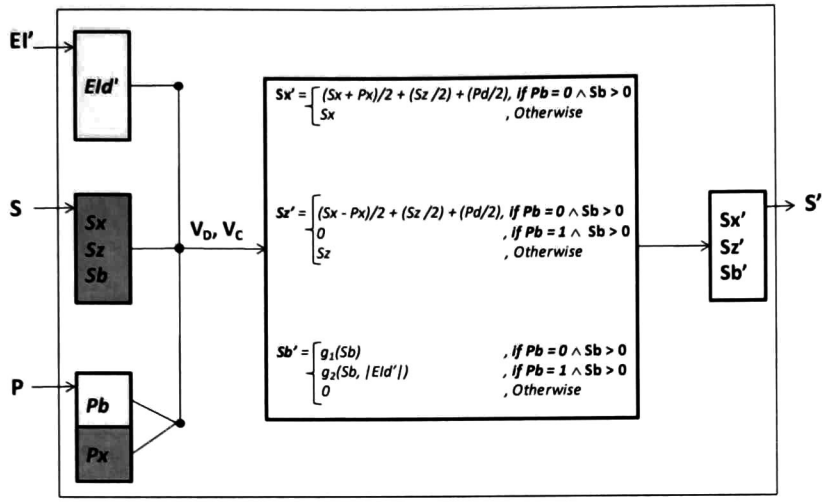


Figure 3.3. Evolution state for Agent A

Vector	*Variable*	*Domain*	*Description*
P_B	*Pe*	*A*	Transmitter agent Status
EI_B	*Eia*	$\{j \mid j \in A\}$	Set of agents system (*A*)
S_B	*Sr*	*A*	Agent chosen randomly to execute broadcast

Table 3.5. Vectors for Agent B

Message	*Condition*
$\{B,Sr',R,\{1\}\}$	$Pe = 1$

Table 3.6. Messages function for Agent B

Vector	*Variable*	*Value*	*Condition*
P_i'	*Pe*	1	$\{I,B,I,\{1\}\}$
		0	*Othewise*

Table 3.7. Perception function for Agent B

Vector	*Variable*	*Value*	*Condition*
S_i'	Sr'	Random(Eia)	$Pe = 1$
		Sr	*Otherwise*

Table 3.8. Evolution State for Agent B

The purpose of Agent B is to select the transmitter agent. The agent has the vectors shown in Table 3.5. The messages function, perception function, and state evolution are shown in Table 3.6, 3.7 and 3.8 respectively.

Chapter 4

MODELING NETWORKED AGENT SYSTEMS USING TIMED HYBRID PETRI NETS

Abstract. The work presented in this chapter deals with the modelling of networked identical agents. A framework for modelling this kind of systems specified by timed hybrid Petri nets (THPN) is presented. The state of an agent, expressed as a set of variables that can be either integer or real valued are represented in a THPN model, as well as their updating functions. The framework is illustrated through a case study regarding a sensor network. [Gudiño-Mendoza,2013].

4.1. State representation

The networked agent system evolution (*mA*) can be seen as a THPN where the evolution is performed by the fundamental equation, where the marking will be the state of the system, and the incidence matrix will establish the inner relations of each agent element (WA^i) and the communication between the agents (*WR*).

$$mA_{t2} = mA_{t1} + \left[\begin{array}{ccc|c} WA^1 & 0 & 0 & \\ 0 & \ddots & 0 & WR \\ 0 & 0 & WA^n & \end{array}\right] \cdot s \tag{10}$$

According to the agent generic scheme described in Figure 3.2 the agent state evolution (mA_i) can be represented by the following equation:

$$mA^i_{t2} = \begin{bmatrix} mS_{t2} \\ mEI_{t2} \\ mP_{t2} \\ mC_{t2} \end{bmatrix} = \begin{bmatrix} mS_{t1} \\ mEI_{t1} \\ mP_{t1} \\ mC_{t1} \end{bmatrix} + \left[\begin{array}{c|c} WA^i & \\ 0 & WR^i \end{array}\right] \cdot \begin{bmatrix} SA^i \\ SR^i \end{bmatrix} \tag{11}$$

The marking in each agent represents its own state (*mS*), its knowledge about environment and other agents (*mEI*), and its relations with other agents (*mP* and *mC*). The incidence matrix is divided in two parts: WA^i establishes the relation among the elements of the agent itself and WR^i its relationships with other agents. The vector SA^i contains the number of firings for discrete transitions and the firing velocities for continuous transitions; all these transitions are inner part of the agent. SR^i includes transitions that mark (unmark) places to transmit (receive) information between agents in communication; also, these transitions can be related with *mS* or *mEI*, a required marking for their firing.

4.2. Sensor networks representation

4.2.1. Generic sensor node representation

The Figure 4.1 shows the THPN model for a sensor node; it represents its basic parts: Communication (P11-P16, T11-T18), sensor (T6), memory (P7-P10, T9-T10), and battery (P1, T1-T6). There are also three operational modes (P2-P4). Basically, the memory part is modelled with two places, one of them controls the capacity of the memory (P8) and the other one the stored information (P7); this data comes from the sensor (T6) itself or from other sensors (P16, T18). This component has also a mechanism to transmit only when there is sufficient information in the memory (P9); similarly when there is enough space in the memory, incoming information is allowed (P10).

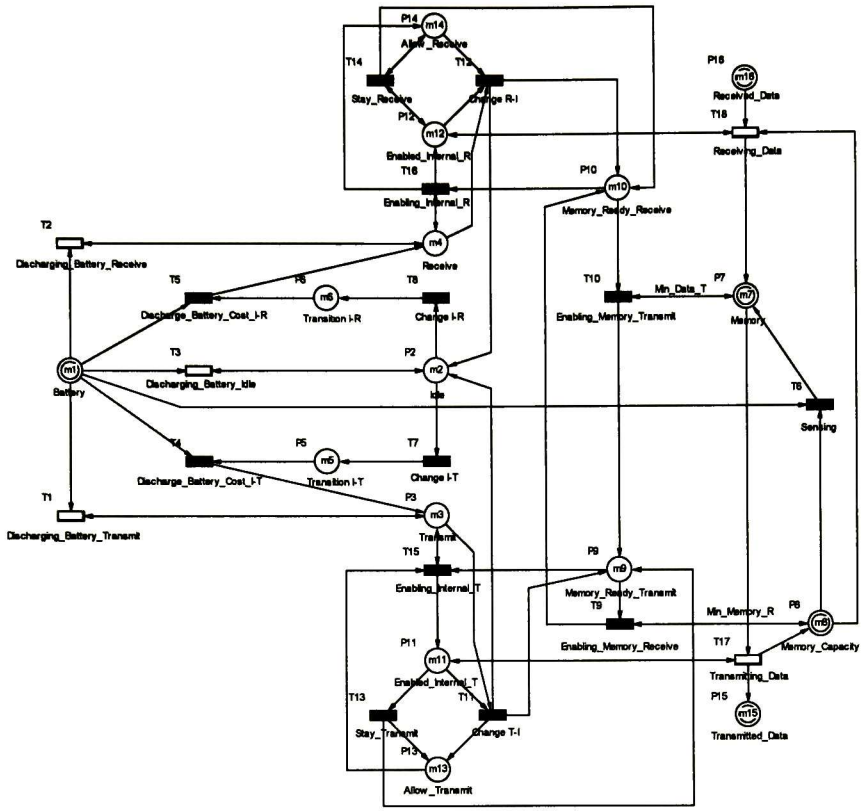


Figure 4.1 State representation for a Sensor node

The communication component is modelled by a mutual exclusion structure through places (P11-P12). When there is enough information to transmit (Min_Data_T) the sensor can transmit its data (P13). When the sensor has enough space in the memory (Min_Memory_R), it marks place (P10) indicating that it is ready to receive (P14); places (P15) and (P16) represent the transfer of information from one sensor to another one.

For this model three basic operational modes are considered: idle, receive, and transmit. The sensor may switch from one operational mode to another, but this switching has a cost in time and energy (P5-P6, T4-T5). The battery (P1) discharges according to the current operational mode (T1-T3) and also due to the sensing operation (T6). Table 4.1 shows a summarized description of the THPN nodes; using this notation the state evolution is described in the following section.

Component	Nat	Description	Node	Component	Nat	Description	Node
Battery	C	Battery	*P1 (B1)*	***Memory***	C	Memory	*P7 (M1)*
	C	Discharging_Battery_Transmit	*T1*		C	Memory_Capacity	*P8 (M2)*

	C	Discharging_Battery _Receive	*T2*		D	Memory_Ready_ Transmit	*P9 (M3)*
	C	Discharging_Battery _Idle	*T3*		D	Memory_Ready_ Receive	*P10 (M4)*
	D	Discharge_Battery_ Cost_I-T	*T4 (d1)*		D	Enabling_Memory_ Ready_Receive	*T9*
	D	Discharge_Battery_ Cost_I-R	*T5 (d2)*		D	Enabling_Memory_ Ready_Transmit	*T10*
Sensor	D	Sensing	*T6 (d3)*	***Communication***	D	Enabled_Internal_ Transmission	*P11 (C1)*
Operational Mode	D	Idle	*P2 (Om1)*		D	Enabled_Internal_ Reception	*P12 (C2)*
	D	Transmit	*P3 (Om2)*		D	Allow_Transmit	*P13 (C3)*
	D	Receive	*P4 (Om3)*		D	Allow_Receive	*P14 (C4)*
	D	Transition Idle-Transmit	*P5(Om4)*		C	Transmitted_Data	*P15 (C5)*
	D	Transition Idle-Receive	*P6 (Om5)*		C	Received_Data	*P16 (C6)*
	D	Change Idle-Transmit	*T7*		D	Enabling_Internal_ Transmition	*T15*
	D	Change_Idle-Receive	*T8*		D	Enabling_Internal_ Reception	*T16*
	D	Change_Transmit_ Idle	*T11*		C	Transmiting_Data	*T17*
	D	Change_Receive_ Idle	*T12*		C	Receiving_Data	*T18*
	D	Stay_Transmit_ Mode	*T13*				
	D	Stay_Receive_Mode	*T14*				

Table 4.1 Basic set of constraints for a TCPN

The making evolution for the sensor node can be described by the following marking evolution equation:

$$\begin{bmatrix} Bl_{t2} \\ Oml_{t2} \\ \vdots \\ Om5_{t2} \\ Ml_{t2} \\ \vdots \\ M4_{t2} \\ Cl_{t2} \\ \vdots \\ C3_{t2} \\ \vdots \\ C6_{t2} \end{bmatrix} = \begin{bmatrix} Bl_{t1} \\ Oml_{t1} \\ \vdots \\ Om5_{t1} \\ Ml_{t1} \\ \vdots \\ M4_{t1} \\ Cl_{t1} \\ \vdots \\ C3_{t1} \\ \vdots \\ C6_{t1} \end{bmatrix} + \begin{bmatrix} WA^t \\ \vdots \\ 0 \end{bmatrix} \begin{bmatrix} \vdots \\ WR^t \\ \vdots \end{bmatrix} \cdot S^t, \quad WA^t = \begin{bmatrix} -1 & -1 & -1 & -d1 & -d2 & -d3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad WR^t = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & -1 & 0 & 0 \\ -1 & & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$M_0 = [\max\_B \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad \max\_M \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0],$$

$$U = [fdt \quad fdr \quad fdi \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad ft \quad fr]$$

$$d = [0 \quad 0 \quad 0 \quad tdi\_t \quad tdi\_i \quad ts \quad tci\_t \quad tci\_r \quad tm\_r \quad tm\_t \quad tct\_i \quad tcr\_i \quad 0 \quad 0 \quad tei\_t \quad tei\_r \quad 0 \quad 0]$$

In the initial marking M_0, *max_B* and *max_M* define respectively the maximal battery and maximal memory capacity for the sensor node.

S^t contains a count of the number of firings from T1 to T2 and the firing velocity for continuous transitions; for this net the discrete places only can have at most one mark, so D-enabling degree always will be 1 or 0; thus, the maximal firing speed for continuous transitions is zero or the flow rate

associated with Tj. The flows *fdt*, *fdr*, and *fdi* are the battery discharging flows corresponding respectively to the states *transmit*, *receive*, and *idle*; flows *ft* and *fr* are the flows to receive or transmit data. The vector *d* represents the delays associated to discrete transitions, especially *ts*, denotes the testing frequency for the sensor node.

Chapter 5

COMPUTING THE SWITCHING DYNAMIC BEHAVIOR OF TCPN USING MLIP

Abstract. The behavior of timed continuous Petri nets (TCPN) can be ruled by linear equations during certain time elapses (IB-states), but changes in the marking and conflict solving policies make non linear the complete computation of the behavior. In this chapter the switching behavior of TCPN is analyzed through Mixed Linear Integer Programming (MLIP); basic constraints sets for TCPN without conflicts and with structural conflicts are considered, as well as cyclic structures.

5.1. Characterization of the behavior of a TCPN

The MILP is stated according to parameters issued from the following data: Incidence matrix (W), initial marking (M_0), *mfs* (V), maximum time of computation (tC), minimum (sE) and maximum (bE) durations for an IB-state, marking upper bound of places (mM), and number of IB-states (K).

The marking upper bound of places (mM) along tC is a necessary information regardless the boundedness of the Petri net. The calculation is performed for a period of time (tC). At the beginning, the number of IB-states is not known although it is at most $2^{|P|}$, where |P| is the number of places in the TCPN. K can be initialized to $2^{|P|}$. When the computed IB-states have the same *ifs* vector, they correspond to an actual IB-state; it is illustrated in Example 2.

The examples presented in this chapter were solved by a PC i7-2600 3.4Ghz using the CPLEX 12.4 package solver running under Matlab 7.9.0. [Cplex,2013].

5.2. TCPN without structural conflicts

The Table 5.1 describes the first set of constraints in the MILP for analyzing a TCPN without structural conflicts. Constraints (13), (14), (16), and (19) are still nonlinear, in the following section the transformation to linear constraints is presented. The evolution marking (12) is expressed through the vector f_k (13) those entries represent the flow of every transition in an IB-state k. This bilinear relation is transformed into a linear one using relations (2) and (3)-(6). The *ifs* (14) acts as a selector between all the possible values that it may have; it considers the *mfs* and all the input flows from the input places to each T_j (15). Through discrete variables (λ) only one value is selected (18). If the transition has input unmarked places, then the minimum flow is chosen according to (14) and (15), and for the non negative marking in places (22) is used (empty places input flow must be bigger than or equal to the output flow). If a place is marked then this flow must not be considered (Restriction (16)). If all the input places to a transition are unmarked and its input flows are equal to zero, then the transition must not be fired; this is stated with restriction (17), which considers case D illustrated in Section 1.2.3. dt_k is discretized according to sE (Minimum duration for an IB-state); with this, it is possible to transform the nonlinear restriction (13) into a linear one. Restriction (21) is an upper bound of the total duration of the results. Finally, the optimization function tries to find the biggest amount of time for each IB-state.

Evolution marking	(12) $\boldsymbol{m_{k+1} = m_k + W \cdot f_k}$
Flow in the interval time k	(13) $\boldsymbol{f_k = v_k \cdot dt_k}$
Instantaneous firing speed $\boldsymbol{v_k = [v_{k,1} \dots v_{k,\lvert T\rvert}]}$	(14) $v_{k,j} = V_j \cdot \lambda_{1,k,j,0} + \sum_{P_i \in {}^{o}T_j} I_i \cdot \lambda_{1,k,j,i}$

	(15)	$I_i = \frac{\left(\sum_{T_h \in {}^\circ P_i} Post(P_i, T_h) \cdot v_{k,h}\right)}{Pre(P_i, T_j)}$ *where* $P_i \in {}^\circ T_j$
	(16)	$if\ [m(P_i) > 0, \forall P_i \in {}^\circ T_j\ or\ {}^\circ T_j = \emptyset] \rightarrow [\lambda_{1,k,j,0} = 1]$
	(17)	$if\ [m(P_i) = 0 \wedge I_i = 0, \forall P_i \in {}^\circ T_j\ ] \rightarrow [v_{k,j} = 0]$
	(18)	$\sum_{i=1}^{P+1} \lambda_{1,k,j,i} = 1$
Duration of time k IB-state	(19)	$\Delta t_k = sE \sum_{i=0}^{\lceil log_2(bE/sE) \rceil} 2^i \cdot \lambda_{2,k,i}$
	(20)	$\sum_{i=0}^{\lceil log_2(bE/sE) \rceil} \lambda_{2,k,i} \geq 1$
Maximum time of computation	(21)	$\sum_{i=1}^{K} \Delta t_i \leq tC$
s.t.	(22)	$0 \leq m_k \leq mMax$ $0 \leq v_k \leq V$ $m_k, v_k, \Delta t_k \in \mathbb{R}$ $\lambda \in \{0,1\}$
Optimization function	(23)	$Max\ f = \sum_{i=1}^{K} \Delta t_i$

Table 5.1 Basic set of constraints for a TCPN

The idea of restriction *(14)* is illustrated in Figure 5.1. All the places are unmarked, which means that T_4 is weakly enabled and T_1 T_2 and T_3 are strongly enabled. The constraints used to calculate the *ifs* are shown in the Table 5.2. Notice that T_1-T_3 fire at their maximal speed, thus the firing of T_4 depends on the input flow from $P_1(I_1=3)$, $P_2(I_2=2)$ and $P_3(I_3=1)$ and its own *mfs* ($V_4=4$); flows 4, 3, and 2 cannot be chosen because this would cause the marking of place P_3 become negative (its input flow is 1), so the selected value is $v_4=1$.

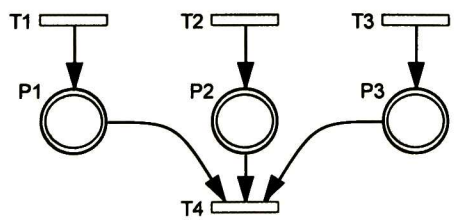


Figure 5.1. TCPN to illustrate the basic set of constraints

Constraints
$v_1 = 3 \cdot \lambda_{1,0},\ \lambda_{1,0} = 1$
$v_2 = 2 \cdot \lambda_{2,0},\ \lambda_{2,0} = 1$
$v_3 = 1 \cdot \lambda_{3,0},\ \lambda_{3,0} = 1$
$v_4 = 4 \cdot \lambda_{4,0} + 3 \cdot \lambda_{4,1} + 2 \cdot \lambda_{4,2} + 1 \cdot \lambda_{4,3}$
$\lambda_{4,0} + \lambda_{4,1} + \lambda_{4,2} + \lambda_{4,3} = 1$
Solution
$v_1 = 3, v_2 = 2, v_3 = 1, v_4 = 1$

Table 5.2. Constraints and solution for Figure 5.1

5.2.1. Linear constraints for logical and nonlinear restrictions

The bilinear relations (13) and (14) and the logical sentences (16) and (19) are equivalent to the linear constraints presented in Table 5.3. Variables initialized as *AC* are additional continuous variables.

Nonlinear constraint	*Equivalent linear constraints*	
Flow in the interval time k (13) $f_k = v_k \cdot dt_k$	(24) $f_k = \sum_{i=0}^{\lfloor \log_2(\Delta E/sE) \rfloor} AC1_{k,i}$ (25) $AC1_{k,i} - Max(v_k) \cdot \lambda_{2,k,i} \le 0$ (26) $AC1_{k,i} - sE \cdot v_k \le 0$ (27) $-AC1_{k,i} + sE \cdot v_k - Max(v_k) \cdot (1 - \lambda_{2,k,i}) \le 0$	$\forall k \in [1, K]$
Instantaneous firing speed (14) $v_{k,j} = V_j \cdot \lambda_{1,k,j,0} + \sum_{P_i \in {}^\circ T_j} I_i \cdot \lambda_{1,k,j,i}$	(28) $v_{k,j} = V_j \cdot \lambda_{1,k,j,0} + \sum_{P_i \in {}^\circ T_j} AC2_{k,j,i}$ (29) $AC2_{k,j,i} - Max(I_i) \cdot \lambda_{1,k,j,i} \le 0$ (30) $AC2_{k,j,i} - sE \cdot I_i \le 0$ (31) $-AC2_{k,j,i} + sE \cdot I_i - Max(I_i) \cdot (1 - \lambda_{1,k,j,i}) \le 0$	
Instantaneous firing speed (16) $if\ [m_{k,i} > 0] \rightarrow [\lambda_{1,k,j,i} = 0]$	(32) $m_{k,i} \le (1 - \lambda_{1,k,j,i}) \cdot mM$ (33) $m_{k,i} \ge (\lambda_{1,k,j,i} - 1) \cdot \varepsilon$	
Instantaneous firing speed (19) $if\ [m_{k,i} = 0 \wedge I_i = 0, \forall P_i \in {}^\circ T_j] \rightarrow [v_{k,j} = 0]$	(34) $\sum_{P_i \in C_j} m(P_{k,i}) + \sum_{T_i \in TC_j} v_{k,i} \le (1 - \lambda_{3,k,j}) \cdot Max\left(\sum_{P_i \in C_j} m(P_{k,i}) + \sum_{T_i \in TC_j} v_{k,i}\right)$ (35) $\sum_{P_i \in C_j} m(P_{k,i}) + \sum_{T_i \in TC_j} v_{k,i} \ge (1 - \lambda_{3,k,j}) \cdot sE$ (36) $\sum_{j=1}^{n} \lambda_{3,k,j} \le (1 - \lambda_{4,k}) \cdot (.5) + \lambda_{4,k} \cdot Max\left(\sum_{j=1}^{n} \lambda_{3,k,j}\right)$ (37) $\sum_{j=1}^{n} \lambda_{3,k,j} \ge (1 - \lambda_{4,k}) + \lambda_{4,k} \cdot (.5)$ (38) $v_{k,j} \le (1 - \lambda_{4,k}) \cdot V_j$	

Table 5.3 Nonlinear constraints for a TCPN without conflicts expressed in linear constraints

The equivalent constraints for (13) and (14) are the same because both contain a binary variable multiplied by a continuous variable. Chapter 2 can serve as reference for these transformations as well as for (16).

A strongly enabled transition can be fired at its *mfs* because of (16) and (18). Also these constraints allow the condition of weakly enabled; but when there is an unmarked cycle, with just these constraints is not enough. See TCPN for Figure 3(a), Case D, it has only one IB-State where *ifs* is [0, 0], but without constraint (19), there are two possible solutions [0,0] and [1,1], any of them satisfy all constraints. This is why (19) is necessary.

The transformation of (19) requires some definitions [Desel, 1995]. Let F a flow relation on $P \cup T$, based on the incidence matrix. A circuit of a Petri net is a nonempty sequence $x_1 \ldots x_1$ of nodes

which satisfies$(x_1, x_2), \ldots, (x_n, x_1) \in F$ and no elements (except x_1) occurs more than once in it. Let $C_1$and C_2 the sets containing the elements of two circuits, if $C_1 \subsetneq C_2$, C_1 is called maximal circuit.

Let T_h a transition in a relation $Pre(P_i, T_h)$, where $P_i \in C_i$ and $T_h \notin C_i$, then $T_h \in TC_i$. Where TC_i represent the set of all transitions connected to the circuit C_i that eventually could mark it.

In order to establish the transitions whose *ifs* will be zero, it is necessary to find all the maximal circuits in the net. And for each transition included in a maximal circuit the following clause must be added:

(39) $$\left[\sum_{P_i \in C_1} m_{k,i} + \sum_{T_i \in TC_1} v_{k,i} = 0\right] \vee \ldots \vee \left[\sum_{P_i \in C_n} m_{k,i} + \sum_{T_i \in TC_n} v_{k,i} = 0\right] \rightarrow [v_{k,j} = 0], \text{where } T_j \in C_1, \ldots, \text{and } T_j \in C_n$$

This is equivalent to (19); in other words it means: For each maximal circuit $(C_1, C_2, \ldots, C_n)$ containing transition T_j, it is necessary to verify if the circuit is unmarked (the term $\sum_{P_i \in C_1} m_{k,i}$ in (45)), and if it does not receive any other incoming flow that could mark the circuit (the term $\sum_{T_i \in TC_1} v_{k,i}$ in (45)). If this is true for at least one circuit in which T_j is involved, then its *ifs* must be zero.

Let's see an example. Figure 5.2 shows a TCPN with two maximal circuits $C_1 = \{P_1, T_2, P_2, T_3\}$ and $C_2 = \{P_5, T_4, P_4, T_5\}$, all the places are unmarked and V=[3 3 1 2 1]; circuit C_1 can be marked by T_1, so $TC_1 = \{T_1\}$. Adding these constraints to the MILP, the solution shown in Table 5.4 can be obtained. In spite of the unmarked places in C_1, through T_1; v_1, v_2 and v_3 acquire velocity; which does not occur with v_4 and v_5, C_2 is unmarked and it will remain like that.

Constraints	***Solution***
$[m_1 + m_2 + v_1 = 0] \rightarrow [v_2 = 0]$; $[m_1 + m_2 + v_1 = 0] \rightarrow [v_3 = 0]$; $[m_5 + m_4 = 0] \rightarrow [v_4 = 0]$; $[m_5 + m_4 = 0] \rightarrow [v_5 = 0]$	$v_1 = 3, v_2 = 3, v_3 = 1, v_4 = 0, v_5 = 0$

Table 5.4 Constrains for Figure 11

Constraint (39) is still nonlinear; it is transformed to the basic constraints shown in Table 5.5. Constraint (40) associates an additional binary variable to each marking and velocity involved in a circuit; the results of the disjunctions are assigned to other logical variable λ_4 (41); and it only remains to verify: if λ_4 is true then *ifs* for T_j must be equal to 0 (42).

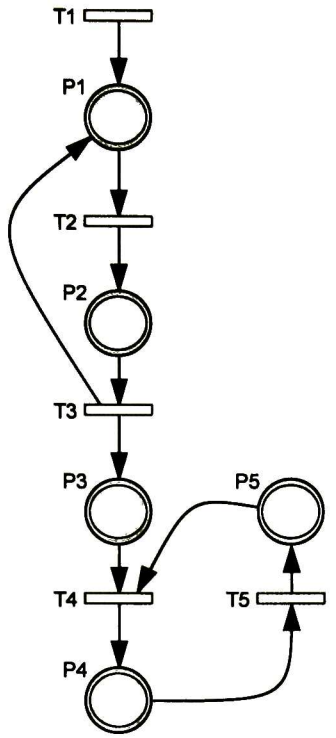


Figure 5.2. TCPN with two maximal circuits

	Basic logical sentences	Linear constraints
(40)	$\left[\sum_{P_i \in C_j} m(P_{k,i}) + \sum_{T_i \in TC_j} v_{k,i} = 0\right] \leftrightarrow [\lambda_{3,k,j} = 1]$	(34) and (35)
(41)	$[\lambda_{3,k,1} = 1] \vee \ldots \vee [\lambda_{3,k,j} = 1] \leftrightarrow [\lambda_{4,k} = 1]$	(36) and (37)
(42)	$[\lambda_{4,k} = 1] \rightarrow [v_{k,j} = 0]$	(38)

Table 5.5 Equivalent logical sentences for (39)

5.2.2. Examples

These examples are based on the same structural model without conflicts (Figure 5.3). Its behavior is stated using the basic constraints (12)-(22) and the optimization function (23). The information that represents each IB-state is computed and listed in the Table 5.6: (IB-S) is the number of the IB-state, *dt* is the duration of the IB-state, *v* is the *ifs*, and *m* is the marking at the end of the IB-state. The value between parenthesis in each case is the time required to compute the results.

Example 1 shows the information of the two IB-states covered in 20 units of time. In Example 2, *K* was increased to show how two consecutive IB-states (1 and 2) have the same *ifs*; this information must be interpreted as one IB-state. In Example 3 the initial marking was modified to show what happen if the time of computation cannot be covered by the given number of total IB-states (*K*); in

the last IB-states (In this case is 2) the *ifs* will be 0 and the marking will remain the same as in the previous one.

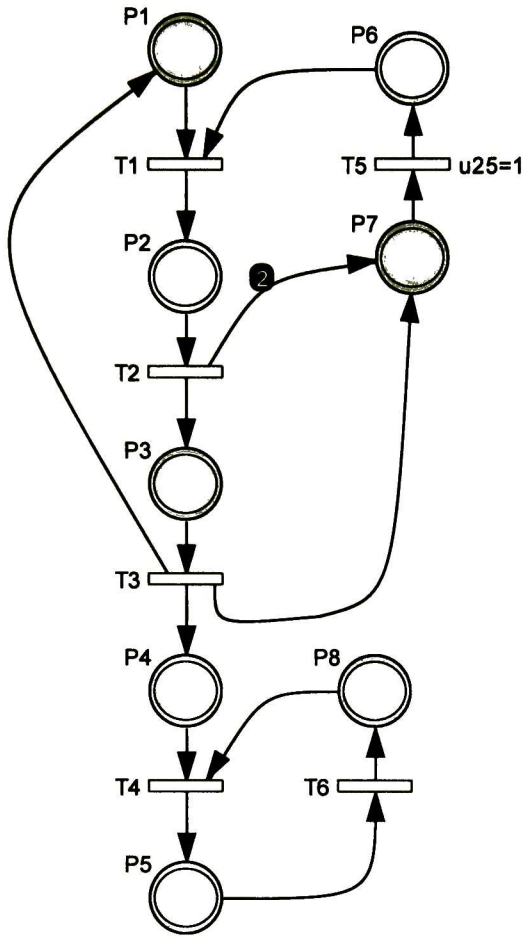


Figure 5.3. Basic TCPN without conflicts

Initial data	*Results*			
	IB-S	*Δt*	*v*	*m*
Example 1 (0.2001 s)				
M_0=[6 0 0 0 0 0 1 0] V=[3 3 1 2 2 1] tC=20, sE=1, bE=20, mM= 50, K=2	1	6	[2 2 1 0 2 0]	[0 0 6 6 0 0 19 0]
	2	14	[1 1 1 0 2 0]	[0 0 6 20 0 14 33 0]
Example 2 (0.1930 s)				
M_0=[6 0 0 0 0 0 1 0] V=[3 3 1 2 2 1] tC=20, sE=1, bE=20, mM= 50, K=3	1	1	[2 2 1 0 2 0]	[5 0 1 1 0 0 4 0]
	2	5	[2 2 1 0 2 0]	[0 0 6 6 0 0 19 0]
	3	14	[1 1 1 0 2 0]	[0 0 6 20 0 14 33 0]
Example 3 (0.0470 s)				
M_0=[0 0 0 0 0 0 1 0] V=[3 3 1 2 2 1] tC=20, sE=0.1, bE=20, mM= 50, K=2	1	0.5	[0 0 0 0 2 0]	[0 0 0 0 0 1 0 0]
	2	19.5	[0 0 0 0 0 0]	[0 0 0 0 0 1 0 0]

Table 5.6. Several examples for TCPN of Figure 5.3

5.3. TCPN with conflicts

In the following two sections some additional constraints are presented in order to compute the *ifs* for transitions in a conflict $SC = \langle Pc, Tc \rangle$. The TCPN can have several conflicts in its structure but only one place can be involved in just one conflict as well as the transitions. This means that for a TCPN with the following conflicts $SC_1, SC_2, \ldots, SC_s$ the sets $Pc_1 \cap Pc_2 \ldots \cap Pc_s = \emptyset$ and $Tc_1 \cap Tc_2 \ldots \cap Tc_s = \emptyset$.

5.3.1. Resolution by priorities

For a TCPN with a conflict $SC = \langle Pc, Tc \rangle$ where the conflict is solved by priority ($T_1 < T_2 < \ldots < T_s$), constraints *(43)-(45)* replace restriction *(14)*.

(43)	$v_{k,j} = V_j \cdot \lambda_{1,k,0} + \sum_{P_i \in {}^{\bullet}T_j} Ic_i \cdot \lambda_{1,k,i}$
(44)	$[P_i \notin Pc] \rightarrow [Ic_i = I_i]$
(45)	$[P_i \in Pc] \rightarrow \left[Ic_i = I_i - \sum_{T_h \in Tc \text{ and } T_h < T_j} v_h \right]$

Table 5.7. Set of constraints for a TCPN with resolution by priority

The basic idea is illustrated through the example of Figure 5.4. This net has the resolution rule $T_2 < T_3$. The constraints that determine the *ifs* for T_1 are *(43)* and *(16)*; this value only depends on the *mfs*, there is no place restricting the flow. For transitions T_2 and T_3 the constraints *(43)* and *(45)* are applied. The *ifs* of T_2 (with the highest priority) is computed by considering its maximal speed and the incoming flow from P_1. The *ifs* of T_3 (with low priority) is computed by considering its own *mfs* and the remaining flow in P_1 after satisfying v_2; thus $v_3 = 0$.

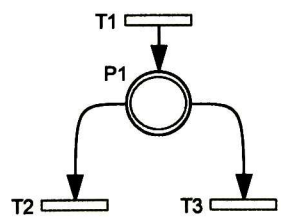


Figure 5.4. Basic TCPN without conflicts

Constraints	*Solution*
$v_1 = 1 \cdot \lambda_{1,0}$ $v_2 = 2 \cdot \lambda_{2,0} + 1 \cdot \lambda_{2,1}$ $v_3 = 1 \cdot \lambda_{3,0} + (I_1 - v_2) \cdot \lambda_{3,1}$	$v_1 = 1, v_2 = 1, v_3 = 0$

Table 5.8. Constraints and solution for Figure 5.4.

5.3.1.1. Examples

For examples 4 and 5 the resolution rule $T_4 < T_5$ is applied (Figure 5.5). In order to solve this model, constraints *(12)*, *(13)*, *(15)-(22)*, *(43)-(45)* and optimization function *(23)* are considered. In Example 4, conformed by only one IB-state, there is not actual conflict; this is because the flow of the two

places in conflict is limited by places P_1 and P_3 respectively. For Example 5, the two possible behaviors for a transition in conflict are shown. Three parameters have been modified M_0, V and k. Each IB-state shows different characteristics; the first one with all places marked, all the transitions go at the *mfs*; in the second, P_3 becomes empty and it makes T_5 weakly enabled, then P_3 limits the flow of T_5 to v_5=2. After 8 units of time, place P_2 becomes empty and the resolution rule is applied; T_5 must reduce its *ifs* to 1, satisfying first the flow v_4=5 for T_4.

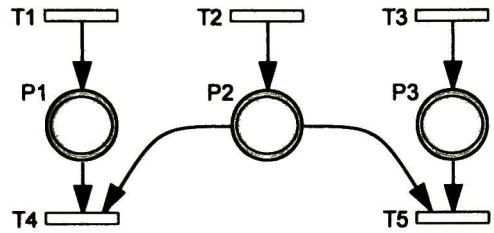


Figure 5.5. TCPN with a conflict resolution by priorities

Initial data	*Results*			
	IB-S	*Δt*	*v*	*m*
Example 4 (0.0630 s)				
M_0=[0 0 0] V=[2 6 2 5 3] tC=20, sE=1, bE=20, mM= 50, K=1	1	20	[2 6 2 2 2]	[0 40 0]
Example 5 (0.0310 s)				
M_0=[6 10 1]	1	1	[5 6 2 5 3]	[6 8 0]
V=[5 6 2 5 3]	2	8	[5 6 2 5 2]	[6 0 0]
tC=20, sE=1, bE=20, mM= 50, K=3	3	11	[5 6 2 5 1]	[6 0 11]

Table 5.9. Several examples for Figure 5.5.

5.3.2. Resolution by priorities in a cyclic TCPN

These nets are similar to that in case E.2 in Section 1.2.3 and the calculation of *ifs* requires some additional constraints. The idea is based in the iterative Algorithm 2 presented in Section 2.2.1. The calculation of the *ifs* for one IB-state is performed in several passages $l = 1, 2, \ldots, s$. For a TCPN with a conflict $SC = <Pc, Tc>$ where it is solved by priority ($T_1 < T_2 < \ldots < T_s$), s passages will be required. In the first passage $l = 1$, all the transitions in the conflict are forced to fire at 0 except T_1 *(53)*, $v_1 \geq 0, v_2 = 0, v_3 = 0, \ldots, v_s = 0$. For passage $l = 2, v_3 = 0, v_4 = 0, \ldots, v_s = 0$ and so on; in the last passage $l = s$ no transition is limited. In each passage, the *ifs* is computed as for a TCPN with conflicts solved by priority *(46)-(48)*; an additional variable called balance *Bc* of the place in conflict *Pc* is calculated *(49)-(51)*; this value represents the remaining flow in *Pc* after the firing all the transitions in the conflict that are allowed to fire. If *Bc* >0, then T_2 is allowed to fire in the next passage. In each passage one transition in the conflict having the highest priority may be fired *(53)*, but their *ifs* always will be limited by *Bc (52)*. The remaining flow after fire the transitions with the highest priority. The *ifs* of the last passage will be the *ifs* of the IB-state *(55)*.

(46)	$v^l_{k,j} = V_j \cdot \lambda_{1,k,0} + \sum_{P_i \in {}^\circ T_j} I^l c_i \cdot \lambda_{1,k,j,i}$
(47)	$[P_i \notin Pc] \rightarrow [I^l c_i = I_i]$
(48)	$[P_i \in Pc] \rightarrow \left[I^l c_i = I^l{}_i - \sum_{T_h \in Tc \text{ and } T_h < T_j} v^l{}_h \right]$
(49)	$B^l{}_c = I^l c_i - O^l c_i$
(50)	$I^l{}_i = \frac{(\sum_{T_h \in {}^\circ P_i} Post(P_i, T_h) \cdot v_{k,h})}{Pre(P_i, T_j)}$ where $P_i \in {}^\circ T_j$
(51)	$O^l c_i = \sum_{v_h \in Tc} v^l{}_h$
(52)	$B^l{}_c + \sum_{j=2}^{S} v^l{}_{k,j} \geq \sum_{j=2}^{S} v^{l+1}{}_{k,j}$
(53)	$\forall j > l, \quad v^l{}_{k,j} = 0$
(54)	Where $l = 1,2, \ldots s$
(55)	$v_{k,j} = v^S{}_{k,j}$

Table 5.10. Set of constraints for a TCPN with a circuit and resolution by priority

5.3.2.1. Example

Example 6 (Figure 5.5) shows the results of the computation of the MLIP with constraints *(12)*, *(13)*, *(16)–(22)*, *(46)-(55)* and the optimization function *(23)*. This net has the resolution rule $T_1 < T_2 < T_3$. In the first two IB-states there is no actual conflict the *ifs* for the transitions in the conflict are the *mfs*. In the IB-state 3 the incoming flow I_1 only allows firing T_1 at maximum. The last IB-state v_1 is restricted to 1.

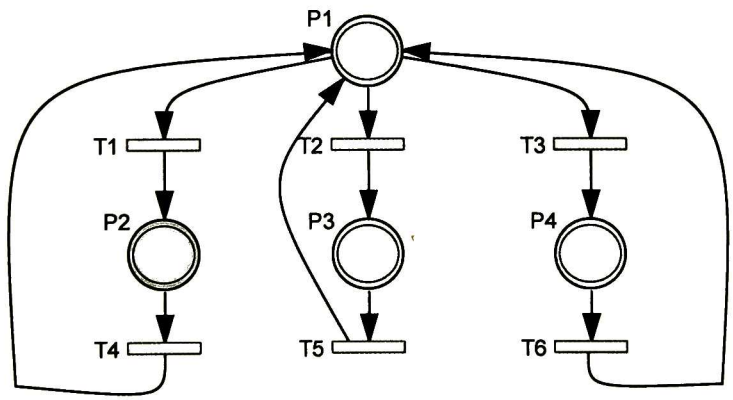


Figure 5.6. TCPN with a conflict and a circuit, resolution rule by priorities

Initial data	*Results*			
	IB-S	*Δt*	*v*	*m*
Example 6(0.3280)				
M_0=[0 0 2 12] V=[2 3 1 1 1 5]	1	3	[2 3 1 1 1 5]	[3 3 8 0]
	2	1	[2 3 1 1 1 1]	[0 4 10 0]
	3	10	[2 0 0 1 1 0]	[0 14 0 0]

tC=20, sE=1, bE=20, mM= 14, K=4	4	6	[1 0 0 1 0 0]	[0 14 0 0]

Table 5.11. Example for Figure 5.5

5.3.3. Resolution by sharing

In a TCPN with a conflict $SC = \langle Pc, Tc \rangle$ which is solved by sharing ($[\alpha_1 T_1, \alpha_2 T_2, \ldots < \alpha_s T_s)$, restriction *(14)* is substituted by *(56)-(57)*. The idea is to characterize the three possible cases presented in Section 1.2.2. Restriction *(56)* is in fact the same than *(14)*; it means that for all the transitions not involved in a conflict the calculation of v_j remains the same.

(56)	$if\ [T_j \notin Tc] \rightarrow v_{k,j} = V_j \cdot \lambda_{1,k,j,0} + \sum_{P_i \in {}^{\circ}T_j} I_i \cdot \lambda_{1,k,j,i}$
(57)	$if\ [T_j \in Tc] \rightarrow pcv_{k,j} = V_j \cdot \lambda_{1,k,j,0} + \sum_{P_i \in {}^{\circ}T_j} Is_i \cdot \lambda_{1,k,j,i}$
(58)	$Is_i = \frac{\left(\sum_{T_h \in {}^{\circ}P_i} Post(P_i, T_h) \cdot v_{k,h}\right)}{Pre(P_i, T_j)}$ *where* $P_i \in {}^{\circ}T_j$ *and* $P_i \neq P_c$
(59)	$if\ \left[\sum_{i=1}^{s} pcv_{k,j} \leq I_c\right] \vee [m_c > 0] \rightarrow [v_{k,j} = pcv_{k,j}]$
(60)	$else \quad [\sum_{i=1}^{s} v_i = I_c]$
(61)	$if\ [v_{k,j} = pcv_{k,j}] \rightarrow [\exists\ \alpha_h v_h \geq \alpha_j v_j]$
(62)	$else\ if\ [v_{k,j} < pcv_{k,j}] \wedge [v_{k,h} < pcv_{k,h}] \rightarrow [\alpha_j v_j = \alpha_h v_h]$ *where* $T_h \in Tc$

Table 5.12. Set of constraints for a TCPN with a circuit and resolution by sharing

In this solution policy it is introduced a new variable called previous calculus of v_j , pcv_j *(57)*. It stores the limit flow of v_j without considering the input flow from the place in conflict *(58)*. The block if...then *(59)-(62)* represents the constraints that must be considered according to some conditions. As mentioned before, the assignation to v_j must be restricted differently when one of three cases is presented:

- There is no actual conflict (Case A from Section 1.2.2), $v_j = pcv_j$, *(59)*
- There is an actual conflict no matter how the input flow *Ic* is shared among all the transitions in *Tc*, the total flow must be assigned to all of them *(60)*, similarly to Cases B and C (Section 1.2.2). $I_c = v_4 + v_5 + v_6 = 3$. And then:
    - If some transition in *Tc* is restricted by other flow different from *Pc*, this is because there exist at least other transition taking more than its corresponding portion. See Case C.1. $[v_4 = pcv_4] \rightarrow [v_5 \geq v_4], v_4 = I_1 = 0.5,\ v_5 = 1.25$; recall that in this case $\alpha_1 = \alpha_2 = \alpha_3 = 1$.
    - If there are two transitions in the conflict taking less flow than the previously calculated, then the portion taking from *Ic* must be shared according to the resolution rule. In Case C.1 $[v_5 < pcv_5] \wedge [v_6 < pcv_6] \rightarrow [\alpha_5 v_5 = \alpha_6 v_6], v_5 = v_6 = 1.25$.

For case C.3, the constraints apply in the following way: $pcv_4 + pcv_5 + pcv_6 = 0.5 + 3 + 0.4 = 3.9 > I_c$; so there is an actual conflict, constraints *(60)* and *(61)* must be considered. T_4 and T_6 cannot consume its corresponding flow from I_2 (the shared place), since both are restricted by other input places. The restriction *(61)* applies for both $[v_4 = pcv_4] \rightarrow [v_5 \geq v_4], v_4 = I_1 = 0.5,\ v_5 = 2.1$ and $[v_6 = pcv_6] \rightarrow [v_5 \geq v_6], v_4 = I_1 = 0.4,\ v_5 = 2.1$; in this case restriction *(62)* does not apply because there are not two transitions taking less flow than the previously calculated pcv. Restriction *(60)* assigns the value 2.1 to v_5.

5.3.4. Linear constraints for logical restrictions

Table 5.13 summarizes the equivalent constraints for each nonlinear equation presented in the previous section.

Nonlinear constraint	***Equivalent linear constraints***
No actual conflict *(59)* $if \left[\sum_{i=1}^{S} pcv_{k,j} \leq I_{k,C}\right] \vee [m_{k,c} > 0] \rightarrow [v_{k,j} = pcv_{k,j}]$	*(63)* $I_{k,c} - \sum_{T_j \in T_c} pcv_{k,j} \geq Min\left(\sum_{T_j \in T_c} pcv_{k,j}\right) \cdot \lambda_{5,k,j}$ *(64)* $I_{k,c} - \sum_{T_j \in T_c} pcv_{k,j} \leq -mM + \left(Max\left(\sum_{T_j \in T_c} pcv_{k,j}\right) + mM\right) \cdot (1 - \lambda_{5,k,j})$ *(65)* $m_{k,c} \leq (1 - \lambda_{6,k,c}) \cdot mM$ *(66)* $m_c \geq (\lambda_{6,k,c} - 1) \cdot \varepsilon$ *(67)* $-\lambda_{5,k,j} + \lambda_{7,k,j} \leq 0$ *(68)* $-\lambda_{6,k,c} + \lambda_{7,k,j} \leq 0$ *(69)* $\lambda_{5,k,j} + \lambda_{6,k,c} - \lambda_{7,k,j} \leq 1$ *(70)* $v_{k,j} \leq pcv_{k,j}$ *(71)* $v_{k,j} - pcv_{k,j} \geq -\varepsilon + (-V_k + \varepsilon) \cdot \lambda_{7,k,j}$
Actual conflict *(60)* $else \quad [\sum_{i=1}^{S} v_i = I_c]$	*(72)* $\sum_{i=1}^{s} v_i - I_c \leq (1 - \lambda_{7,k,j}) \cdot Max\left(\sum_{i=1}^{s} v_i = I_c\right)$ *(73)* $\sum_{i=1}^{s} v_i - I_c \geq (1 - \lambda_{7,k,j}) \cdot -Max\left(\sum_{i=1}^{s} v_i = I_c\right)$
Partial rule application *(61)* $if \left[v_{k,j} = pcv_{k,j}\right] \rightarrow \left[\exists\ \alpha_h v_{k,h} \geq \alpha_j v_{k,j}\right]$	*(74)* $pcv_{k,j} - v_{k,j} \geq -V_k \cdot \lambda_{8,k,c} + \varepsilon \cdot (1 - \lambda_{8,k,c})$ *(75)* $pcv_{k,j} - v_{k,j} \leq (1 - \lambda_{8,k,c}) \cdot V_k$ *(76)* $\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} \geq -V_k \cdot \lambda_{9,q,k,c} + \varepsilon \cdot (1 - \lambda_{9,q,k,c})$ *(77)* $\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} \leq V_k \cdot (1 - \lambda_{9,q,k,c})$ *(78)* $\sum_{q=1}^{\frac{1}{2}C} \lambda_{9,q,k,c} \leq (\frac{1}{2}C + 1) \cdot \lambda_{10,k,c}$ *(79)* $\sum_{q=1}^{\frac{1}{2}C} \lambda_{9,q,k,c} \geq \lambda_{10,k,c}$ *(80)* $(1 - \lambda_{7,k,c}) + (1 - \lambda_{8,k,c}) + \lambda_{10,k,c} \geq 1$
Rule application*(62)* $else\ if \left[v_{k,j} < pcv_{k,j}\right] \wedge \left[v_{k,h} < pcv_{k,h}\right] \rightarrow \left[\alpha_j v_{k,j} = \alpha_h v_{k,h}\right],\ where\ T_h \epsilon\ Tc$	*(81)* $\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} \leq (1 - \lambda_{11,k,i,p}) \cdot Max(\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p})$ *(82)* $\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} \geq \varepsilon + (Min(\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p}) - \varepsilon) \cdot \lambda_{11,k,i,p}$ *(83)* $-\alpha_i \cdot v_{k,i} + \alpha_p \cdot v_{k,p} \leq (1 - \lambda_{11,k,i,p}) \cdot Max(\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p})$ *(84)* $-\alpha_i \cdot v_{k,i} + \alpha_p \cdot v_{k,p} \geq \varepsilon + (Min(\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p}) - \varepsilon) \cdot \lambda_{11,k,i,p}$ *(85)* $v_{k,i} - pcv_{k,i} \geq Min(v_{k,i} - pcv_{k,i}) \cdot \lambda_{12,k,i} + \varepsilon(1 - \lambda_{12,k,i})$ *(86)* $v_{k,i} - pcv_{k,i} \leq Max(v_{k,i} - pcv_{k,i}) \cdot (1 - \lambda_{12,k,i})$ *(87)* $(\lambda_{7,k,c} - 1) + (\lambda_{12,k,i} - 1) + (\lambda_{12,k,p} - 1) + \lambda_{11,k,i,p} \geq 1$

Table 5.13 Nonlinear constraints for a TCPN with conflicts solved by sharing expressed as linear constraints

The constraints are described for TCPN with one conflict $SC = <Pc, Tc>$ which is solved by sharing ($[\alpha_1 T_1, \alpha_2 T_2, ... < \alpha_s T_s$), but it can be easily extended to cases where there is more than one conflict.

These nonlinear constraints can be translated in two steps:

1. Associating to each clause an additional logic variable and creating a dependency relation between these statements.

2. Translating these more basic logical relations using Table 4 into linear constraints.

Table 5.14 summarizes the conversion for constraints (59) and (60). λ_5 is true when the input flow in the place in conflict is not enough to satisfy the flows for all the transitions in the conflict (93); λ_6 is true when the place in conflict is unmarked (94); λ_7 is true when there is an actual conflict (95); v_j will always be at most pcv_j (96) whether or not there is a conflict, but when there is no conflict the flow chosen must be pcv_j and this is achieved with constraint (97). Additionally, when there is a conflict the incoming flow I_c must be shared among the transitions in the conflict (98).

	Basic logical sentences	*Linear constraints*
(88)	$\left[\sum_{i=1}^{s} pcv_{k,j} \leq I_c\right] \leftrightarrow [\lambda_{5,k,j} = 0]$	(63) and (64)
(89)	$[m_c > 0] \leftrightarrow [\lambda_{6,k,c} = 0]$	(65) and (66)
(90)	$[\lambda_{7,k,c} = 1] \leftrightarrow [\lambda_{5,k,j} = 1] \wedge [\lambda_{6,k,c} = 1]$	(67)-(69)
(91)	$v_{k,j} \leq pcv_{k,j}$	(70) and (71)
(92)	$[\lambda_{7,k,c} = 0] \rightarrow [v_{k,j} \geq pcv_{k,j}]$	
(93)	$[\lambda_{7,k,c} = 1] \rightarrow \left[\sum_{i=1}^{s} v_i = I_c\right]$	(72) and (73)

Table 5.14 Equivalent logical sentences for (59) and (60)

Table 5.15 summarizes the conversion for (61) into logical sentences. In (94) the equality $v_{k,i} = pcv_{k,i}$ is associated to a binary variable λ_8; (95) allows to associate a binary variable for each pair of transitions in the conflict; $\lambda_{10}=1$ means there exist at least one $[\alpha_h v_{k,h} \geq \alpha_j v_{k,j}]$; now only it remains (97), which means: if there is a conflict ($\lambda_7=1$) and the transition is taken the flow previously calculated ($v_{k,i} = pcv_{k,i}$), then there exist at least one transition taking more flow than it is allowed ($\alpha_h v_{k,h} \geq \alpha_j v_{k,j}$).

	Basic logical sentences	*Linear constraints*
(94)	$[\lambda_{8,k,c} = 1] \leftrightarrow [v_{k,i} = pcv_{k,i}]$	(74) and (75)
(95)	$[\lambda_{9,k,i,p} = 1] \leftrightarrow [\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} \leq 0]$; *where* $T_i \neq T_p$	(76) and (77)
(96)	$[\lambda_{10,k,c} = 1] \leftrightarrow \left[\sum_{q=1}^{\frac{s}{2}C} \lambda_{9,k,i,p} \geq 1\right]$	(78) and (79)
(97)	$[\lambda_{7,k,c} = 1] \wedge [\lambda_{8,k,c} = 1] \rightarrow [\lambda_{10,k,c} = 1]$	(80)

Table 5.15 Equivalent logical sentences for (61)

In Table 5.16 it can be found three equivalent logical sentences for (62). In (98) a true value is assigned to λ_{11} when the sharing policy applies to two transitions in the conflict. In (99) λ_{12} is true when the *ifs* for a transition is lower than the previously calculated. Finally, in (100) if there is a conflict ($\lambda_7=1$) and two transitions are taking less flow than the previously calculated ($\lambda_{12}=1$) then its flows must accomplish with the sharing policy established.

Basic logical sentences		Linear constraints
(98)	$[\lambda_{11,k,i,p} = 1] \leftrightarrow [\alpha_i \cdot v_{k,i} - \alpha_p \cdot v_{k,p} = 0]$; *where* $T_i \neq T_p$	(81)-(84)
(99)	$[\lambda_{12,k,i} = 1] \leftrightarrow [v_{k,i} < pcv_{k,i}]$	(85)(86)
(100)	$[\lambda_{7,k,c} = 1] \wedge [\lambda_{12,k,i} = 1] \wedge [\lambda_{12,k,p} = 1] \rightarrow [\lambda_{11,k,i,p} = 1]$	(87)

Table 5.16 Equivalent logical sentences for (62)

5.3.4.1. Examples

In the model of Example 7 (Figure 5.7), the set of constraints (12), (13), (16) -(22), (56)-(62) and optimization function (23) are considered. The resolution rule [T_4,T_5,T_6] is applied. In the first IB-state, T_1, T_2, and T_3 go at their *mfs*, then I_5=9 must be shared among T_4, T_5 and T_6, but there are other considerations: T_4 is limited by I_4=2 and T_5 is limited by its own *mfs* (V_5=1), which causes v_4=2 v_5=1 and the remaining flow from I_5 v_6=6; at the end of the IB-state P_3 becomes zero and P_6 has increased its marking. In the second IB-state, these events disable T_3 and the vector *ifs* for the transitions in the conflict remain unchanged. In the third IB-state, P_1 and P_6 become empty, which disables T_1, T_3, T_4 and T_6; only T_2 and T_5 goes at their *mfs*. At the beginning of the last IB-state only P_5 is marked, T_5 is strongly enabled (v_5=1), and all the other transitions are disabled; this period ends after 40 time units.

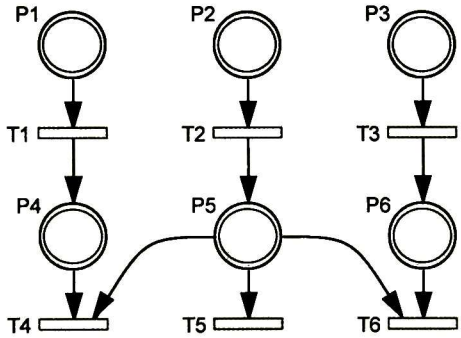


Figure 5.7. TCPN with a conflict and resolution by sharing

Initial data	Results			
	IB-S	Δt	v	m
Example 7 (0.0780)				
M_0=[10 90 30 0 0 0]	1	3	[2 9 10 2 1 6]	[4 63 0 0 0 12]
V=[2 9 10 3 1 9]	2	2	[2 9 0 2 1 6]	[0 45 0 0 0 0]
tC=50, sE=1, bE=50, mM= 90, K=4	3	5	[0 9 0 0 1 0]	[0 0 0 0 40 0]
	4	40	[0 0 0 0 1 0]	[0 0 0 0 0 0]

Table 5.17. Example for Figure 5.8

Example 8 (Figure 5.8) deals with a TCPN that includes a circuit in its structure; this case it is not considered by the algorithms proposed in the related works presented in Section 1.2.3. This is a net where the resolution of the conflict (<{P_2},{T_4,T_5,T_6}>) depends on the computation of some other transitions (T_1,T_2,T_3) and this calculation, at the same time, depends on the computation of the *ifs* for the transitions in the conflict. This behavior is calculated using the same set of constraints as in Example 6. The flow propagation in this net is as follows. At the beginning only P_5 is marked at this

point, then only T_5 can be fired at 4 (its *mfs*), P_1, P_2 and P_3 are fed, then T_1, T_2 and T_3 can be fired at v_1=3, v_2=4 and v_3=3 respectively; now P_4 and P_6 are fed at 3, allowing the firing of T_4 and T_6. Now, there is not actual conflict T_4, T_5 and T_6 go at the *mfs*; with these flows T_2 can go at its *mfs* 6. This final result it is correctly calculated by the MLIP model. After 1 time unit, P_5 becomes empty and then there is a conflict; the places marked are P_1,P_2 and P_3, so v_1, v_2 and v_3 go at the *mfs*, and then the flow I_5 must be shared among the transitions in the conflict, thus v_4=v_5=v_6=2. At the end of the IB-state, P_1 and P_3 become empty and P_4 and P_6 become marked; so in the following IB-state, the transitions in the conflict again share the incoming flow I_5=6 (v_4=v_5=v_6=2); now T_1 and T_3 are weakly enabled, so its *ifs* depend on I_1 and I_3 respectively, thus v_1=v_3=2. As it can be noticed the fourth IB-state presents the same values for *v* and *m*; this means that the TCPN has reached an equilibrium state.

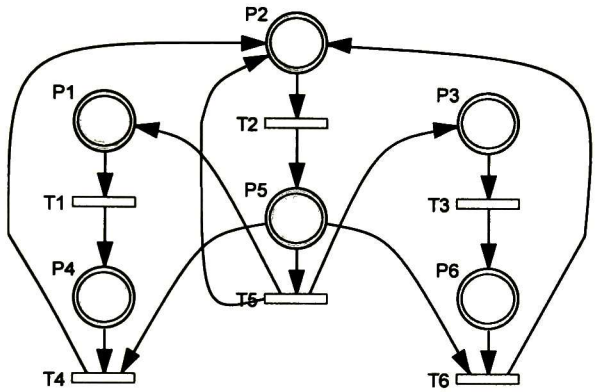


Figure 5.8. TCPN with a conflict and resolution by sharing

Initial data	*Results*			
	IB-S	*Δt*	*v*	*m*
Example 8 (0.3580)				
M_0=[0 0 0 0 4 0]	1	1	[3 6 3 3 4 3]	[1 4 1 0 0 0]
V=[3 6 3 3 4 3]	2	1	[3 6 3 2 2 2]	[0 4 0 1 0 1]
tC=10, sE=1, bE=10, mM= 5, K=4	3	6	[2 6 2 2 2 2]	[0 4 0 1 0 1]
	4	2	[2 6 2 2 2 2]	[0 4 0 1 0 1]

Table 5.18. Example for Figure 5.8

Chapter 6

SIMULATION OF NETWORKED AGENT SYSTEMS MODELS

Abstract. This chapter presents a scheme for executing the model of a networked agent system modeled as a THPN. It is based on the modeling presented in Chapter 3, 4 and also in the MLIP solution presented in Chapter 5. At the end, a study case about distributed energy household networks is presented.

6.1. Overview

Modeling a networked agent system as a THPN can result in a huge net; thus performing the calculation of the *ifs* for the continuous transitions is a complex task whose computational time increases exponentially with the number of places and transitions. The herein proposed scheme allows developing systematically the necessary software to simulate a networked agent system modeled as a THPN. This scheme is composed by two main parts; the first one is the computation of *ifs* for all possible macro-markings in a THPN, which is implemented in Matlab using the solver Cplex. The second one is a procedure to compute the agent's IB-states implemented in Java using the development framework JADE.

This simulation is based on the assumption that all the agents are identical; consequently, the complete THPN model will result in several identical structures interconnected. Then the calculation of the *ifs* can be performed over the THPN model of one agent, and later considering the agent interactions; thus the number of equations to solve the state's networked agent system will be reduced.

Several considerations must be taken into account for creating a simpler way to compute the state of the agent's system:

- Referring to the networked agent system:
  - The number of agents is fixed
  - The agent network topology is known and does not change during the system's operation
  - The continuous interactions between agents are performed only in one direction, meanwhile the discrete interaction can be in both ways
- Regarding the agents:
  - They are identical
  - In the THPN, the continuous part is ruled by linear equations and the discrete part enables/disables the continuous part
- and regarding the Petri net:
  - The Petri net structure is a simple Petri net, it means that each transition can only be concerned by one conflict at the most
  - The discrete part is 1-bounded

6.2. General description system

This section presents a general description of the proposed scheme, starting with the explanation of the main functional units, following with the detailed activities performed by these units, the class design for the Java part, and a small example to illustrate a particular implementation, where it can

be seen from the modeling part, the input information to the system, the generated *ifs* table for an agent, and the agent action and interaction with other agents to perform the simulation.

6.2.1. Functional units

As mentioned before, the general objective of the scheme is to perform the simulation of a THPN model as a networked agent system; the input data includes only THPN matrix information; then a csv file with the IB-State information for the complete system is obtained.

The use case diagram from Figure 6.1 shows the main functional units of the system. It can be divided in two main parts: one of them, represented at the leftside, is performed in Matlab environment (Instantaneous Firing Speed for macro-makings) and the other one in Java (THPN Simulation).

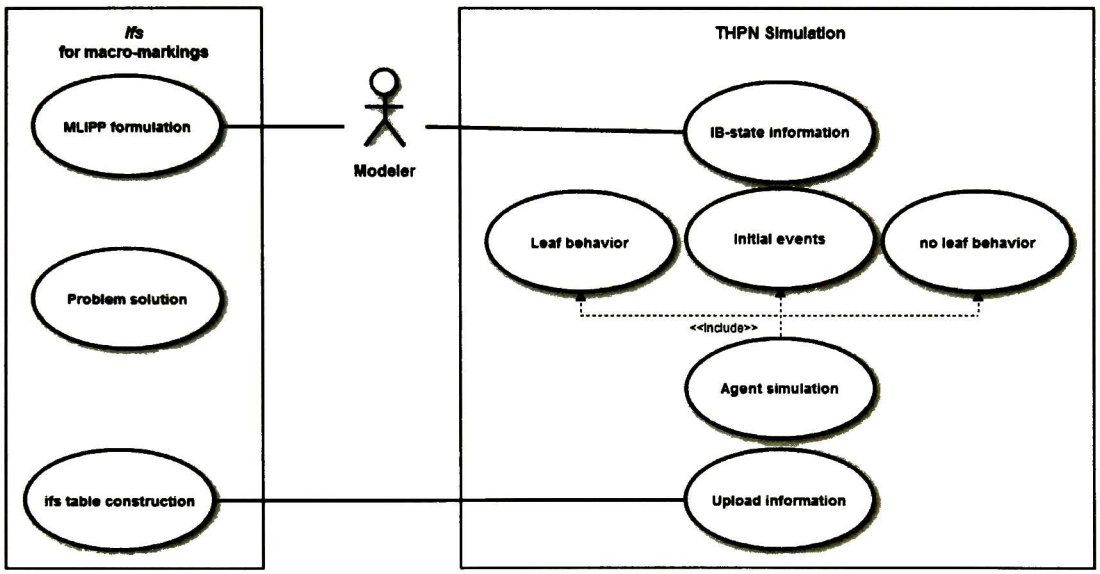


Figure 6.1. Use case diagram

In order to perform an agent networked system simulation, the modeler can use the methodology from Chapter 3 and 4 to represent the system as a THPN; this information is uploaded into the simulation environment. The *ifs* for macro-markings perform the functions described in Table 6.1 to obtain an *ifs* table, this table contains all the possible *ifs* for a THPN.

Use Case	***Description***
MLIPP formulation	With *Pre* and *Post* matrix, M_0, maximal firing speeds V, and policy resolution data given by the modeler, the system can construct a set of constraints according to the structural characteristics of the net.
Problem solution	Using CPLEX solver, the solution from the MLIPP is found.
ifs table construction	With the solution given, the *ifs* table is stored in a csv file.

Table 6.1. Functional units from *ifs* for macro-markings

The simulation of each agent can be performed through a Java program. Table 6.2 describes the necessary functions to run this simulation.

Use Case	*Description*
Upload information	In order to perform a simulation, the following information is needed: *ifs* table, *Pre* and *Post* (Representing one generic agent), *WA = Pre-Post*, *WR* (incidence matrix that represents agent communication), dTiming (timing associated to each discrete transition), M_0 (Initial marking), the hybrid function and also the simulation time. This information is represented in an internal form.
Agent simulation	All the necessary agents are created; each one of them is going to perform a part of the simulation for the agent system. At the beginning, all of them receive the same input information (since they have the same structure), but the initial marking; this can lead the agents behaving differently, in despite of be identical. Each agent can operate in two ways according to its operation mode (Leaf or no leaf). In both operation modes they deliver at the end of simulation the IB-state information: initial marking, *ifs*, and the IB-state duration *dt*.
Initial events	This process is executed at the beginning for all agents; it performs initial immediate discrete events.
Leaf behavior	This behavior is performed by agents who do not receive input communication; they can independently obtain its own IB-state information.
No leaf behavior	This other behavior is performed by agents receiving input communication. They must always be listening for changes in the input flows from incoming neighbors.

Table 6.2. Functional Units from THPN simulation

6.2.2. Activities step by step

Figure 6.2 shows the general activity diagram of the complete software. This diagram details the activities and actions performed in each use case described before; the name of activities that appear underlined will be described later. The *ifs* macro-makings procedure performs the following steps: reads the input information, represents it in an internal form, creates the necessary information for each variable in the problem, constructs the equations, solves the problem, and finally the information is presented to the user in a csv file. For the THPN simulation, after the input information is read, all the agents are created, each agent perform the simulation of its part, and when all the agents are finish, the simulation ends, and the IB-state information is obtained.

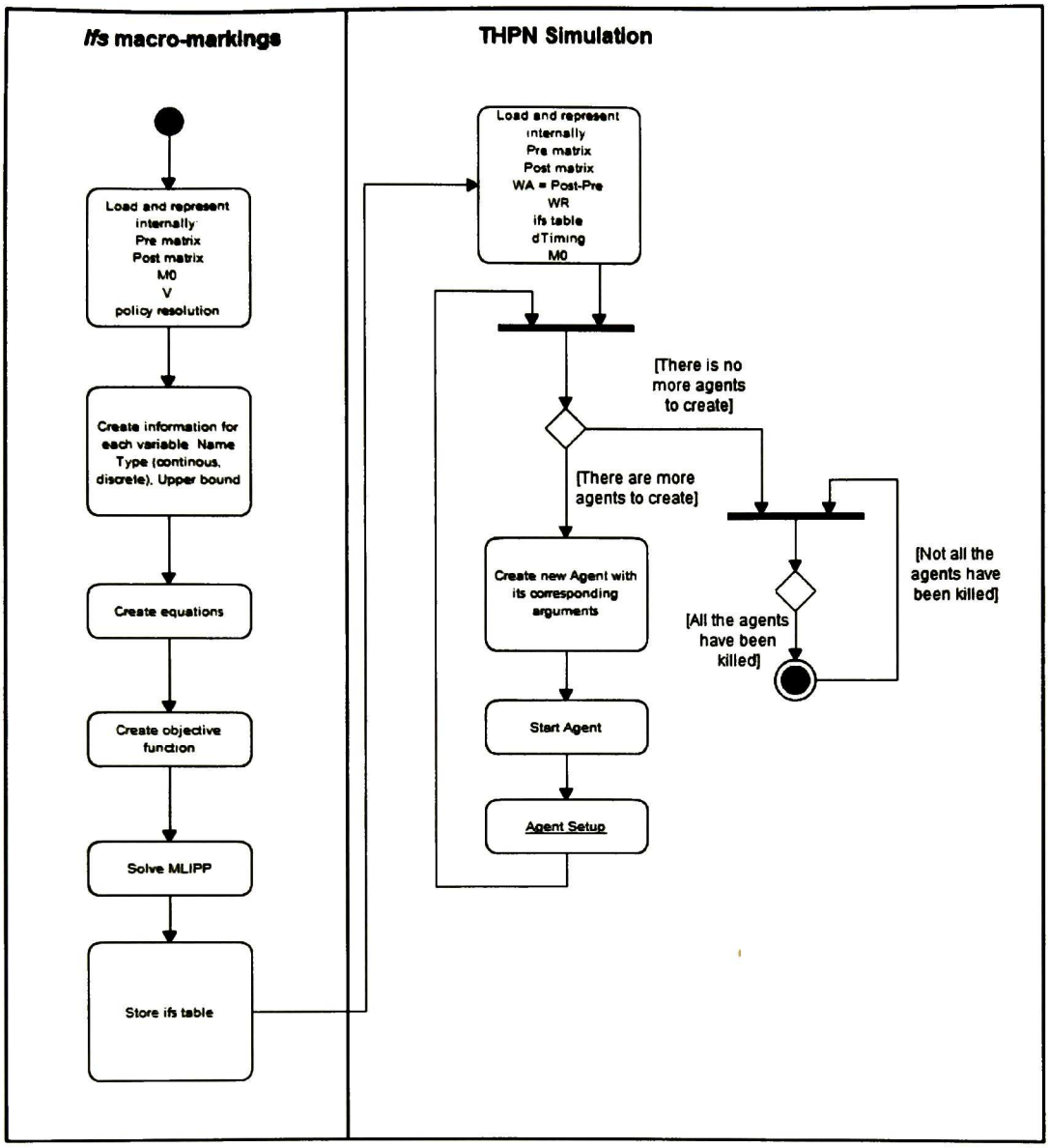


Figure 6.2. General activity diagram

The activity Agent Setup establishes agent's initializations; the activities performed in this part are shown in Figure 6.3.

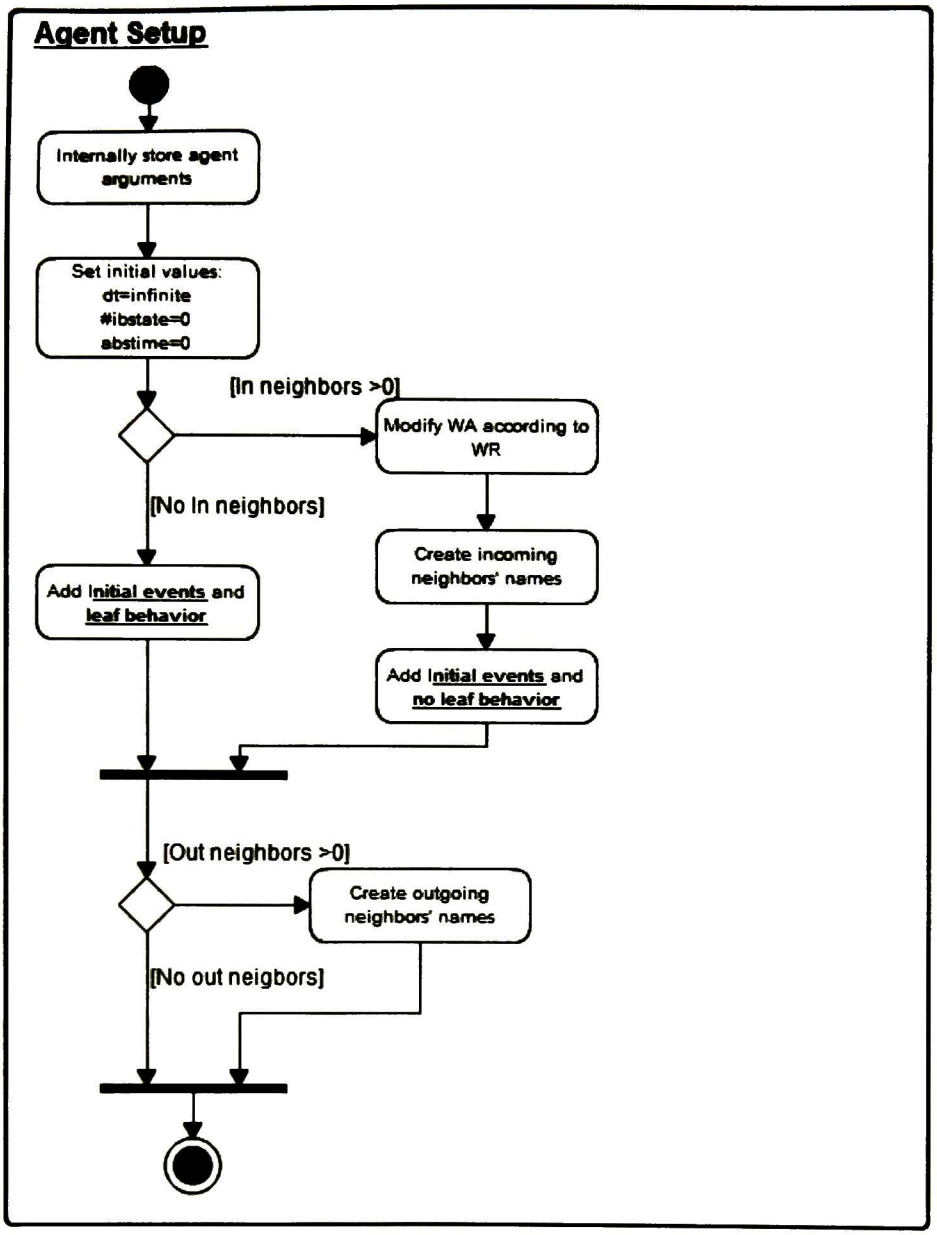


Figure 6.3. Setup Agent activity diagram

When an agent is created, at the beginning, all the information regarding to its THPN is stored in an internal representation, also the variables *dt*, *#ibstate* and *abstime* are set at their initial values, *abstime* makes reference to its own time. In the first IB-state when the information is still not calculated, the duration of the IB-state, is considered as infinite. For agents with incoming neighbors; it is necessary to modify *WA* matrix to add the corresponding incoming transitions, this information is obtained through *WR*. A list with all the incoming neighbors is created and also the procedures

Initial events and noleaf node behavior are set. For agents with no incoming neighbors the procedures Initial events and leaf behavior are added. At the end, if the agent has outgoing neighbors, a list with their names must be constructed for future communication.

The underlined procedures Initial events, leaf behavior and no leaf behavior are described in Figures 6.4, 6.5 and 6.6 respectively. Initial events performs any discrete event occurring in time 0 and send a message to all outgoing neighbors with discrete communication.

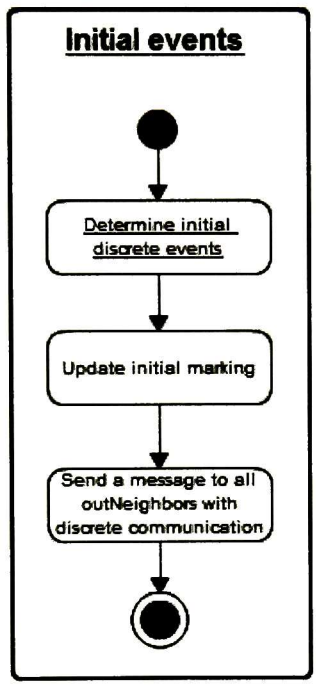


Figure 6.4. Initial events activity diagram

Leaf behavior is for those agents with no incoming neighbors. The main activities are made in a General Node Procedure, it performs the necessary steps to construct the information for the IB-state. After this procedure is done, if it is not the end of simulation, its time is updated (*abstime*) and the agent behavior is blocked until *dt* has passed, otherwise *dt* is updated to accomplish with the time of end of simulation, and the agent is killed.

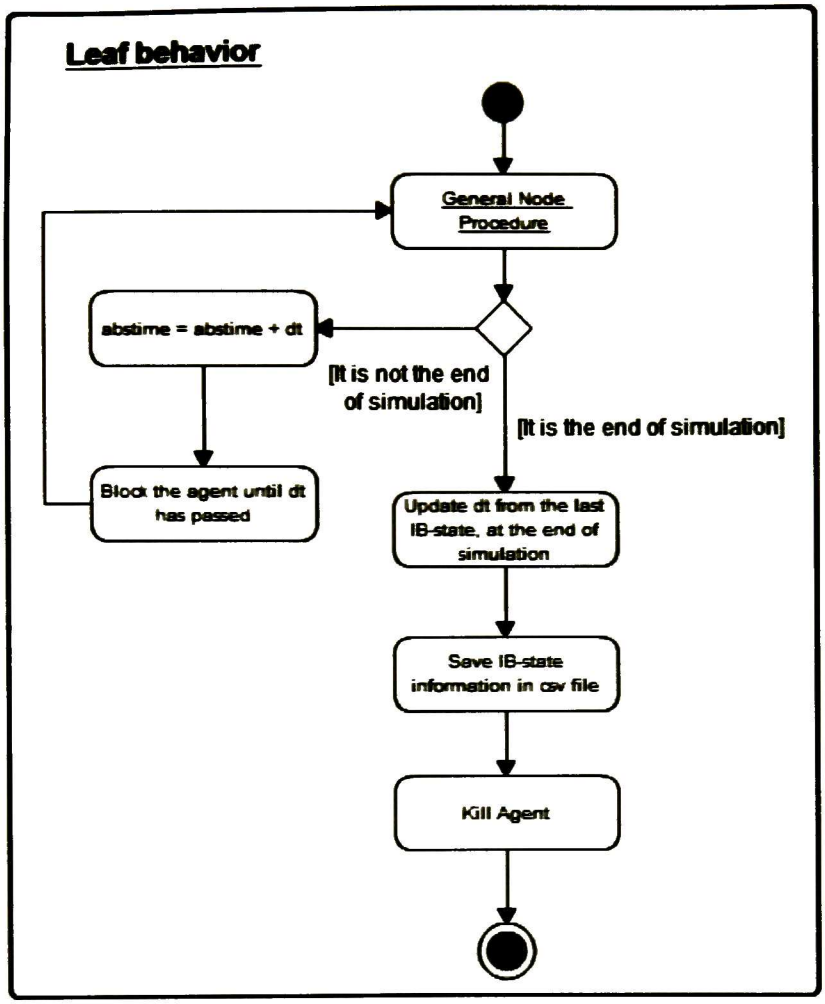


Figure 6.5. Leaf behavior activity diagram

No leaf behavior starts getting the incoming messages; all the messages are stored. For the first IB-state it is necessary to receive all the incoming messages (Discrete and continuous ones) in order to obtain IB-state information, when it does not occur, the agent is blocked until it receives a new message. If it is not the first IB-state or all the incoming flows were received, then the General Node Procedure can be performed; if it is not the end of simulation the agent is blocked until it receives a message or *dt* has passed. Otherwise, as stated for leaf behavior, *dt* is updated to the time of end of simulation, and the agent is killed.

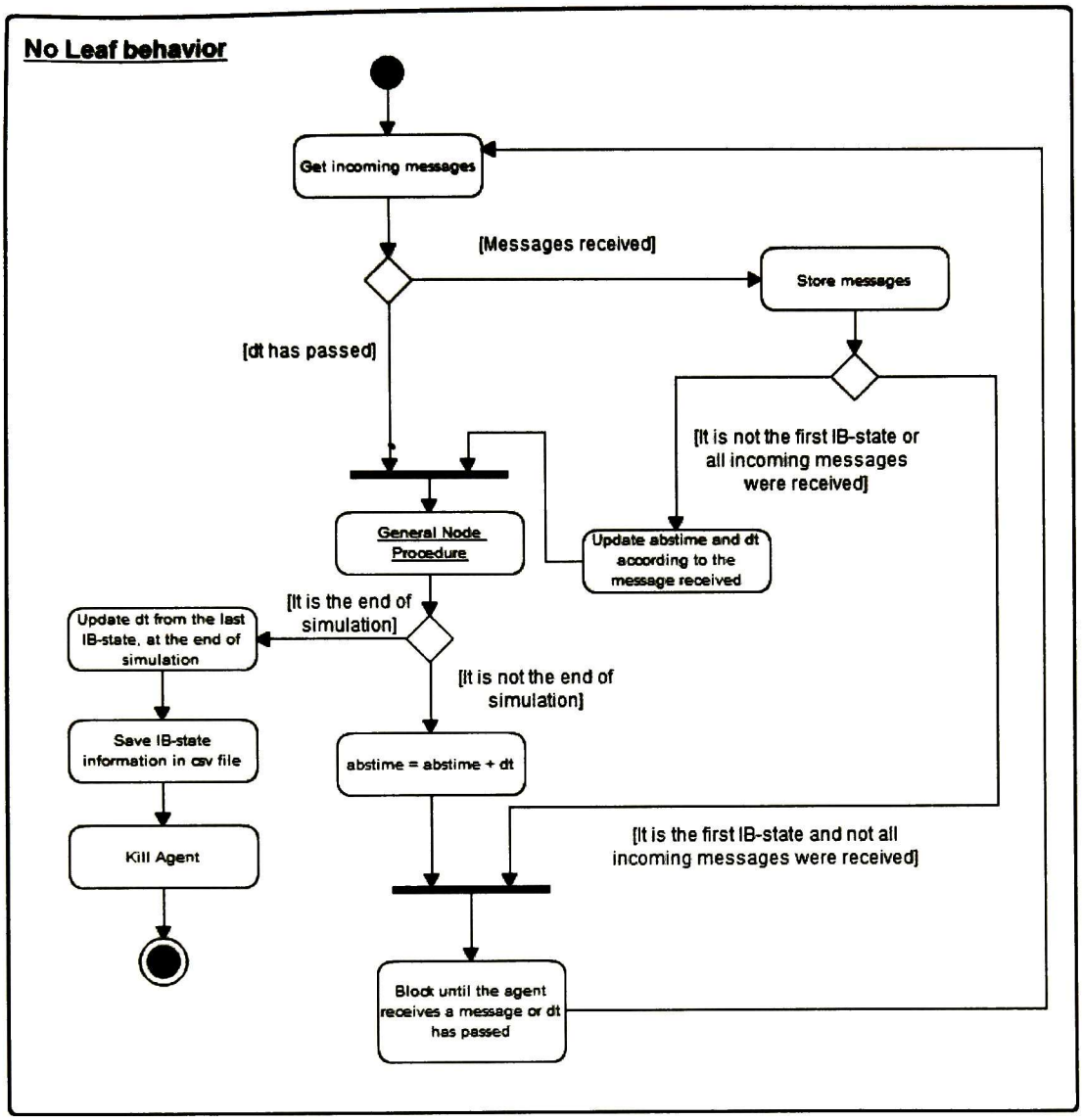


Figure 96. No Leaf behavior activity diagram

Notice that both behaviors, leaf and no leaf, share a common procedure: General Node Procedure. Figure 6.7 presents its steps. First the marking at the current IB-state is calculated, the last IB-state information is used to do that (marking, *ifs* and discrete events); of course, if it is the first IB-state, the initial marking is considered; from here, *ifs* can be obtained. Now, it is necessary to determine the following discrete and continuous events and then *dt* is set to the value of the minimum event. Finally, if the agent has outgoing neighbors and there is a change to communicate, like a change in the outgoing flow from a continuous transition or the firing of a discrete transition, then a message is sent to the outgoing neighbors; otherwise the number of IB-state is increased.

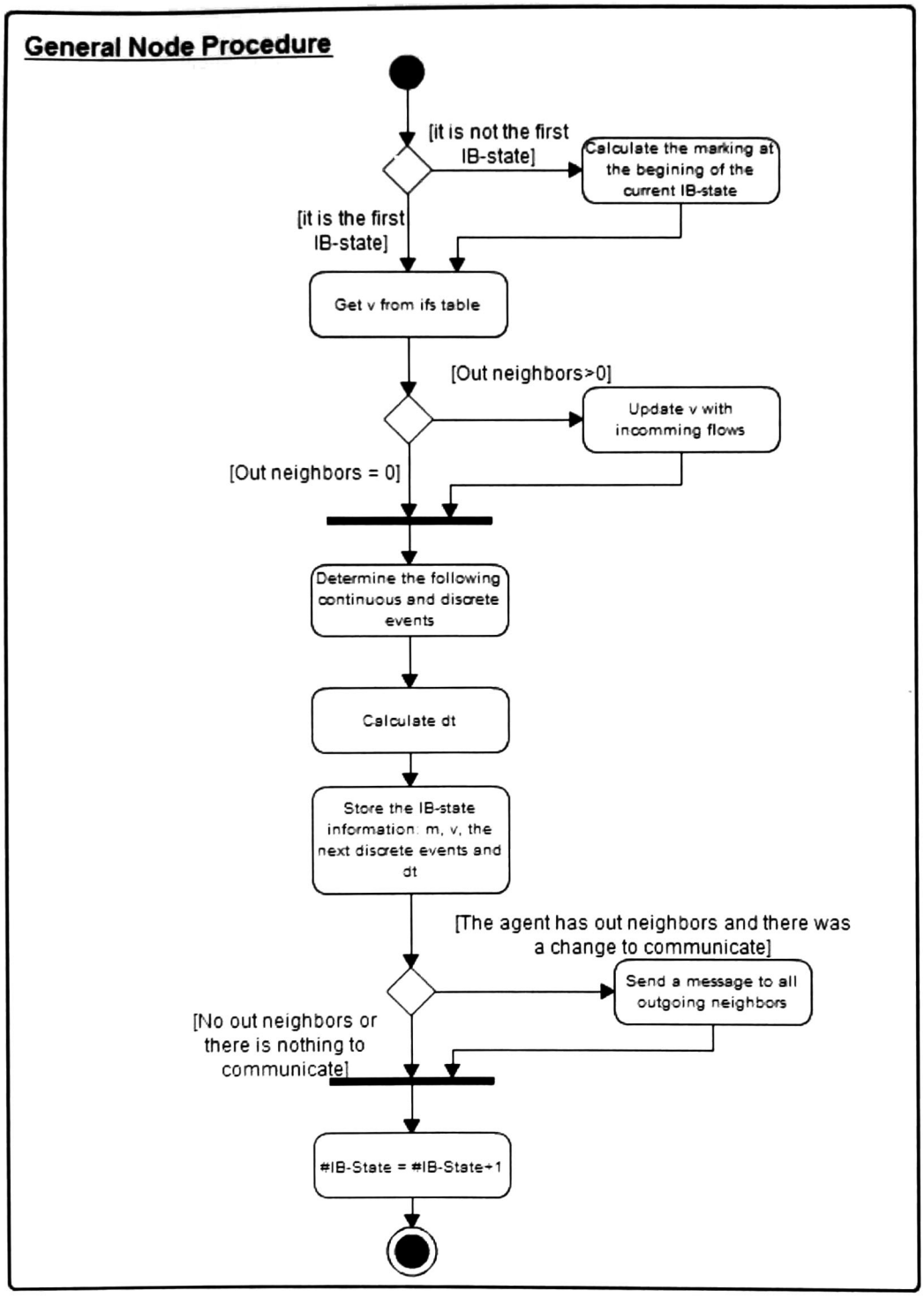


Figure 107. General node procedure activity diagram

6.2.3. Class diagram

Figure 6.8 shows the class diagram including the classes implemented for the THPN simulation. The main class *Init* begins with the simulation, *InputDataGui* provides a user interface to introduce the routing information, *Simulation* performs the agents simulation by creating the *ThpnAgents* through class *AgentContainer*. This class includes: *initialEvents*, *leafNode* and *noleafNode*. For further information about these classes and its methods please refer to Appendix A at the end of this document.

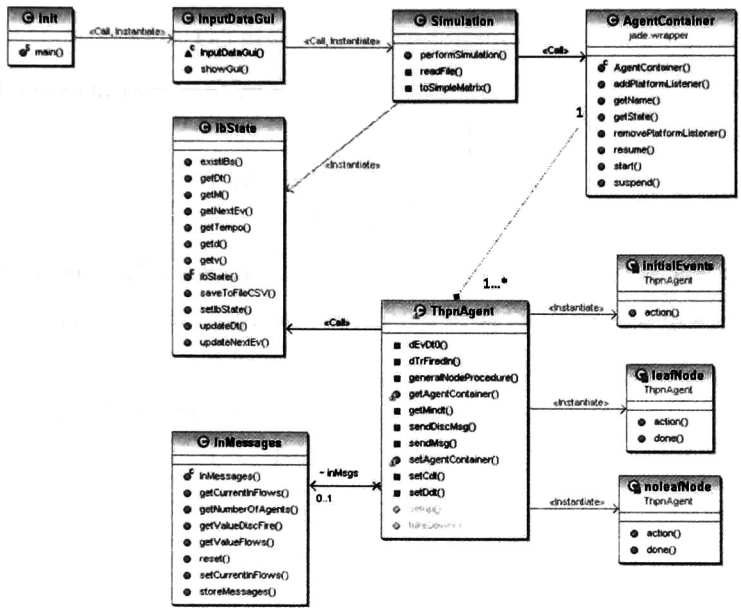
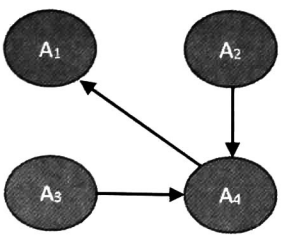


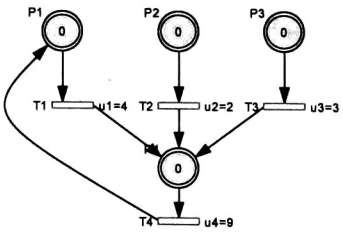
Figure 6.8. Class diagram for THPN simulation

6.2.4. Illustrative example

This section presents a simple example to explain in detail the procedure described before. This agent network is composed by four identical agents and its topology is shown in Figure 6.9(a). The internal dynamics of each agent is represented by the TCPN in Figure 6.9b.



(a) Agent network topology representation



(b) TCPN agent

Figure 6.9. Networked Agent system

Formally this networked agent system can be described by the directed graph G = {{A_1,A_2,A_3, A_4},{(A_2,A_4), (A_3,A_4),(A_4,A_1)}}. Figure 6.10 shows the connection nodes between the agents.

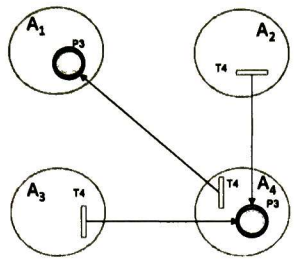


Figure 6.10. Networked Agent system

This configuration can be expressed by Matrix *WR*:

$$WR = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

In this example one place (P_3) and one transition (T_4) of every agent model will receive/transmit flow from/to other(s) agent(s). Each column corresponds to T_4 and each row to P_3 from each agent respectively.

The incidence matrix WA^i for one isolated agent is:

$$WA^i = \begin{bmatrix} -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 1 & -1 \end{bmatrix}$$

The perception and communication parts are unique, because of this, only one kind of message is necessary $<A_i,A_j,abstime, v_4>$. The complete networked agent system modeled with TCPN is shown in Figure 6.11.

Once the system is modeled, it is necessary to compute all the possible *ifs* for each macro-marking. For this, consider a TCPN generic agent. This piece of net must represent any agent in the networked system. In order to determine all the *ifs* for the possible macro-markings, it is necessary to know all possible incoming flows. The agent that receives more incoming flows is A_4, so this agent will serve as generic agent (Figure 6.12). The transitions I_1 and I_2 will represent all the possible incoming flows that an agent of this net can receive.

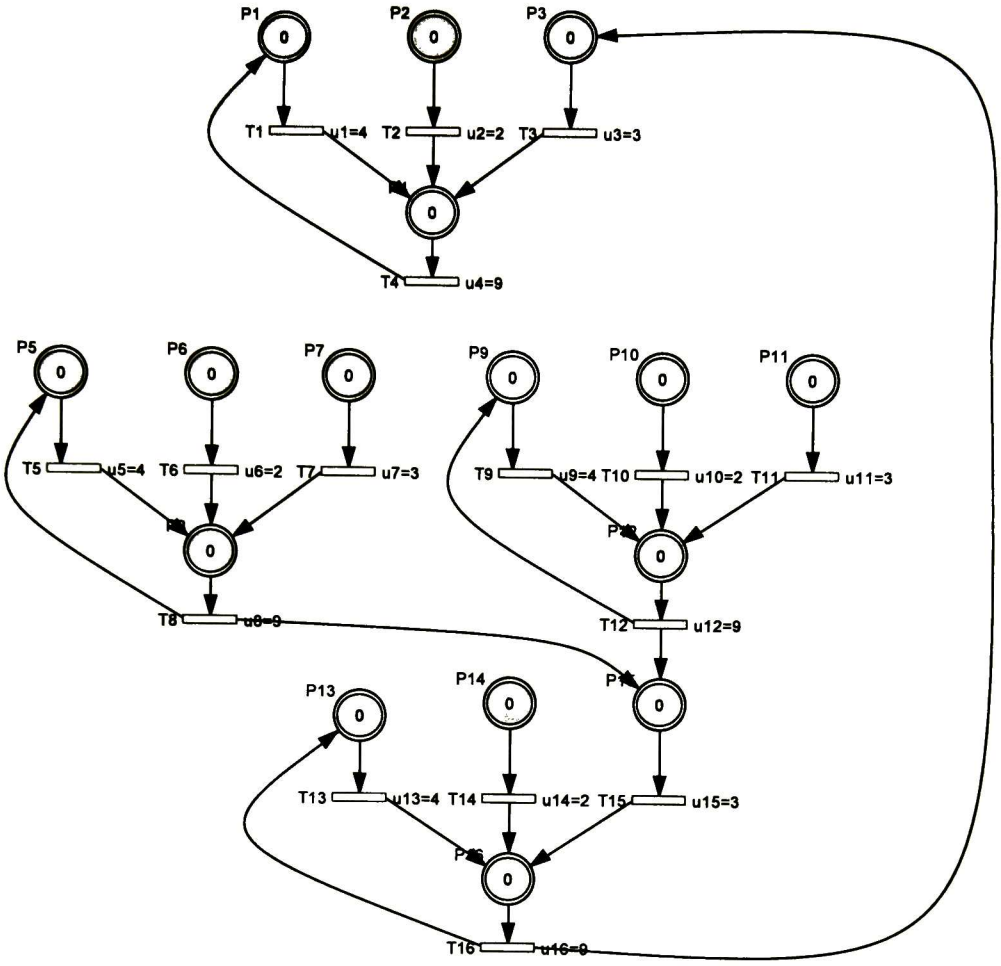


Figure 6.11. Networked Agent system modeled with TCPN

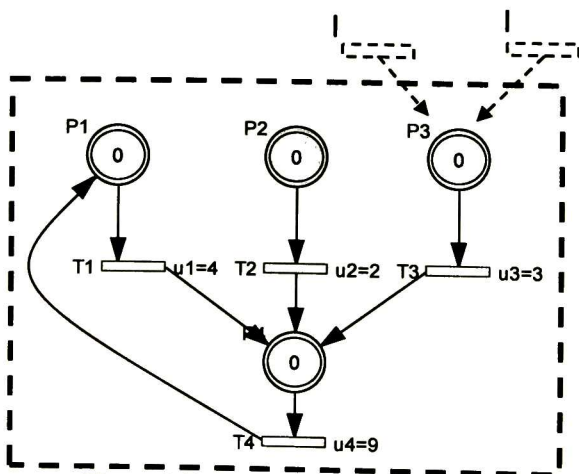


Figure 6.12. TCPN generic agent

Let us suppose $V=[4\ 2\ 3\ 9]^T$ and the following initial marking

- $mA_1=[10\ 0\ 0\ 50]^T$
- $mA_2=[0\ 20\ 0\ 0]^T$
- $mA_3=[0\ 10\ 0\ 0]^T$
- $mA_4=[0\ 0\ 0\ 0]^T$

The sequence for simulating this scenario will be presented for clarity in several sequence diagrams. Figure 6.13 shows the interaction with *ifs* macro-markings, to obtain *ifs* table.

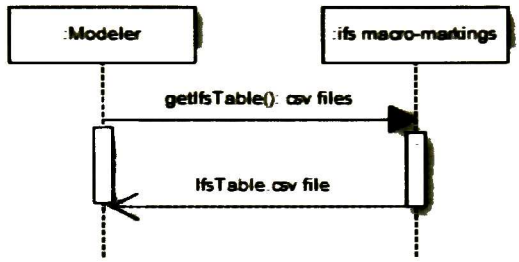


Figure 6.13. *Ifs* macro-marking sequence diagram

This piece of net has not conflicts so basic constraints *(12)-(22)* can be selected to obtain Table 6.3. The first 16 rows consider the incoming flow 0. From here, now it is possible to know all the values for v_4 (the outgoing flow, that at the same time will become the incoming flow for other agents), these possible values can be 0, 4, 6, 7 and 9. A new computation is needed, considering all the 2-combination, but in this particular case considering only $I_1=4$, $I_2=0$ and $I_1=0$, $I_2=4$ is enough to cover all the cases because $V_3=3$, and any incoming flow bigger than 3 will not generate a different table than the one presented in the rows 17-32.

#	*Macro-marking*				*Incoming Flow*	*ifs*				*Outgoing Flow*
	P_1	P_2	P_3	P_4	$I_1 + I_2$	v_1	v_2	v_3	v_4	v_4
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	1	0	4	0	0	9	9
3	0	0	1	0	0	4	0	3	7	7
4	0	0	1	1	0	4	0	3	9	9
5	0	1	0	0	0	4	2	0	6	6
6	0	1	0	1	0	4	2	0	9	9
7	0	1	1	0	0	4	2	3	9	9
8	0	1	1	1	0	4	2	3	9	9
9	1	0	0	0	0	4	0	0	4	4
10	1	0	0	1	0	4	0	0	9	9
11	1	0	1	0	0	4	0	3	7	7
12	1	0	1	1	0	4	0	3	9	9
13	1	1	0	0	0	4	2	0	6	6
14	1	1	0	1	0	4	2	0	9	9
15	1	1	1	0	0	4	2	3	9	9
16	1	1	1	1	0	4	2	3	9	9
17	0	0	0	0	4	4	0	3	7	7
18	0	0	0	1	4	4	0	3	9	9
19	0	0	1	0	4	4	0	3	7	7
20	0	0	1	1	4	4	0	3	9	9

21	0	1	0	0	4	4	2	3	9	9
22	0	1	0	1	4	4	2	3	9	9
23	0	1	1	0	4	4	2	3	9	9
24	0	1	1	1	4	4	2	3	9	9
25	1	0	0	0	4	4	0	3	7	7
26	1	0	0	1	4	4	0	3	9	9
27	1	0	1	0	4	4	0	3	7	7
28	1	0	1	1	4	4	0	3	9	9
29	1	1	0	0	4	4	2	3	9	9
30	1	1	0	1	4	4	2	3	9	9
31	1	1	1	0	4	4	2	3	9	9
32	1	1	1	1	4	4	2	3	9	9

Table 6.3. *ifs* Table

Once the *ifs* table is obtained, the THPN simulation can be performed. Figure 6.14 shows the first steps; the initial procedure Init begins creating an user interface usrInterface to obtain the location of the files: *Pre*, *Post*, *WA*, *WR*, M_0, *ifs* table, dTiming, and the simulation time. Now the simulation can be performed through the object sim. An object ibs from the class ibState is created to store the IB-state information; the agents are created through the object mc (the agentController). For this process each agent is created and startedby the object sim; the controller agent manages the interaction with the objects Agents of the class ThpnAgent ("Create new agent" and "Start Agent"); this process is not specifically described here because class AgentController is provided by JADE; for further information the reader can consult JADE's documentation [JADE,2014].

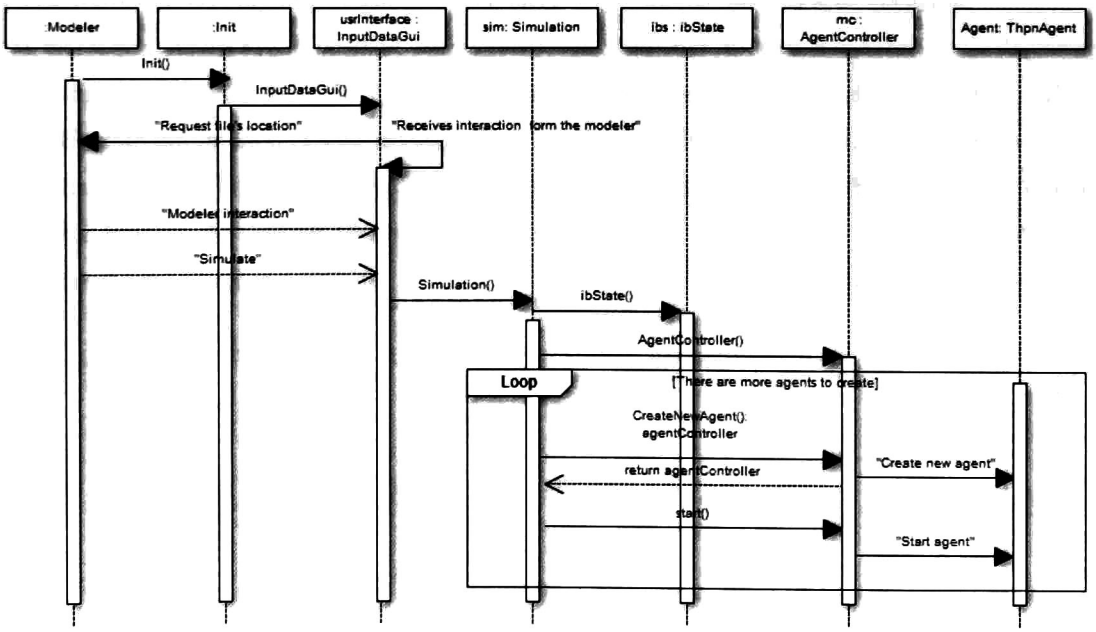


Figure 6.14. *Initial steps THPN simulation* sequence diagram

For leaf nodes the agent must acquires leafNode behavior, otherwise it acquires noleafNode behavior and create an inMsg object (Figure 6.15). For this example, procedure InicialEvents is not performed; this is because the Petri net only contains continuous nodes. Following the example, leaf nodes are Agents 2 and 3; and no leaf nodes are Agents 1 and 4.

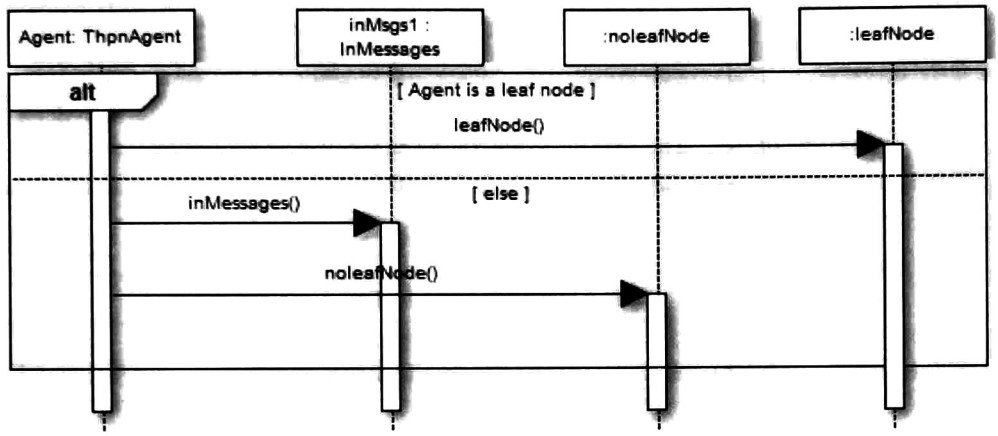


Figure 6.15. *Setup for agents* sequence diagram

Each leafNode performs the following sequence (Figure 6.16): In the first IB-state v is obtained, the next continuous event is calculated, the information is stored (*#ibstate*, Agent, marking, *dt*, *ifs*), and if the agent has outgoing neighbors, a message to each one of them is sent. As mentioned before the message contains $<A_i, A_j, abstime, outputflow>$. After the first IB-state, it is necessary to know the last information stored, then v is obtained again, the following continuous event is calculated, and the new IB-state information can be stored and sent to the outgoing neighbors as long as the outgoing flow has changed. This process is performed until the end of simulation or when the last IB-state is infinite. Finally, the last *dt* IB-state is updated at the end of simulation and all the IB-states information are stored in a csv file. The agent is killed and this is notified to the *AgentController*.

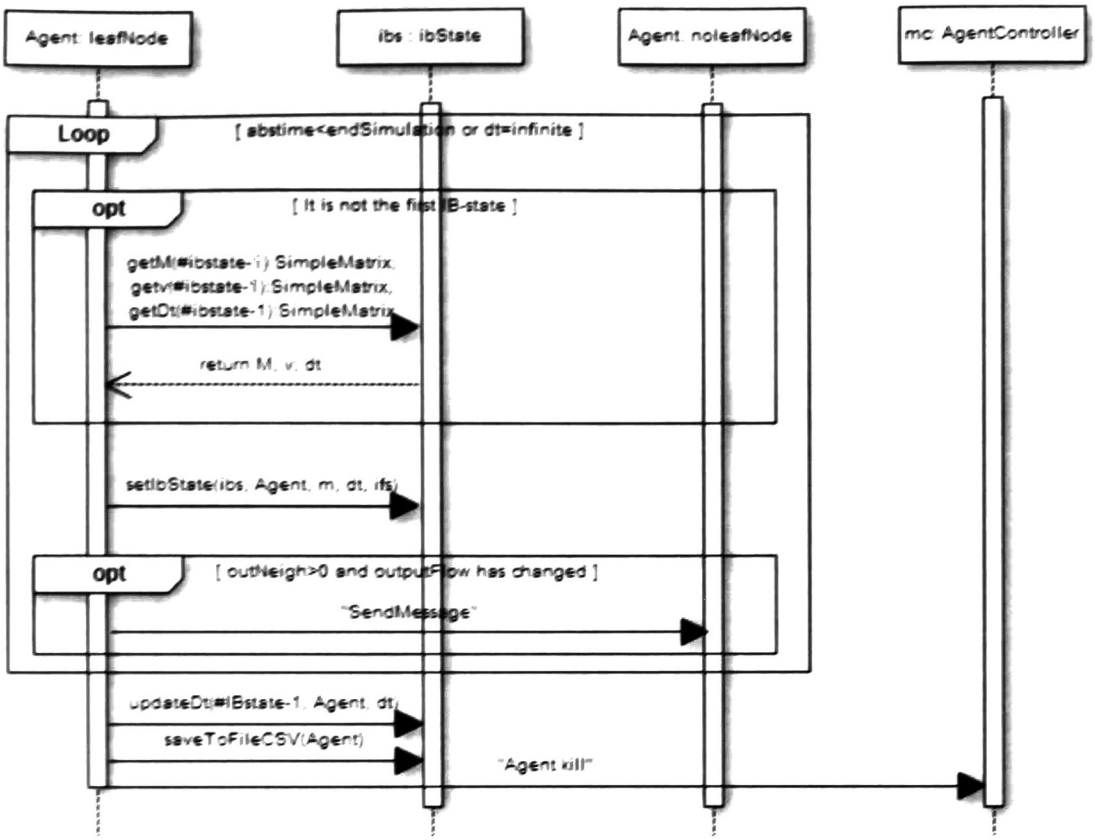


Figure 6.16. Action for leaf nodes sequence diagram

Figure 6.17 shows the sequence of actions of no LeafNodes. There is a loop and it is performed until the end of simulation. This loop is composed by two optional procedures: storage received messages, when a message is received; or determine IB-state information, when *dt* has passed.

In the first IB-state *dt* is set at infinite, because it is necessary to receive all the information from the incoming neighbors (Incoming flows or discrete fires). When all the incoming messages were received and stored, then *dt* is updated at 0 and the second procedure can be performed: v is obtained, the following continuous event is calculated, the information is stored (ibs, Agent, *m*, *dt*, *ifs*) and if the agent has outgoing neighbors, a message to each one of them is sent.

In a posterior IB-state if a message was received it is stored, the last IB-state is updated, because the end of this IB-state is going to be due to an external event, not because an internal event as previously calculated. Then the second procedure can be performed. At the end of simulation, the last *dt* IB-state is updated at the end of simulation and all the IB-states information are stored in a csv file. The agent is killed and this is notified to the AgentController.

In figures 6.18 and 6.19 the marking evolution of each place and the *ifs* evolution are shown. The data omitted is because it remains in zero.

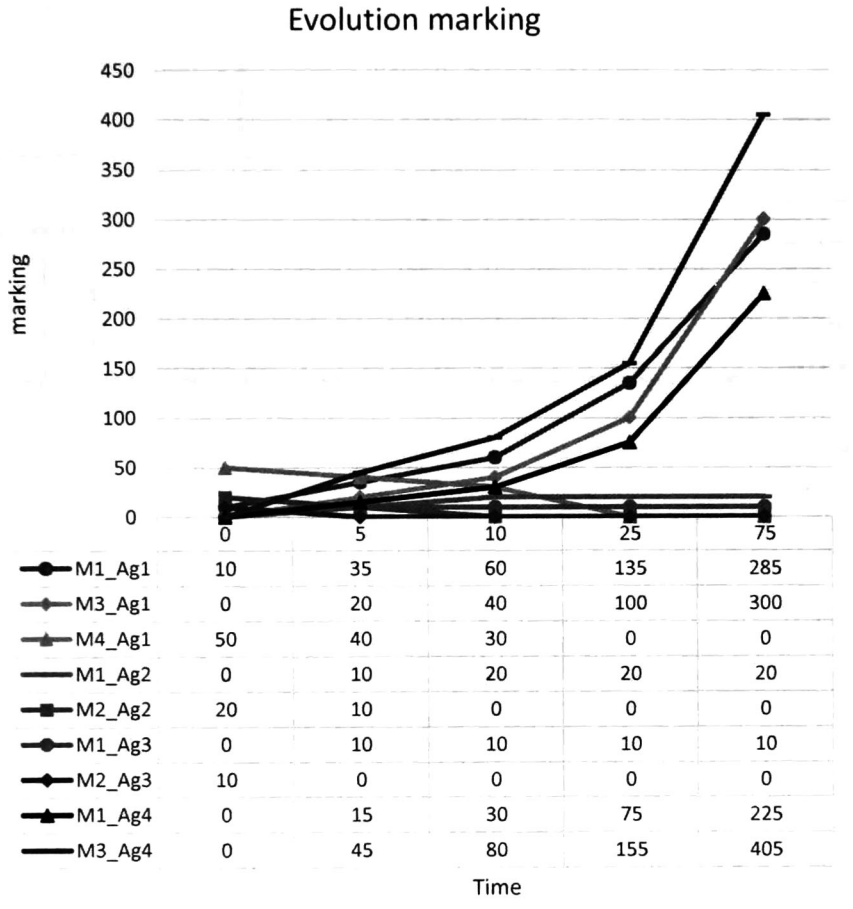


Figure 6.18. *Evolution marking*

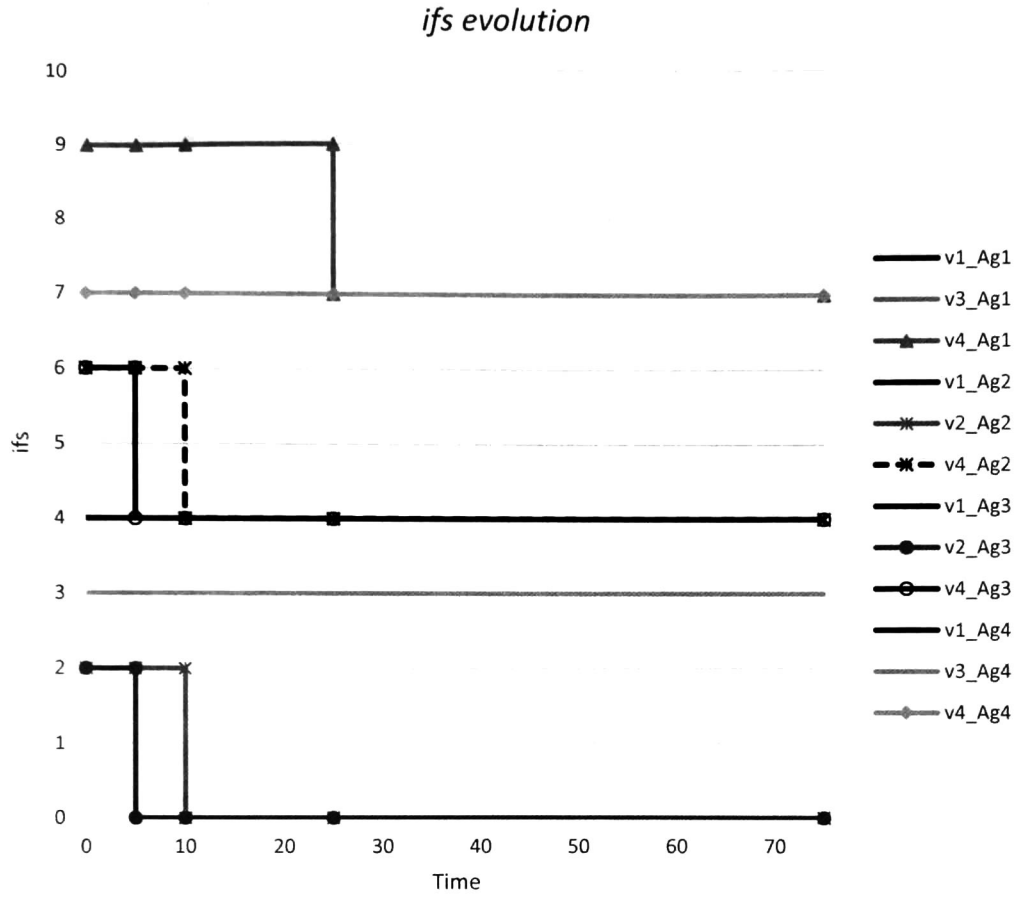


Figure 6.19. *ifs evolution*

6.3 Case study

In this section a networked agent system for simulating energy households is considered. This is academic example is based on the system functioning described in [Negenborn,2007]

The electricity systems of many countries are currently undergoing a process of transformation, due to market liberalization and environmental regulations. New technologies such as renewable energy and micro combined heat and power (μCHP) are emerging [Pehnt,2006]. In particular μCHPs allow becoming a household in a distributed energy resource, comprising power and heat generators with its corresponding storage units. Households can operate independently of energy suppliers, and they can buy and/or sell power among them.

Figure 6.20 shows an overview of the system under study. A μCHP can produce both electricity and usable thermal energy. This μCHP consists of a conversion unit 1 which converts gas into electricity and heat, and a conversion unit 2 acting as an auxiliary burner, producing only heat. Both

conversion units are equipped with built-in fixed controllers that are designed to keep the level of heat storage between predefined upper and lower bounds.

The electricity is stored in a battery and it can be used directly by the household or it can be sold; also it can be imported from another household. The generated heat is supplied to a heat storage unit in the form of hot water, which is used for the own household.

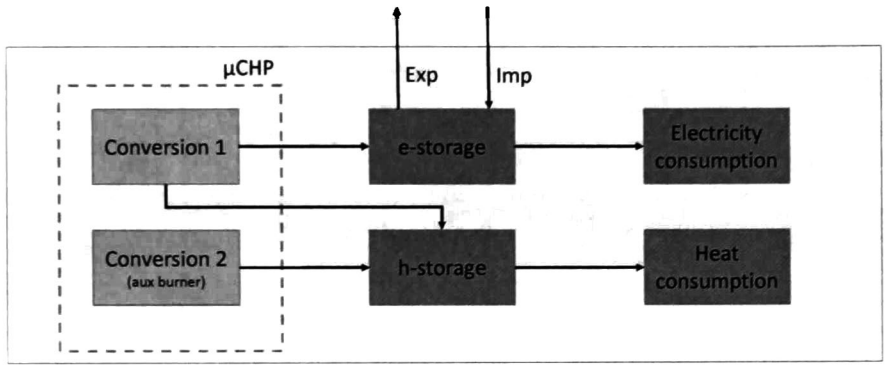


Figure 6.20. Conceptual overview of the system

The network under study consists of five households with the topology shown in Figure 6.21. Each one of them have a μCHP installed, this allows them produce, store, consume and interchange energy. The arrows indicate the allowed electricity flows between households, but also discrete communication is performed in both directions between them, for the electricity interchange.

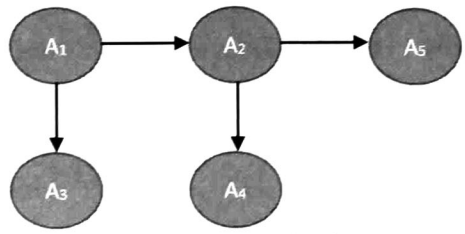


Figure 6.21. *Action for no leaf nodes* sequence diagram

The internal dynamics of each agent is modeled with the THPN shown in Figure 6.22; the named elements Td and Pd represent discrete nodes, and the elements Tc and Pc are the continuous ones. This THPN models the four functionalities: Production, storage, consumption, and interchange of energy between households.

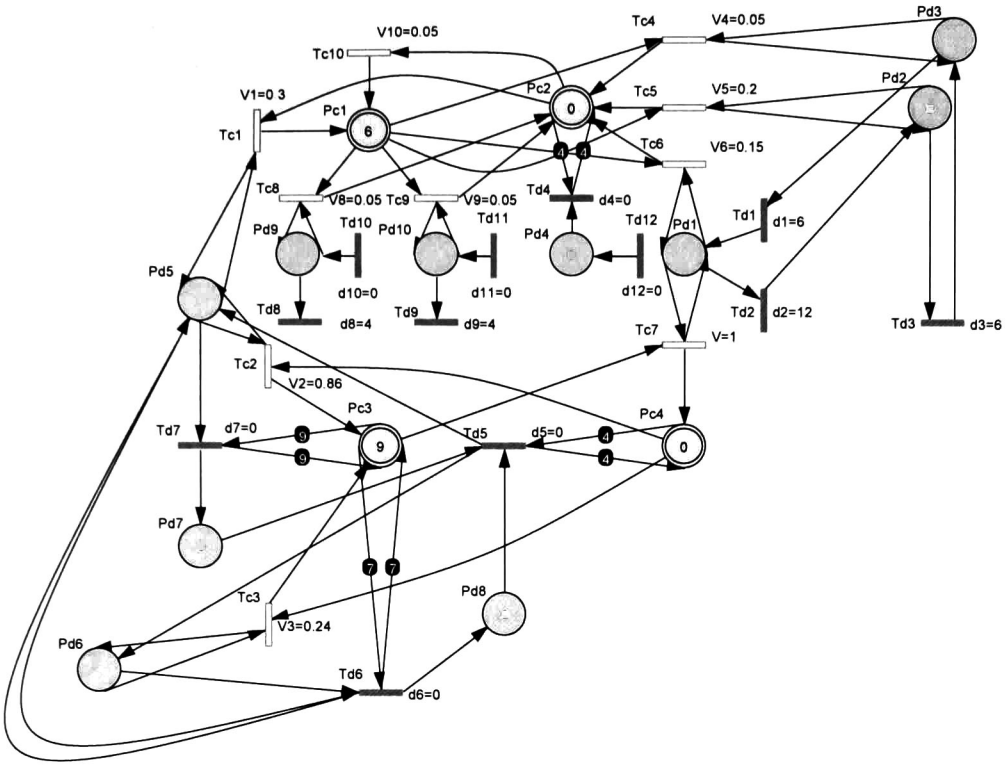


Figure 6.22 Action for no leaf nodes sequence diagram

Electricity production (Unit 1) is modeled by Tc_1, and the heat production by Tc_2 (Unit 1) and Tc_3 (Unit 2). The nodes Pd_7, Pd_8 and Td_5-Td_7, turn on and off units 1 and 2. When the heat storage is lower than 5, both units are turned on (Td_5 fires), when the level arrives at 7, unit 2 is turned off (Td_6 fires); and when it arrives at 9, unit 1 is now turned off (Td_7 fires).

Electricity storage is modeled by Pc_1 and Pc_2, these two places allow limiting the storage capacity (For this example it is 6); in the same way Pc_3 and Pc_4 model the heat storage (The heat limit capacity for this example is 9).

Three different type electricity consumptions are considered through the day: at afternoon Tc_4, at night Tc_6 and at morning Tc_5. For heat consumption, only one type of consumption is considered during the morning Tc_7. The discrete nodes Pd_1-Pd_3 and Td_1-Td_3, perform these changes.

Electricity can be exported using transitions Tc_8 and Tc_9, and imported through transition Tc_{10}. The discrete nodes Pd_4, Pd_9, Pd_{10} and Td_9-Td_{12}, allow interchanging energy. Figure 6.23 shows the interconnection of agents between these nodes; for every agent only the interacting nodes of their models are depicted.

The purchase request from a household is modeled with Pd_4 and Td_4; when the place is marked and the storage energy level is lower than 2, Td_4 is fired; this event marks the place Pd_9 or Pd_{10} from another agent. For example: firing Td_4 from Agent 2, marks Pd_9 in Agent 1 and Td_4 from Agent 3 marks Pd_{10} in Agent 1. These events allow exporting electricity, enabling Tc_8 and Tc_9 in Agent 1. After 4 units of time Td_9 and Td_8 are fired, marking the places Pd_4 in Agents 2 and 3.

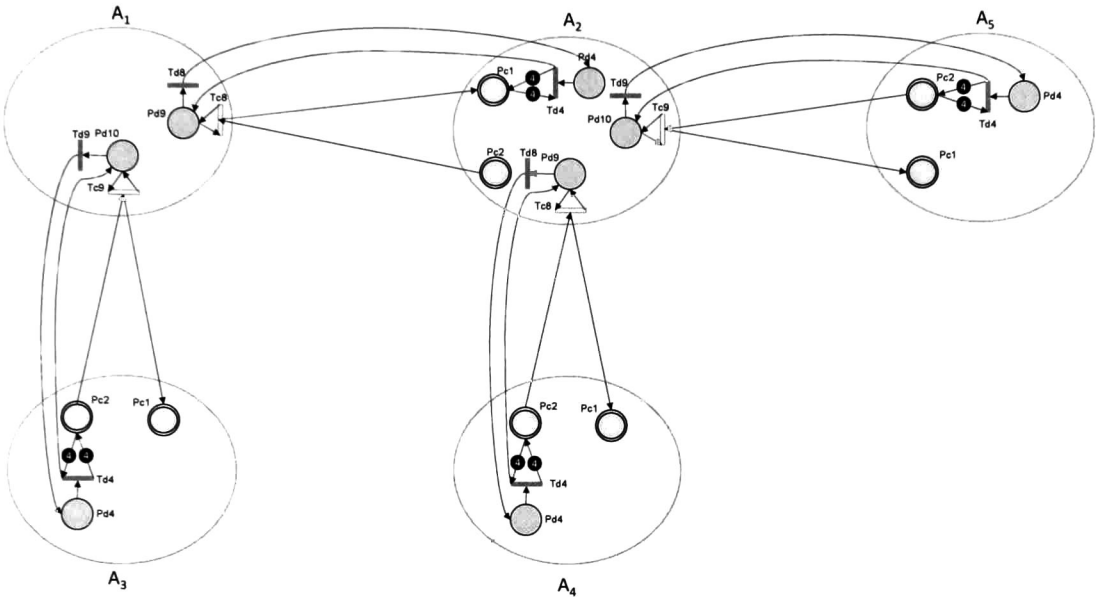


Figure 6.23 Communication nodes in the networked agent system

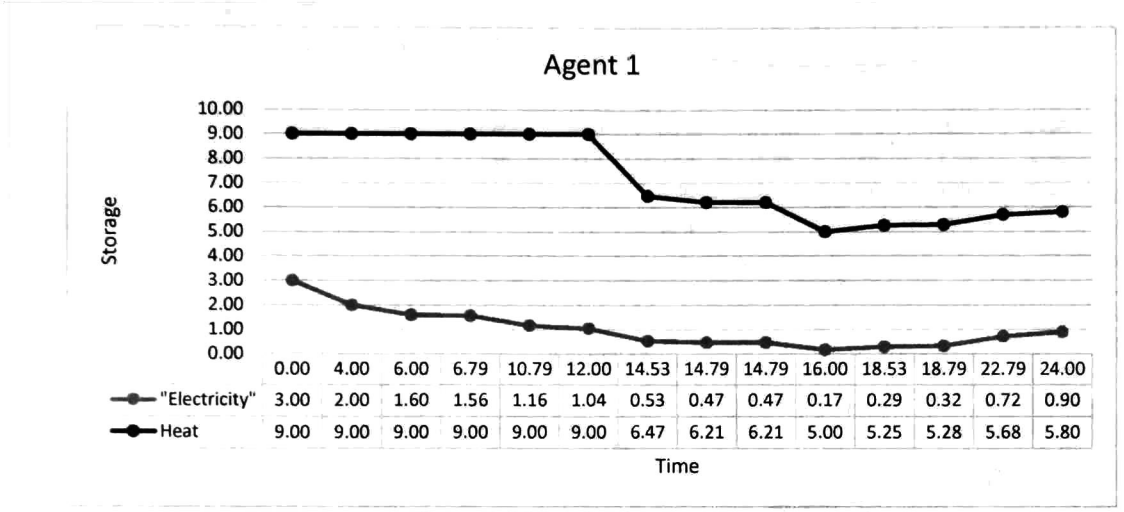
Pd_1 share the enabling of two continuous transitions, Tc_6 and Tc_7. When the continuous transitions are enabled at the same time, the marking of Pd_1 influences the firing velocity of these transitions as if the marks were split, by reducing their firing velocities by half (Refer to one server semantics in [David,2010]). With the aim of do not increase the complexity to the presented model, the instantaneous firing speeds of Tc_6 or Tc_7 are never reduced, as if Pd_5 was duplicated, just connected to one continuous transition. These duplicated nodes are omitted. The same case applies to Pd_5, Tc_1 and Tc_2.

For the simulation, the flow rate and the timing associated to each transition are those shown in Figure 5.23. The initial marking considered for each agent is:

- $mA_1=[0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 3\ 3\ 9\ 0]^T$
- $mA_2=[0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 2\ 4\ 4.5\ 4.5]^T$
- $mA_3=[0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 2.5\ 3.5\ 4.5\ 4.5]^T$
- $mA_4=[0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 4\ 2\ 8\ 1]^T$
- $mA_5=[0\ 1\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 0\ 0\ 2\ 4\ 8\ 1]^T$

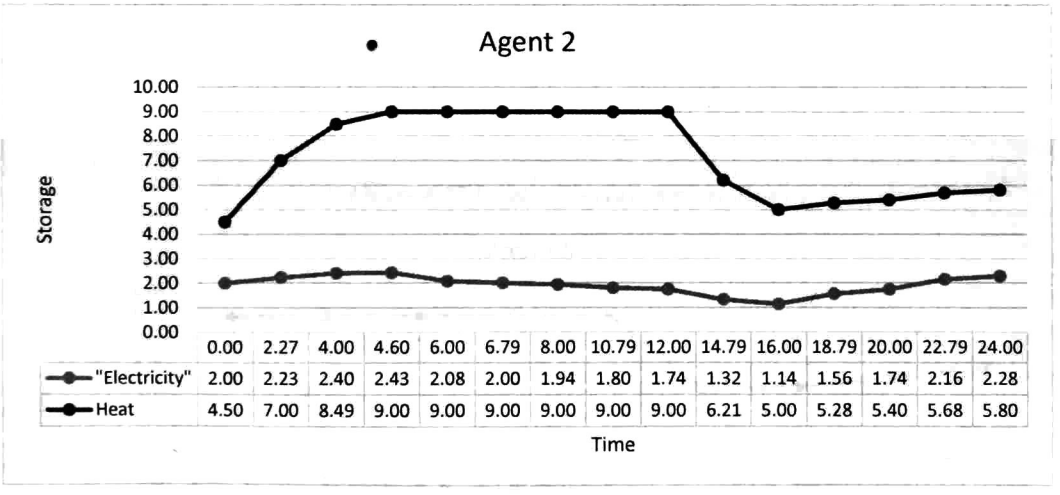
The following policy resolutions are considered: [Tc4, Tc5, Tc6] < [Tc8 Tc9], Tc10 < Tc1 and Tc2 < Tc3. The energy units considered are kWh and the time for simulation is 24 hrs.

The simulation results are presented in the following graphs. Figures 6.24-6.28 show the energy level storage for all agents.



	0.00	4.00	6.00	6.79	10.79	12.00	14.53	14.79	14.79	16.00	18.53	18.79	22.79	24.00
"Electricity"	3.00	2.00	1.60	1.56	1.16	1.04	0.53	0.47	0.47	0.17	0.29	0.32	0.72	0.90
Heat	9.00	9.00	9.00	9.00	9.00	9.00	6.47	6.21	6.21	5.00	5.25	5.28	5.68	5.80

Figure 6.24 Levels of energy storage, Agent 1



	0.00	2.27	4.00	4.60	6.00	6.79	8.00	10.79	12.00	14.79	16.00	18.79	20.00	22.79	24.00
"Electricity"	2.00	2.23	2.40	2.43	2.08	2.00	1.94	1.80	1.74	1.32	1.14	1.56	1.74	2.16	2.28
Heat	4.50	7.00	8.49	9.00	9.00	9.00	9.00	9.00	9.00	6.21	5.00	5.28	5.40	5.68	5.80

Figure 6.25 Levels of energy storage, Agent 2

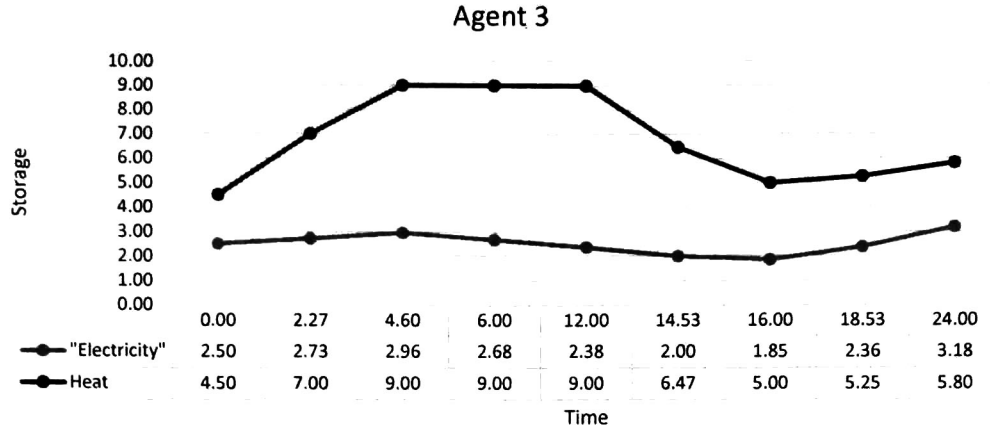


Figure 6.26 Levels of energy storage, Agent 3

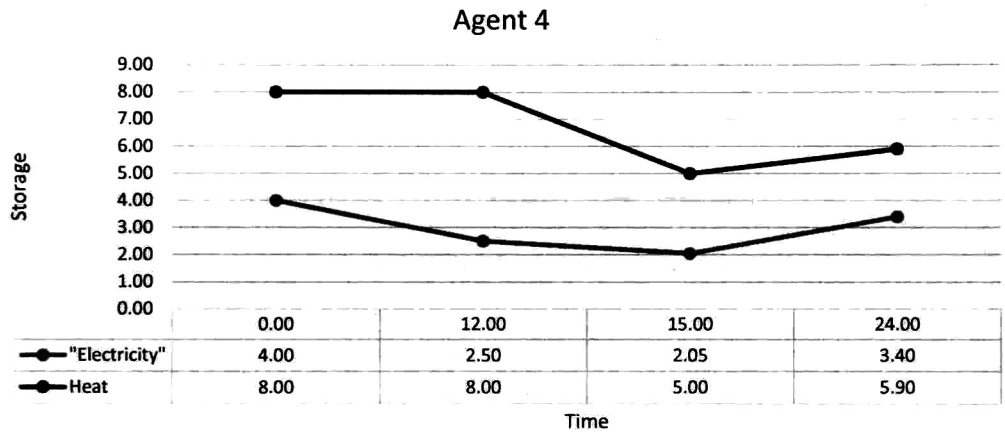


Figure 6.27 Levels of energy storage, Agent 4

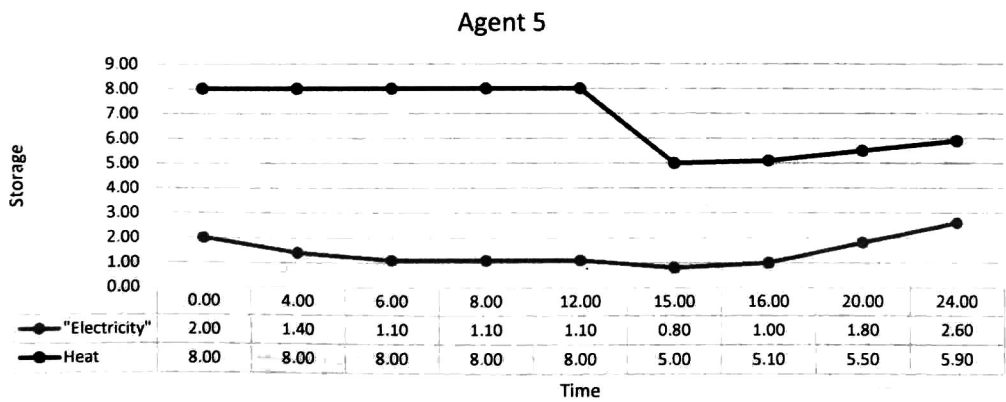


Figure 6.28 Levels of energy storage, Agent 5

Figures 6.29-6.31 present the energy production of units 1 and 2. The production only depends on the level of the heat storage, influenced by the heat production and heat consumption both flows do not depend of external events, only internal. Agents 2 and 3 start with the same initial marking for heat storage, thus the graphs for Agents 2 and 3 are the identical; similarly this situation occurs for Agents 4 and 5.

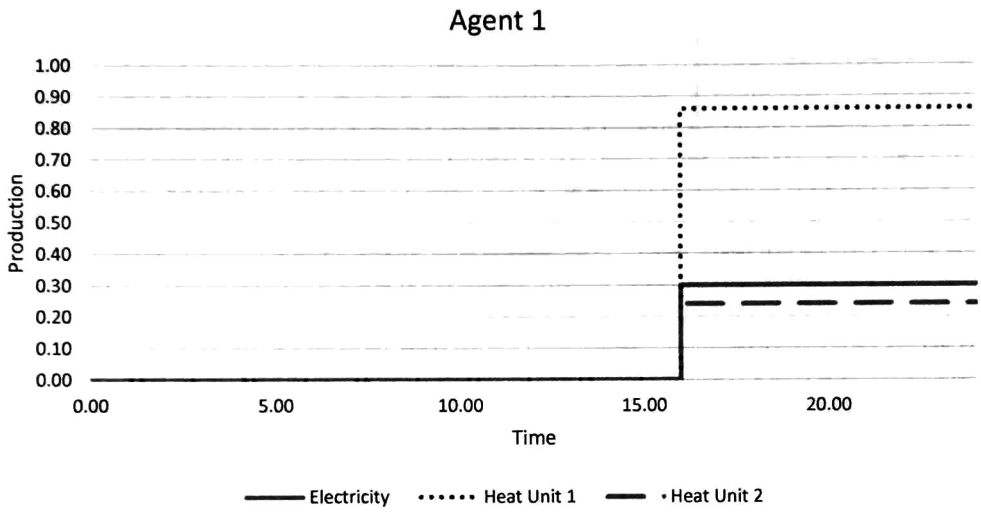


Figure 6.29 Levels of energy storage, Agent 1

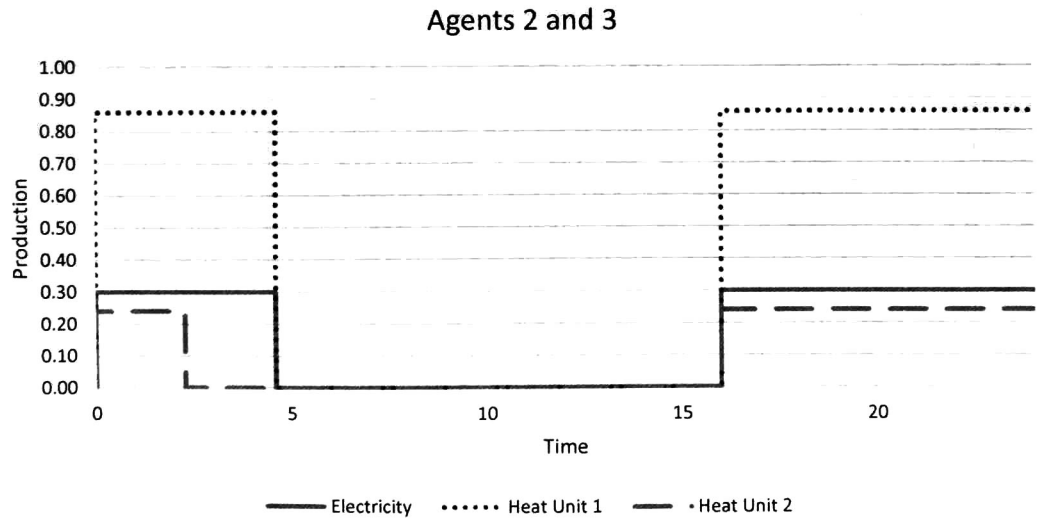


Figure 6.30 Levels of energy storage, Agents 2 and 3

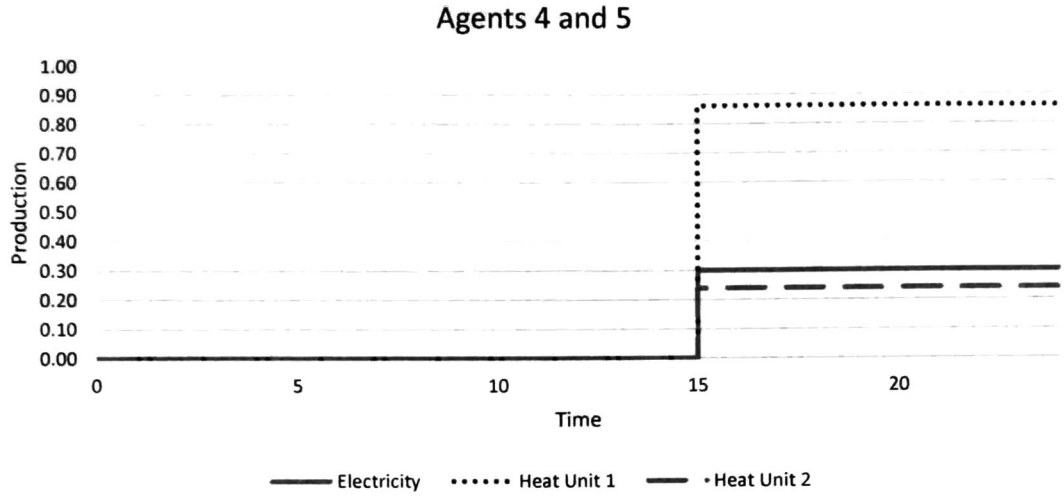


Figure 6.31 Levels of energy storage, Agents 4 and 5

Figures 6.32 and 6.33 present the exported electricity from agents 1 and 2, these corresponding flows are the imported electricity from agents 2, 3, 4 and 5.

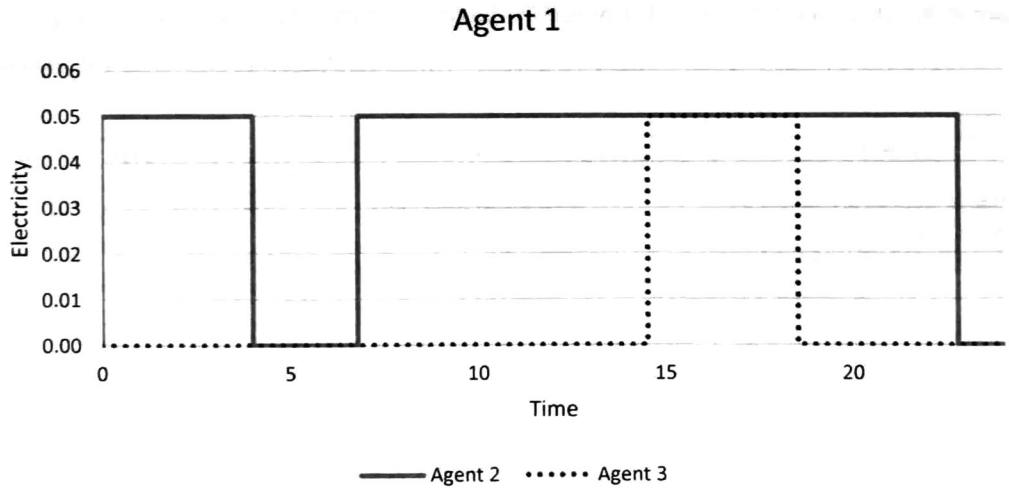


Figure 6.32 Exported electricity, Agent 1

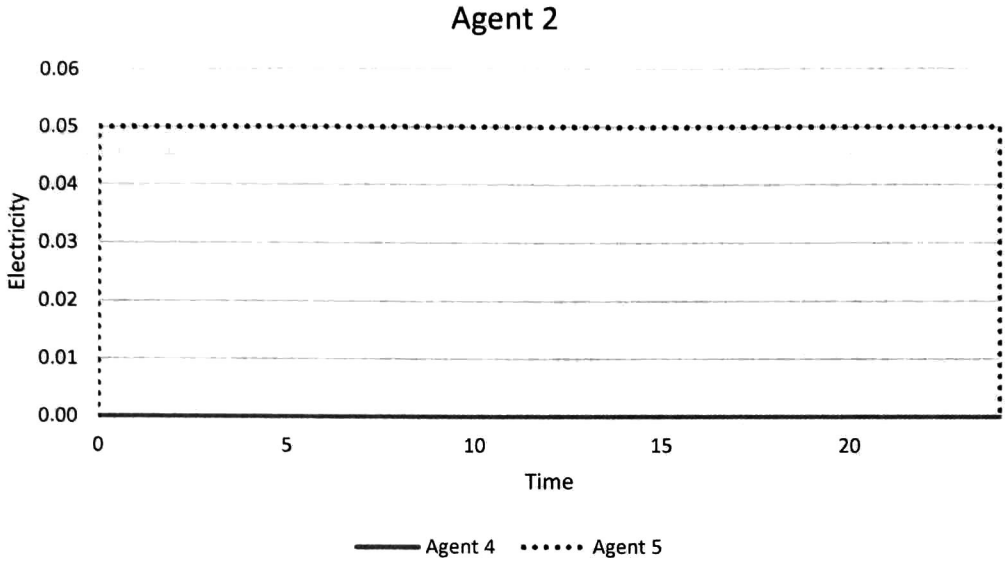


Figure 6.33 Exported electricity, Agent 2

The table 6.5 describes the events generated in each agent, the information is organized in the columns: simulation time, agent, event occurred in the THPN, its meaning in the household system, and the graphs where this event can be observed. Notice that the three first events are performed in all agents, the electricity consumptions.

Time	*Agent*	*Event in THPN*	*Meaning in the system*	*Graphs*
6	All	Td_3	From afternoon to night	Figures 6.24-6.28, data: Electricity
12	All	Td_1	From night to morning	Figure 6.24-6.28, data: Electricity
24	All	Td_2	From morning to afternoon	Figure 6.24-6.28, data: Electricity
16	Agent 1	Td_5	Storage heat level less or equal to 5. It switches on unit 1 and 2.	Figure 6.24, data: Heat Figure 6.29
0	Agent 2	Td_4	Storage electricity level less or equal to 2. It starts importing.	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
4	Agent 2	Td_{12}	Importation electricity ends	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
6.79	Agent 2	Td_4	Storage electricity level less or equal to 2. It starts importing.	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
10.79	Agent 2	Td_{12}	Importation electricity ends	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
10.79	Agent 2	Td_4	Storage electricity level less or equal to 2. It starts importing.	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
14.79	Agent 2	Td_{12}	Importation electricity ends	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
14.79	Agent 2	Td_4	Storage electricity level less or equal to 2. It starts importing.	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
18.78	Agent 2	Td_{12}	Importation electricity ends	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2

18.78	Agent 2	Td_4	Storage electricity level less or equal to 2. It starts importing.	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
22.79	Agent 2	Td_{12}	Importation electricity ends	Figure 6.25, data: Electricity Figure 6.32, data: Agent 2
0	Agent 2 and 3	Td_5	Storage heat level less or equal to 5. It switches on unit 1 and 2.	Figure 6.25 and 6.26, data: Heat Figure 6.30
2.27	Agent 2 and 3	Td_6	Storage heat level has reach 7 units. It switches off unit 2.	Figure 6.25 and 6.26, data: Heat Figure 6.30: Heat Unit 2
4.6	Agent 2 and 3	Td_7	Storage heat level has reach 9 units. It switches off unit 1.	Figure 6.25 and 6.26, data: Heat Figure 6.30: Electricity, Heat Unit 1
16	Agent 2 and 3	Td_5	Storage heat level less or equal to 5. It switches on unit 1 and 2.	Figure 6.25 and 6.26, data: Heat Figure 6.30
14.53	Agent 3	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.26, data: Electricity Figure 6.32, data: Agent 3
18.53	Agent 3	Td_{12}	Importation energy ends.	Figure 6.26, data: Electricity Figure 6.32, data: Agent 3
15	Agent 4 and 5	Td_5	Storage heat level less or equal to 5. It switches on unit 1 and 2.	Figure 6.27 and 6.28, data: Heat Figure 6.31
0	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
4	Agent 5	Td_{12}	Importation energy ends.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
4	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
8	Agent 5	Td_{12}	Importation energy ends	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
8	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
12	Agent 5	Td_{12}	Importation energy ends	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
12	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
16	Agent 5	Td_{12}	Importation energy ends	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
16	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
20	Agent 5	Td_{12}	Importation energy ends	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
20	Agent 5	Td_4	Storage energy level less or equal to 2. It starts importing.	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5
24	Agent 5	Td_{12}	Importation energy ends	Figure 6.28, data: Electricity Figure 6.33, data: Agent 5

Table 6.5 Generated events in the networked system

Through the simulation the following behavior was observed. Agent 1 attends the exporting energy request from Agent 2 at the beginning of the simulation and again in time 6.79. From this time until 22.79, the electricity level from Agent 2 remains lower or equal than 2 units; every 4 units of time (Delay from transition Td_8, in Agent 1) Pd_4 in Agent 2 is marked and immediately Td_4 is fired requesting electricity from Agent 1. For Agent 3, only in time 14.53 an electricity request is made. Agent 4 never requests electricity because its electricity level always is bigger than 2. And for Agent 5 occurs something similar to Agent 2, its electricity level is lower or equal than 2 all the time, except at the end of simulation, at time equal to 24, and every 4 units of time, an electricity request to Agent

2 is made. As it can be seem in Figures 6.24-6.28 the electricity level storage is never empty for any agent. It means that the consumptions and the interchange of electricity is satisfied all the time.

Simulating continuous and discrete events systems is not an easy task; the described scheme removes a part of its complexity by computing off-line all the possible *ifs* for one agent, and latter using this information in the simulation part. The software allows to simulate each agent, giving them calculation autonomy of its own IB-state information. Only when there is a change in an Agent on the continuous flows or a discrete event occurs affecting their neighbors, communication is performed, interrupting the simulation of each neighbor agent; then the other agents must consider these new events, allowing them to build together the correct computation of the overall networked agent system.

CONCLUSIONS

A general framework for the analysis of networked agent systems has been presented; it addressed modeling, computing and simulation of networks of identical interactive multi-role agents. It supports the design process and analysis of networked agent systems. The main feature of this approach is the possibility to address agent applications in which a hybrid state representation and updating is required. The identical characteristic of these agents does not restrict them of performing different activities, since a multirole execution can be performed by them through the interaction of discrete and continuous events; several operational modes can be set in the agent's model.

The proposed modeling methodology allows describing the agent's internal state where the discrete and continuous components may be distinguished and their interactions may be clearly established. In the methodology an important stage is focused on the agent's state representation as THPN. It provides a support for the analysis of agent network systems through simulation. Two study cases about sensor networks illustrate this part.

In order to cope with the non linear computation of TCPN models, a mathematical representation based on MILP has been proposed. It allows building systematically MILP problems allowing the computation of IB-state information: marking, transition firing speeds and duration. In the representation both structural conflicts and cycles are included; resolution policies known as sharing and priorities have been addressed. The tests performed on models of diverse complexity using Matlab and CPLEX demonstrated the feasibility of the mathematical representation and its efficiency.

Finally, a scheme for the simulation of identical networked agent system is presented. An off-line procedure based on the mathematical representation mentioned above, obtains all the possible IB-states for one agent. Using this information a simulation of the overall networked agent system is performed, where the continuous and discrete parts of the agents are included. The developed software for the simulation was implemented in Java, using the framework JADE. This approach overcomes the performance of algorithmic procedures, proposed in literature, since the instantaneous firing speeds for one agent are off-line computed; this avoids the necessity of repeat this computation for the overall network. A case study about distributed energy household networks was presented.

The work herein presented goes ahead the current research in literature on the addressed problems. However, there is a lot of challenges regarding such problems; for example it could be useful to study more complex agent interconnections, even the possibility to include in the system different kind of agents. This is absolutely possible since the computation of all IB-states for one agent can be performed by knowing all the possible incoming entrances through the time for one agent. Also,

regarding the simulation issue, the scheme has the bare bones for a distributed simulation, but new communication protocols must be developed to cope with the asynchronous independent processing of each agent and the absence of a global clock.

APPENDIX A DOCUMENTATION OF JAVA CLASSES FOR THPN SIMULATION

public class **Init**
extends Object

Main initial class, creates an InputDataGui

Method	Notes	Parameters
Static **main()** void Public	Shows Gui in the screen	String[] [in] args args

public class **InputDataGui**
extends JFrame

This class shows the GUI interface for the input csv files: *WA*, *M0*, *WR*, *ifs* Table and time for simulation

Method	Notes	Parameters
InputDataGui() Package	Constructor	
showGui() void Public	Method to show Gui	

public class **Simulation**
extends Object

Class to perform the simulation

Method	Notes	Parameters
performSimulation () void Public	Method to store all the cvs files in to SimpleMatrix and creates all the agents	
readFile() void Private	Creates and internal ArrayList for each cvs file	String [in] file the csv file's name String [in] var kind of variable to be store
toSimpleMatrix() SimpleMatrix Private	Method to convert an arrayList <Double> to a SimpleMatrix @return SimpleMatrix	ArrayList<ArrayList<Double>> [in] convert the ArrayList to be converted

public class **ibState**
extends Object

Class to represent the IB-State information: *#ibstate*, agent, marking, *dt* and v.

Method	Notes	Parameters
existIBs() boolean Public	Method to verify if exist an specific IB-State @return true/false	int [in] ibs number of IB-State int [in] Agent

Method	Notes	Parameters
		number of Agent
getDt() double Public	Method to get *dt* @return *dt*	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
getM() SimpleMatrix Public	Method to get the marking @return marking	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
getNextEv() SimpleMatrix Public	Method to get nextEv @return nextEv	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
getTempo() SimpleMatrix Public	Method to get tempo = duration + instantaneous firing speed @return tempo	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent **int** [in] noPc
getv() SimpleMatrix Public	Method to get instantaneous firing speed @return *ifs*	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
getv() SimpleMatrix Public	Method to get instantaneous firing speed vector @return *ifs*	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
ibState() Public	Constructor	**int** [in] places for an agent **int** [in] transitions for an agent
saveToFileCSV() void Public	Method to save the information stored for an Agent in a cvs file	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent
setIbState() void Public	Method to add the information for an IB-State agent	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent **SimpleMatrix** [in] marking marking for the Agent **double** [in] delthat *dt* **SimpleMatrix** [in] *ifs* instantaneous firing speeds
updateDt() void Public	Method to update dt	**int** [in] ibs number of IB-State **int** [in] Agent number of Agent **double** [in] delthat new *dt* to be substituted

Method	Notes	Parameters
updateNextEv() void Public	Method to update nextEv @param new nextEv to be substituted	double [in] absT int [in] ibs number of IB-State int [in] Agent number of Agent SimpleMatrix [in] nE

public class **ThpnAgent**
extends Agent

Class Agent to simulate a THPN

Method	Notes	Parameters
dEvDt0() void Private	Method for performing initial discrete events	Agent [in] myAgent
dTrFiredIn() SimpleMatrix Private	Method to set if a discrete transition can be immediately fired @param @return enabled transitions fired immediately	
generalNodeProcedure() void Private	Method to perform the general node procedure	Agent [in] myAgent Object agent who is performing the procedure double [in] inFlow Incoming flow
getMindt() double Private	Method to choose the smallest *dt* @return	SimpleMatrix [in] balance vector with the balance of each place (Who is going to be the first empty place?)
setCdt() SimpleMatrix Private	Method to set the *dt* for continuous places @return	SimpleMatrix [in] balance vector with the balance of each place (Who is going to be the first empty place?)
setDdt() SimpleMatrix Private	Method to set the *dt* for discrete transitions @param @return	SimpleMatrix [in] balance
sendMsg() void Private	Method to send a message to the outNeighbors	Agent [in] myAgent Object agent who is sending the
setup() void Protected	Initial behavior for the agent	
takeDown() void Protected	Method to perform agent clean-up operations here	

Method	Notes	Parameters
dEvDt0() void Private	Method for performing initial discrete events	Agent [in] myAgent
dTrFiredIn() SimpleMatrix Private	Method to set if a discrete transition can be immediately fired @param @return enabled transitions fired immediately	

public class **inMessages**
extends Object

Class to store the messages send to each agent who has incoming flows, there is an object for each one of them

Method	Notes	Parameters
getCurrentInFlows() SimpleMatrix Public	Method to get the flows in the corresponding vector @return	double [in] absTime SimpleMatrix [in] sm
getNumberOfAgents() int Public	Method to get the number of agents @return number of agents	
getValueDiscFire() SimpleMatrix Public	Method to get if all the discrete messages was received @return a vector with the fire of discrete transitions	
getValueFlows() double Public	Method to get the sum of incoming flows @return a value for flows	
reset() void Public	Method to put all discrete fires to zero	
InMessages() Public	Constructor	ArrayList <Integer> [in] Agents List of all In Neighbors
setCurrentInFlows() void Public	Method to update at the current inflow from a received messages	
storeMessages() double Public	Method to store the messages sent by In neighbors	String [in] msg Message sent by a In neighbor

private class **InitialEvents**
extends Beahavoir

Class to perform the initial behavior for agents with discrete elements

Method	Notes	Parameters
action() void Public	Initial procedure	
done() boolean Public	Method for checking stop conditions	

private class **leafnode**
extends Beahavoir

Class to perform the behavior for leaf nodes

Method	Notes	Parameters
action() void Public	Perform the cyclic procedure until the end of simulation for agents without communication.	
done() boolean Public	Method for checking stop conditions	

private class no**leafnode**
extends Beahavoir

Class to perform the behavior for no leaf nodes

Method	Notes	Parameters
action() void Public	Perform the cyclic procedure until the end of simulation for agents with communication.	
done() boolean Public	Method for checking stop conditions	

REFERENCES

[Anastasi,2009] *Anastasi, G., Conti, M., Di Francesco, M., Passarella, A. (2009). Energy conservation in wireless sensor networks: A survey. Ad Hoc Networks, 7(3), 537-568.*

[Balduzzi,1999] *Balduzzi, F.. Menga, G., Giua, A., Seatzu, C. (1999). "A Linear State Variable Model for First-Order Hybrid Petri Nets" Proc. 14th IFAC World Congress (Beijing, China), Vol. J, pp. 205-210.*

[Balduzzi,2000] *Balduzzi, F., Menga, G., and Giua, A. (2000).First-order hybrid Petri nets: a model for optimization and control. IEEE Trans. on Robotics and Automation, 16(4):382–399.*

[Balduzzi,2001] *Balduzzi, F., Di Febbraro, A., Giua, A., Seatzu C. (2001). "Decidability Results in First-Order Hybrid Petri Nets," Discrete Event Dynamic Systems, Vol. 11, No. 1&2, pp. 41-58.*

[Bemporad,1999] *Bemporad, A., Morari, M. (1999). Control of systems integrating logic, dynamics, and constraints. Automatica. Vol. 35, no. 3, pp. 407-427.*

[Bos,1999] *Bos, A., Weerdt, M.D., Witteveen, C., Tonino, H., Valk, J. (1999). A dynamic systems framework for multi-Agent experiments. European Summer School on Logic, Language, and Information, Foundations and Applications of Collective Agent Based Systems workshop.*

[Bosse,2007] *Bosse, T., Sharpanskykh, A., Treur, J. (2007). Integrating agent models and dynamical systems. Proceedings of the 5th international conference on Declarative agent languages and technologies V, 50–68.*

[Bratman,1998] *Bratman, M. E., Israel, D. J., Pollack, M. E. (1988). Plans and resource-bounded practical reasoning. (R. Cummins & J. L. Pollock, Eds.)Computational Intelligence, 4(3), 349-355.*

[Cplex,2013] *http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/*

[David,2010] *David, R., Alla, H. (2010). Discrete, continuous, and hybrid Petri nets. Springer.*

[Dotoli,2008] *Dotoli,,M.,Fanti, M.P., Giua, A., Seatzu, C. (2008), First-order hybrid Petri nets. An application to distributed manufacturing systems. Nonlinear Analysis: Hybrid Systems Vol. 2., No. 2, pp. 408-430.*

[Dotoli,2009] *Dotoli, M., Fanti, M.P., Iacobellis, G., Mangini, A.M.. (2009)."A First Order Hybrid Petri Net Model for Supply Chain Management", IEEE Transactions on Automation Science and Engineering, vol. 6, no. 4, New York, USA, ISSN 1545-5955.*

[Feber,1996] *Ferber, J., Müller, J.-P. (1996). Influences and reaction: a model of situated multiagent systems. (M. Tokoro, Ed.)2nd International Conference on Multiagent Systems Japan AAAI Press, 72–79.*

[Fischer,1996] *Fischer, K., Müller, J. P., Pischel, M. (1996). A Pragmatic BDI Architecture. (M. N. Huhns & M. P. Singh, Eds.)Environment, 1037, 203-218.*

[Floudas,2004] *Floudas, C.A. and X. Lin.(2004). "Mixed Integer Linear Programming in Process Scheduling: Modeling, Algorithms, and Applications", Annals of Operations Research.*

[Franceschelli,2011] *Franceschelli M., Giua A., Seatzu C. (2011). "Distributed Averaging in Sensor Networks Based on Broadcast Gossip Algorithms," Sensors Journal, IEEE, vol. 11, pp. 808-817.*

[Fregene,2001] Fregene,K.,Kennedy, D., Wang,D. (2001). HICA: A framework for distributed multiagent control. In: Proceedings of IASTED International Conference on Intelligent Systems and Control. 187-192.

[Glober,1975] Glover, F. (1975). "Improved Linear Integer Programming Formulations of Nonlinear Integer Problems." Management Science 22, 455–460.

[Gudiño-Mendoza,2011] Gudiño-Mendoza, B., López-Mellado, E. (2011). A Modeling Framework for Developing Networked Agents Applications. 8th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE).

[Gudiño-Mendoza,2013] Gudiño-Mendoza, B., López-Mellado, E. (2013). Modelling networked agents' behaviour using timed hybrid Petri nets. The 2013 Iberoamerican Conference on Electronics Engineering and Computer Science.

[Gudiño-Mendoza, 2014] Gudiño-Mendoza, B., López-Mellado, E., Alla, H. (2014). A Linear Characterization of the Switching Dynamic Behavior of Timed Continuous Petri Nets with Structural Conflicts. Nonlinear Analysis: Hybrid Systems. Submitted.

[Holger,2005] Holger, K., Willig, A. (2005), Protocols and Architectures for Wireless Sensor Networks. Wiley.

[Ingrand,1992] Ingrand, F. F., Georgeff, M. P., Rao, A. S. (1992). An architecture for real-time reasoning and system control. Ieee Expert Intelligent Systems And Their Applications, 7(6), 34-44. IEEE Educational Activities Department

[JADE,2014] http://jade.tilab.com/doc/api/

[Julvez,2004] Júlvez, J., Bemporad, A., Recalde, L., Silva, M., (2004). Event-driven optimal control of continuous Petri nets. 43rd IEEE Conference on Decision and Control.

[Negenborn,2004] Negenborn, R.R, De Schutter, B., Hellendoorn, J. (2004).Multi-Agent Model Predictive Control: A survey. Delft Center for Systems and Control, Delft University of Technology.

[Negenborn,2006] Negenborn, R. R., Schutter, B. D., Hellendoorn, H. (2006). Multi-agent model predictive control for transportation networks with continuous and discrete elements. Tech Rep 06014 Delft Center for Systems and Control Delft University of Technology Delft, 19.

[Negenborn,2007] Negenborn, R. R. (2007) Multi-Agent model predictive control with applications to power networks. TRAIL Thesis Series T2007/14, The Netherlands TRAIL Research School.

[Nishi,2009] Nishi, T., Shimatani, K., & Inuiguchi, M. (2009). "A decomposition method for optimal firing sequence problems for first-order hybrid Petri nets". IEEE International Conference on Systems Man and Cybernetics.

[Pehnt,2006] Pehnt, M., Cames, M., Fischer, C., Praetorius, B., Schneider, L., Schumacher, K., Vob, J. (2006). Micro Cogeneration: Towards Decentralized Energy Systems. Springer.

[William,1999] William, H.P.(1999). Model building in mathematical programming, fourth Edition, John Wiley & Sons, Ltd.

[Wooldrige,2002] Wooldridge, M. (2002). An Introduction to MultiAgent Systems. (W. Edition, Ed.) Computer (Vol. 86, p. 348). John Wiley & Sons.

[Yu,2006] Yu, Z., Cai, Y. (2006). Object-Oriented Petri nets architecture description language for multi-agent systems. IJCSNS, 6(1), 256-260.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Modelado y cómputo del comportamiento de agentes interconectados usando redes de Petri híbridas temporizadas

del (la) C.

Gema Berenice GUDIÑO MENDOZA

el día 22 de Agosto de 2014.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Antonio Ramírez Treviño
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Carlos Renato Vazquez Topete
Profesor Investigador
Centro Universitario de los Valles,
UdG