Protocol Engineering for Multi-Agent Interaction

Amal El Fallah-Seghrouchni[1], Serge Haddad[2], and Hamza Mazouzi[2]

[1] LIPN - Université Paris Nord,
elfallah@lipn.univ-paris13.fr
[2] LAMSADE - Université paris Dauphine,
{haddad, mazouzi}@lamsade.dauphine.fr

Abstract. This paper focuses on the study of the pragmatics of multi-agent systems design, proposing an efficient approach for the interaction protocol engineering. This approach combines two complementary paradigms: 1) Distributed observation is used to capture the concurrent events inherent to the interactions between agents, through causal dependency, in order to explain the relationships within conversations or group utterances; 2) Colored Petri Nets are used as a suitable formalism to identify interaction-oriented designs while providing the means to model, analyze, and validate large scale applications.

1 Interaction protocols issue: languages and models

This paper focuses on the study of the pragmatics of multi-agent systems (MAS) design, proposing an efficient approach to build a robust and secure interaction between agents, as an essential component of the dynamics of MAS. Communication is often used by agents to enable cooperation, common tasks and goals achievement, and information exchanged, i.e. data, knowledge and plans. Message passing is a common paradigm for agents' communication. Such a paradigm is the necessary mean for cooperation which requires a shared background of the skills of the agents, specially when such skills are as complex as perception, learning, planning, and reasoning. Interaction modeling, as well as the development of standard languages for agents' communication (ACLs), remains a crucial problem in MAS design.
We propose an approach that tackles the problem of protocol engineering through the specification, the analysis, and the verification of such protocols when several agents are involved. This approach combines two complementary paradigms: Distributed observation is used to capture the relevant events underlying the interacting situations, while Colored Petri Nets (CPN) [10] provide an efficient formalism to specify, model, and study several kinds of communication protocols. We also claim that our approach is generic, in the sense that it is independent of any ACL.

1.1 Interaction languages

In order to support organizational interaction, communication, and cooperation in MAS, many frameworks [11] [8] have been proposed towards standardization for formalizing the flow of interaction between agents . These frameworks intend

to develop a generic interaction languages by specifying messages and protocols for inter-agent communication and cooperation. Such languages focus especially on how to describe exhaustive speech acts [1] [20], both from the syntactical and semantical points of view that support a language of knowledge representation. Nevertheless, the ontological aspect and the resort to conventions may help to ensure a coherent collective behavior of the overall system, even if the conversational aspect is not easy to be guaranteed.
Research in interaction languages includes the Knowledge Sharing Effort (KSE) that outputs specification for the Knowledge Querying and Manipulation Language (KQML) [11], and the Knowledge Interchange Format (KIF). KQML proposes an extensible set of performatives, which define the permissible operations that agents may attempt on each other's knowledge and goal stores. Nevertheless, some observations about KQML has been recently pointed out, e.g., some performatives are ambiguous while others are not really performatives at all, and there are no performatives that commit an agent to do something. Another criticism to KQML is its deficiency regarding to a clear semantics independent of the structure of the agents [4].
More recently, the international collaboration of member organizations within FIPA (*Foundation for Intelligent Physical Agents*), proposed and specified some standards for the agent technology and especially, an agent communication language called ACL [8]. ACL recovers the syntactical idea of KQML which allows to build interactions enriched by a formalized semantics of performatives, and consequently, it enables the expression of a set of high level protocols and primitives to control the information exchange between agents.
However, few work related with the analysis and the validation of these protocols have been done. In [5] [6] a formal study of the FIPA Protocols is proposed and it shows that some of them may lead to some incorrect behaviour (e.g. deadlock situations between agents).

1.2 Interaction Models

The formal specification of interactions have been less successful. In fact, the modeling of interactions confines itself with the use of misfit formalisms, such as the graph of predefined states, to describe the progress of the agent according to the kind of received messages. Other models like automata or more specific graphs (e.g. the Dooly-graph [18]) have also been used to describe conversations between agents. These models of representation are practical to specify the structure of the conversations when they appear as isolated communications, but they exhibit poor capacity for computing complex protocols basically because: 1) any graph state includes all the local states of the agent leading to the combinatory explosion in the case of real and complex protocols; 2) most of the formalisms used take in charge only sequential processes. Moreover, when these formalisms take into account temporal aspects (in our case, specified through the causality concept), they assume the existence of a global clock what constitutes a strong constraint, i.e., agents must run on the same site. In addition, these models are very limited when it is useful to take into account the concurrency, which is the key stone of MAS since agents are often involved simultaneously in several interactions.

Our point of view is that it should be more judicious to resort to well-known and well-tested formalisms for concurrent systems, such as CPN [10] for at least three reasons: 1) they naturally take in charge concurrency; 2) they make easier the factorization process of treatments; and 3) they offer several methods for the analysis and the validation of the modeled protocols.
This paper is organized as follows: section 2 outlines both our framework hypotheses and our aims, namely the modeling and the analysis of interactions. Section 3 briefly introduces the paradigm of distributed observation through the causality concept and illustrates its usefulness to build the causal graph of interactions. Section 4 presents the CPN formalism through some modeling aspects. The pattern matching algorithm presented in Section 5 is our main focus in this paper. The algorithm leads to recognize the nature of interaction involved between agents. Section 6 concludes and lights up our future work.

2 Conceptual framework

Our approach is structured on five phases (see figure 1):
1) In [5] we have proposed an efficient study of interaction. Based on an *on-line* distributed observation of the MAS running, it allows to capture traces of the relevant events underlying the agents' interactions (see phase 1).
2) While exploiting the obtained traces, the second phase builds the causal graph of events [6] underlying the interactions that have been occurred during the distributed execution, i.e., agents may be situated and run on distinct sites (see phase 2). Let us remark that the reconstitution mechanism is ensured *off-line* and based on logical clocks proposed by J. Fidge [9] and F. Mattern [14] which take into account the local concurrency, i.e., on the same site, for multi-threading process for instance.
3) The third phase represents our main interest in this paper. It corresponds to the recognition of interactions based on a pattern matching algorithm. The algorithm is jointly based on the causal graph of events and the CPN models as filters (CPN_patterns). Knowing that several computations may be associated with the same CPN model and consequently with the same protocol, it becomes necessary to identify the right instance of the right protocol. We overcome this difficulty thanks to the true-concurrency semantics by means of the unfolding Petri Net techniques. In fact, at the opposite of the interleaving concurrency semantics, the unfolding Petri nets method enables to associate a set of unfolding nets with a given protocol modeled as CPN.
4-5) Work in progress addresses the phases four and five that will not be detailed in this paper. Briefly, the fourth one aims to explain the behaviour of the MAS according to the adopted protocols while the fifth one corresponds to the learning process. Supported by the central agent, the learning process tries to deduce both qualitative and quantitative criteria to be communicated to the other agents. These criteria should be taken into account by the agents as guidelines to improve their behavior and especially their future interactions. The hypotheses of our framework are:

- the MAS is composed of a set of cognitive agents distributed on different sites and may run concurrently.
- the agents communicate exclusively by asynchronous messages.
- each agent has a local strategy for problem solving, reasoning mechanisms, etc.

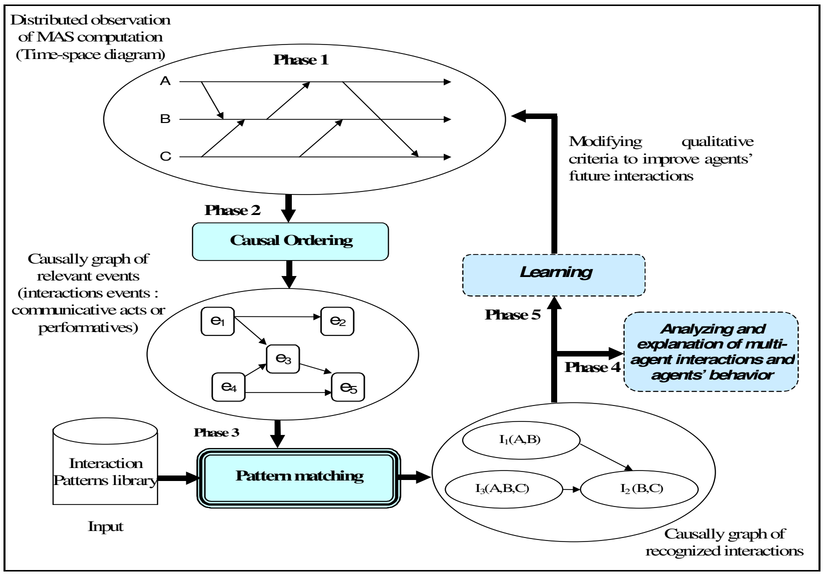

Fig. 1. A structured approach for interaction design in MAS

- all the protocols are modeled through CPN formalism and are available in the protocol library (CPN_Patterns) shared by agents.
- each agent communicates with each other w.r.t. the predefined protocols.
- the observation is distributed in the sense that each agent has a local module, which observes and traces the significant events corresponding to the emissions/receptions of the agent messages and some local events.
- the analysis and learning processes are ensured *off-line* by the central agent.

3 Distributed observation applied to interaction in MAS

Distributed problem solving involves complex interaction strategies of cooperation, in the sense that they are non-deterministic, hard to be interpreted and sometimes neither completely reproducible, nor predictable.
A possible solution is to observe the computation at run time in order to record the most significant events and their causality relation. Generally, at a given level of an application, only few events are relevant to the observation process. For example, in interaction protocol, only events according to the protocol running are significant. An observer of MAS may be any entity that attempts to watch the system while the computation is in progress or examine a post-mortem log or trace of events. In our framework, we consider a distributed execution as composed of a set of agents that communicate asynchronously by message passing over a communication network [19]. At the most abstract level, a computation of MAS can be defined as a set of events. Each agent generates an execution trace, which is a finite set of local atomic events in some specific order. There are two kinds of event: *interaction events*, i.e., sending and receiving messages, and *internal events*

(e.g. updating internal state of the agent).
We are particularly interested about the concept of time in distributed systems which may be usefully used in many ways, especially for ordering events using logical clocks. Another aspect is that many applications require identifying "*cause and effect*" relationships in event occurrences (e.g. debugging programs). In any case, the model (called *happened-before*) [13] is necessary to give information about the *causally precedes* relation among the events. This relation can be defined as follows:

- *Locally precedes* relation between events on a single site which generally provides a total order,
- *Immediately precedes* relation between a couple of events e and f of exchange messages ; if e is the sending of the message and f is the reception of the same message.

Now, the *causally precedes* relation denoted by "$\rightarrow$" can be defined as the transitive closure of the union of the two relations above. Let us remark that one of the major difficulties in distributed agents is that the ordering relation between events is a *partial* order.
Our model goes on to ***partially order*** the events on a single site which would allow events within an agent to be independent or concurrent (e.g. independent threads). This model [21] extends the happened-before model in order to: 1) address the problem of *false causality* when local events are independent; and 2) capture the semantics of multiple local threads of control.
This relation can be represented with a ***minimal oriented graph*** denoted *causal dependency graph* (see [5], [6] and [15] for the graph construction).
Finally, the mechanisms that capture causality are widely developed depending on the level of information required. In our approach, the vector clocks mechanism [9] [14] is adopted since it enables to exactly know the *causally precedes* relation between events.

4 Modeling interactions by means of CPN

The protocol engineering typically comprises various stages including specification, verification, performance analysis, implementation, and testing.A computational specification of complex interactions is the description of a combinatory of exchanging performatives between agents [12].We consider interaction protocols as the basic ones from which complex and high-level interactions can be built [2]. As argued in section 1.2, we adopt the CPN formalism to handle interactions. Next, we present the syntactical and semantical features of CPN [10] illustrated through a significant example, FIPA-Query-Protocol [8] (see figure 2). FIPA-Query-Protocol simply allows one agent to request another to perform some kind of inform action (*query-act*) and the receiver to answer with a normal *inform act* or, in some way, that it cannot answer (i.e. *refuse*, *failure* or *not-understood*). The presented example extends the FIPA-Query-Protocol since it involves one sender and any set of receiving agents (e.g. broadcast of a query).

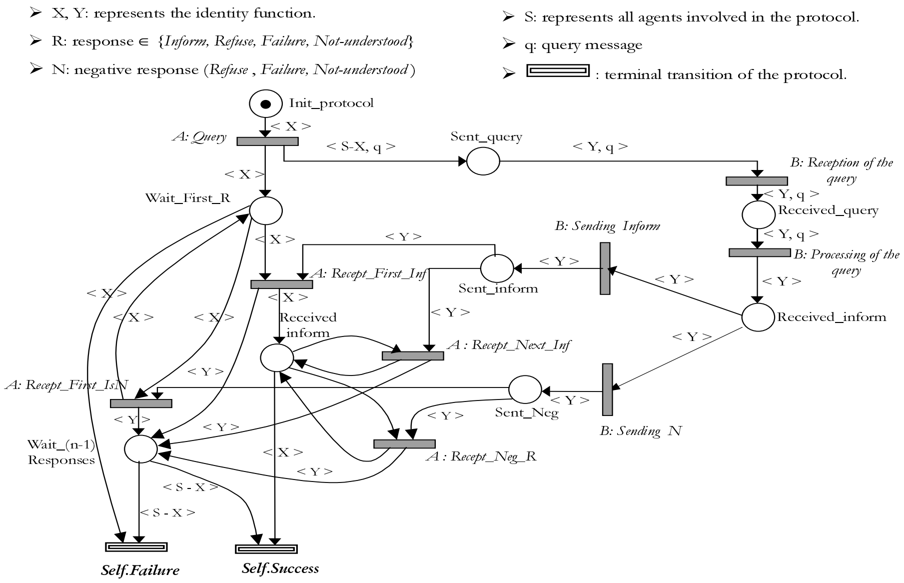

Fig. 2. The CPN model of FIPA-Query Protocol

4.1 Syntactical aspect of CPN

Petri Nets are state and action oriented models at the same time. In our framework, the modeling is concentrated on actions that represent the events to be observed. A Petri net is defined as usual: it consists of *places* (circles) and *transitions* (rectangles) which are connected by arcs.

- Places: each place contains tokens called *marking* which describe the state of the system. In ordinary Petri nets, tokens do not support information while in colored nets, the tokens are labeled by a type of data - possibly structured - called a color. Each token carries a data value which belongs to the type of the corresponding place. For instance the associated data type of the place *Init_Protocol* is the set AG which represents all the agents in our system and its initial marking is the involved agents' identifiers. Let us remark, that in the general case, a place may contain more than one token with the same data value, i.e. we have a multi-set of tokens. Hence a marking is a function which maps each place into multi-set of tokens of the associated type.
- Transitions: they model the change of states. In our model, each transition is associated with an action of an agent. The activation of a transition is called a firing, for instance as the result of the query communication act represented by the transition *Query*. When the condition of activation of a transition is fulfilled we say that the transition is fireable.
- Arcs: an *incoming arc* (from place to transition) indicates that the transition may consume tokens from the corresponding place while an *outgoing arc* (from transition to place) indicates that the transition may add tokens in the corresponding place. The exact number of tokens and their data values are

determined by the arc expressions w.r.t. the semantics of the CPN (i.e. the firing rules of transitions).

4.2 Semantical aspect of CPN

The dynamic of the modeled system (i.e. the behaviour of the net) is given through the firing of transitions.The firing of a transition T_i is a two-steps operation: it consumes tokens from input places ($Pre(T_i)$) and produces tokens into output places ($Post(T_i)$). In colored Petri nets a set of variables is associated with each transition. The expressions that label the arcs around the transition are built with these variables. The firing of a transition involves an instantiation of the variables and an evaluation of the expressions which give the multi-set of tokens to be consumed or produced. For example, the arc labeled $< S\text{-}X,\ q >$, where q is a constant (query message), means that the firing of the transition *Query* needs to bind $< S\text{-}X >$ to a value from the set of all agents except the sender, i.e. $AG \backslash \{A_i\}$.
Remarks:
1) In the CPN model there is no connection between particular input tokens and particular output tokens, i.e. both their numbers and their values may differ.
2)The protocol execution is ensured by a thread or a process of each involved agent what allows then the concurrency within an agent.
The CPN-Protocol works as follows: The sender sends a *Query act* for a proposal to all the other agents and enters a waiting state (*Wait_First_R*). On receiving the message (*Reception_Query*), each receiver processes the query, sends a response which can be positive (*Sending_Inform*) or negative (*Sending_N*), and then enters in waiting state (*Sent_Inform* or *Sent_Neg*). The sender accepts only the first positive answer while others are rejected. Once all the responses of the agents are received, the agents come out of the protocol either in successful (*Self_Success*) or failure way (*Self_Failure*). Let us note that one can distinguish four cases when receiving the responses according to the following transitions:

- *Recept_First_Inf*: reception of the first inform,
- *Recept_Next_Inf*: reception of the next answers whose type is “Inform” and necessarily after the first *Inform* is received,
- *Recept_First_IsN*: the first response received is a negative response,
- *Recept_Neg_R*: reception of a negative responses that occurs after the first *Inform* is received.

The reader can easily verify that the protocol successes if the sender receives at least one positive answer, otherwise it fails.

5 Recognition Algorithm based on unfolding Petri Nets

Our algorithm is based on the partial-order semantics of Petri Nets and well-known as unfoldings of Petri Nets [17]. The main interest of this method is that, at the opposite of the interleaving concurrency semantics, it enables to associate a set of unfoldings with a given CPN, in our case, an interaction protocol. An unfolding, also called a “process net”, formalizes a concurrent run of a protocol which can be interpreted in terms of causality between the associated events [3].

5.1 Partial-order semantics of Petri Nets

An unfolding is an acyclic Petri net where the places represent tokens of the markings and the transitions represent firings of the original net (see figure 4). To build an unfolding, the following steps have to be executed iteratively:

- start with the places corresponding to the initial marking,
- develop the transitions associated to the firings (w.r.t. to the semantics of CPN) of every initially enabling transition,
- link input places to the new transitions,
- produce output places,
- link the output places to the new transitions.

Let it be remarked that the unfolding may be infinite if the original net includes an infinite sequence. Several methods [16] [7] have been proposed in order to avoid the infinite state problem in the verification of systems and provide finite unfoldings. In our case, the infinite state is not faced since the unfolding we look for corresponds to a specific protocol computation and necessarily finite.

5.2 The recognition algorithm through an example

To begin, a partially ordered set of events is extracted from the global trace since our approach supports that an agent may be involved, simultaneously, in several interactions [6]. The algorithm inputs are the causal graph (CG) and the CPN_Patterns while the expected outputs are the process nets that match with the CG.

Hypothesis

1. In the general case, the same event may be associated with more than one transition labeled with the same event (cf. transitions *Recept_First_Inf* and *Recept_Next_Inf* in figure 2). In order to gain simplicity, a choice function is introduced which returns the transition to be considered for a given event during the pattern matching process. Obviously, if the choice is wrong, we have to backtrack and consider an other alternative.

2. The CPN_Patterns are one-safe (i.e. a place contains at most, one token per color). This makes the detection easier by avoiding the combinatory explosion of the number of states following multiple firings of a transition by the same color tokens. Let us note that since the CPN is one-safe and the events are associated with the transitions, given an event e to be recognized, there is exactly one state in the unfolding net reachable with a transition labeled by e.

Petri Net unfolding example

Let us consider two protocols (cf. figure 3) which provide the same service (sending query messages to agents and reception of their answers). In the first protocol the execution is optimal, i.e., in parallel way; whereas in the second protocol the sending of messages and the reception of the associated answers are in sequence. One can easily verify in $Protocol_2$ that, except for the first firing of T_2 initially enabled from the initial marking, each following firing of T_2 requires at least a token in the input places P_1 and in P_5. As for P_1, (n-1) tokens have been produced by T_1, while a token in P_5 imposes the firing of T_3 which corresponds to a (n-1) sequences

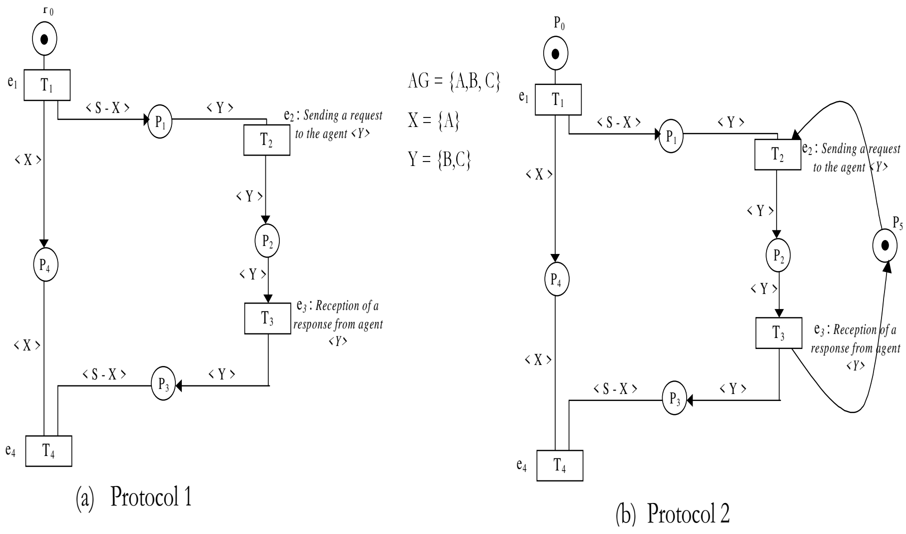

Fig. 3. Two CPN protocols to be recognized

of T_2 followed by T_3 .
Let us now observe a computation of one of the two protocols given through a CG in (figure 4.b). The algorithm presented below develops all the possible process nets (see figure 4.a) in order to recognize the right CPN and the right process. The cycle of our algorithm (Step 1 to 5) is executed iteratively until all the events of CG are not examined.

Step 0: the algorithm begins at the places corresponding to the initial marking of each CPN (P_0 in $Protocol_1$ and (P_0, P_5) in $Protocol_2$). The set of events without predecessors is extracted from the GC (i.e. initially the only event ($e_1(A)$)).

Step 1-2: For each of the expected events (here $e_1(A)$), the algorithm tries to recognize while firing the transitions labeled by these events concurrently in the two CPNs. In our example, only the transition T_1 labeled by $e_1(A)$ is fired both in $Protocol_1$ and $Protocol_2$. Consequently, the event is recognized by the two protocols and the output places are created and linked accordingly.

Step 3-4: *Step 3* checks that the causal dependency of the recognized events through the process net is the same one as the CG. In the contrary case, the corresponding process net is rejected. When the transition labeled by an event is not fireable (*Step 4*) the associated process net is rejected. This is the case if the $Protocol_2$ for the transitions $T_2(A)$ and $T_2(B)$.

Step 5: The set of events without predecessors is updated by removing the events already examined and adding new ones, i.e. their successors (of course, only those without predecessors).

Step 6: The algorithm fails because all the developed process nets are eliminated.

Step 7: if the CG has been covered by the algorithm, it is necessary to check that the obtained process net is maximal, i.e. no transition can be fired. Otherwise, the protocol has not been executed completely.

The algorithm

Inputs:
The causal graph $CG = (E, U)$: *Where E is the set of events and U denotes the Causal dependency relation*
CPN_Patterns = $\{\sum_i / \sum_i = (CPN_i, M0_i)\}$: *The set of CPN-Patterns available in our library of protocols*
Output:
The set of expected unfoldings (i.e. process nets)
Begin
Unfold_Nets:=CPN_Patterns
$Ewp = \{e \in E/e$ is without predecessor$\}$
$MC_i = M0_i$: *the current marking of $\sum_i$ initialized with the initial marking*
1: **while** $Ewp \neq \emptyset$ **do**
for_all e_j such that $e_j \in Ewp$ **do**
2: **for_each** $\sum_i \in$ Unfold_Nets **do**
if $\forall t_{k,i} in \sum_i / e_j$ is labeled by $t_{k,i}$ and is fireable from MC_i **then**
$MC_i := MC_i \oplus Post(t_{k,i}) \ominus Pre(t_{k,i})$: *update the marking after firing $t_{k,i}$*
3: **if** not (Causal-Dependency-Satisfied (CG,Ewp)) **then**
Unfold_Nets:= Unfold_Nets $\backslash \{\sum_i\}$: *eliminate $\sum_i$ unfolding: causal dependency violation*
end_if
else
4: **if** $(\exists t_{k,i}/e_j$ is labeled by $t_{k,i}$ and $t_{k,i}$ is not fireable with $MC_i)$ **then**
Unfold_Nets:= Unfold_Nets $\backslash \{\sum_i\}$: *abort unfolding : transition $t_{k,i}$ cannot be fired*
end_if
end_if
end_for_each
end_for_all
5: $Ewp = Ewp \setminus \{e_j\} \cup \{e' \in E$ such that: $e' = succ(e_j)$ and e' is without predecessor$\}$
end_while
6: **if** Unfold_Nets=$\emptyset$ **then** return FAILURE : *No unfolding net found*
else
7: **for_each** $\sum_i \in$ Unfold_Nets **do**
if there is no more transition $t_{k,i}$ fireable with the marking MC_i **then**
return SUCCESS: $\sum_i$ *matches with CG*
else
Unfold_Nets:= Unfold_Nets $\backslash \{\sum_i\}$: *The unfolding of $\sum_i$ is not maximal*
end_if
end_if
End

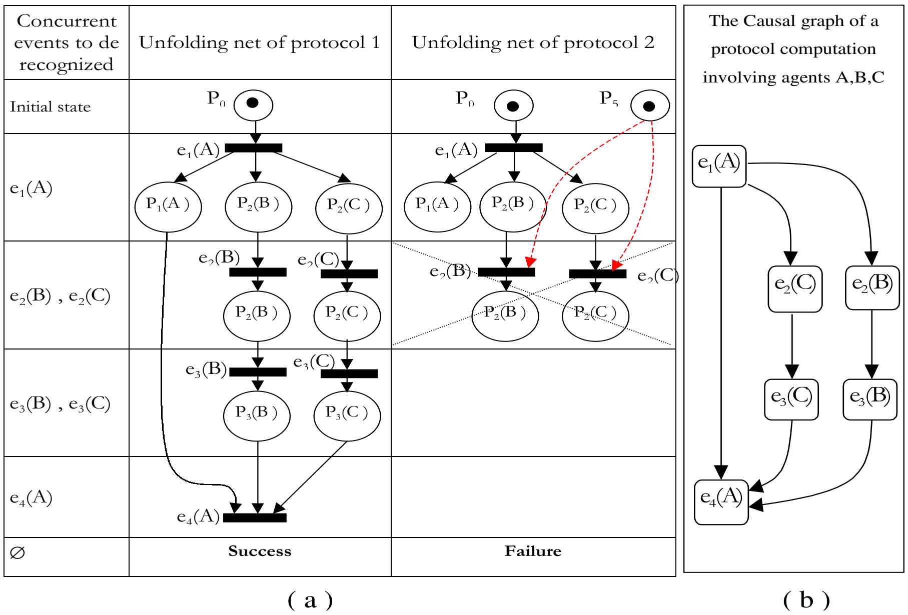

Fig. 4. The unfolding process of the two protocols according to the CG

6 Conclusion and future work

This paper proposes an original approach for protocol engineering in the case of complex interactions. The main advantages of our approach may be summarized as follows:

- It is generic, i.e. independent of any communication protocols and languages;
- It enables a formal study of complex interactions (i.e. modeling, analysis and verification) through a suitable formalism namely the Colored Petri Nets;
- Our algorithm is based on the partial-order semantics of Petri Nets unfolding (i.e. true concurrency) and enables, at the opposite of the interleaving concurrency semantics, the partial order representation of concurrent behaviours;
- Based on distributed observation, our approach develops the causal graph of events (the most relevant ones) w.r.t. the local concurrency and hence it addresses the problem of *false causality*;
- Finally, it allows not only the detection of the success/failure situations but also the explanation of such situations.

Our future work, intends to use the results of the analyzed situations analysis, and consequently the evaluation that it provides regarding to a set of interactions, in the following way: a learner agent recovers these results and generates a qualitative criteria to be communicated to other agents in order to enrich their social knowledge and improving their future interactions.

References

1. J.L. Austin:How To Do Things With Words. Oxford University Press. (1962).
2. H. Bachaténe, M. Coriat and A. El Fallah Seghrouchni: Using Software Engineering Principles to Design Intelligent Cooperative Systems. In: Proc. of SEKE'93 (KSI Press. San Fransisco, USA. (1993).
3. R. Boubour: Suivi de pannes par corrélation causale d'alarmes dans les systèmes réparties: Application aux réseaux de Télécommunications. Theses l'université de Rennes 1, Oct, 1997.
4. P.R. Cohen and H.J. Levesque: Communicative actions for artificial agent. In Proceedings of the First International Conference On Multi-agent Systems (ICMAS'95), San Francisco, CA (1995).
5. A. El Fallah Seghrouchni, S. Haddad and H. Mazouzi: Etude des interactions basée sur l'observation répartie dans un systéme multi-agents. In Proceedings of JFI-ADSMA'98, Eds Hermés. Nancy, Novembre (1998).
6. A. El Fallah Seghrouchni, S. Haddad and H. Mazouzi: A Formal Study of Interactions in Multi-Agent Systems. To appear in Proc. of CATA'99. April, Cancun, Mexico.
7. J. Esparza, S. Romer, and W. Volger: An improvement of McMillan's unfolding algorithm. In Proc. of the second international workshop TACAS'96, volume 1055 of LNCS, pp. 87-106, Passau, Germany, March 1996. Springer Verlag.
8. Foundation for Intelligent Physical Agents: FIPA 97 Specification. Part 2, Agent Communication Language, (1997). http://www.cselt.stet.it/ufv/leonardo/fipa/index.htm.
9. J. Fidge: Timestamps in message passing systems that preserve the partial ordering. In Proc. 11th Australian Computer Science Conference,(1988), 55-66.
10. K. Jensen and G. Rozenberg: High Level Petri Nets, Theory and Applications. Springer-Verlag (1991).
11. DARPA Knowledge Sharing Initiative External Interfaces Working Group: Specification of the KQML Agent-Communication Language - plus example agent policies and architectures, http://www.cs.umbc.edu/agents/kse/kqml/ (1993).
12. J.L. Koning, G. Franois and Y. Demazeau : Formalization and pre-validation for interaction protocols in multi agent systems, 13th European Conference on Artificial Intelligence, Brighton, (1998).
13. L. Lamport: Time, Clocks, and the ordering of events, in distributed system. Communication of the ACM. Vol. 21, Num. 7 (1978) pp. 558-565.
14. F. Mattern: Virtual time and global states of distibuted systems. In. Proc. of the Workshop on Parallel and Distributed Algorithmes, Bonas, North Holland (1988).
15. H. Mazouzi: Formal Study of Interactions based on Distributed Observation in Multi-Agent Systems. Technical report, (1999) htpp://www.lamsade.dauphine.fr/∼mazouzi
16. K.L. McMillan: On-the-fly verification with stubborn sets. In Proc. of Computer Aided Verification, vol. 663 of LNCS, 164-175, Montreal, June 1992. Springer Verlag.
17. M. Niellsen. G. Plotkin and G. Winskel: Petri Nets, Event Structures and Domains. Theoretical Computer Science (13)1, pp. 85-108 (1980).
18. V. Parunak: Visualizing Agent Conversations: Using Enhanced Dooley Graphs for Agent Design and Analysis. In. Proc. of the 2nd ICMAS (1996) 275-282.
19. Michel Raynal: Logical Time : A way to capture causality in distributed systems. Technical Report INRIA-2472, (1995).
20. J.R. Searle: Speech Acts. Cambridge University Press (1969).
21. A. Tarafdar and V. K. Garg: Adressing False Causality while Detecting Predicates in Distributed Programs. Proceedings of the IEEE 18th ICDCS, pages 94 - 101, Amsterdam, Netherlands, May 1998.