A Formal Study of Interactions in Multi-agent Systems

A. El Fallah-Seghrouchni
LIPN - Universit Paris Nord
Avenue J-B Clment
93430 Villetaneuse, FRANCE
elfallah@lipn.univ-paris13.fr

S. Haddad and H. Mazouzi
LAMSADE - Universit paris Dauphine
Place du Marchal de Lattre de Tassigny
75775 Paris Cdex 16, FRANCE
{haddad , mazouzi}@lamsade.dauphine.fr

Abstract

This paper presents an original approach to model, analyze, and design interactions in multi-agent systems. It combines two complementary paradigms: observation in distributed systems and interaction in multi-agent systems. The first paradigm is frequently used to observe concurrent activities in distributed systems through the causal dependency of events. The second paradigm aims to identify interaction-oriented designs and describes them with a formalism enabling to prove their quality. Our approach is based on distributed observation of events inherent to agents' interactions, which may explain relationships within conversations, or group utterances in order to improve agent's behavior.

Keywords: *causality, interaction, distributed observation, performatives, colored Petri nets.*

1 Introduction

Multi-agent systems are currently a major area of research and development activity. An agent is a computational entity, more or less autonomous, which implies a problem solving that both perceives and acts upon the environment in which it is situated. A multi-agent system is defined as a set of agents, possibly heterogenous, that communicate, cooperate, coordinate, and negotiate with one another to advance both their individual goals that interact and the global goal of the overall system. A crucial component of this paradigm is modeling interactions between agents as well as the development of standard agents' communication languages (ACLs) [7, 8].

This paper proposes an approach to study agents' interactions as essential components of the dynamic of multi-agent systems to improve individual behavior of the agents and naturally the one of the multi-agent system. Our approach is generic, i.e. independent of any communication language, and combines two complementary paradigms: distributed observation to capture the events underlying the interacting situations and the colored Petri net (CPN) [9] as an efficient formalism to specify, model and study several kind of communication protocols.
This paper is organized as follows: we begin in section 2 by outlining basic elements of our framework relative to the background of activities in interaction issues. We then describe our general framework as well as our hypothesis and our aims namely the modeling and the analyzing of interactions in section 3 and we briefly present the paradigm of distributed observation through the causality concept in section 4 and the technique we adopt (i.e. Vector Clocks) to carry out the causally graph. Section 5 details our model of interaction following the steps developed in our approach: 1) how to build the causally graph of interactions, 2) modeling aspects of protocols by means of CPN. Section 6 focuses on the recognition process, which lead to recognize interactions from causally graph. Section 7 concludes and lights up our future work.

2 The interaction issue

2.1 Interacting situations

In a multi-agent system, agents are often led to communicate together in order to cooperate, to achieve common tasks and goals, to exchange data, knowledge, and plans, etc. A common paradigm for agents' communication is the message passing. A such paradigm is the necessary mean for cooperation and it requires a shared background of agents' skills as complex as perception, learning, planning and reasoning. With distributed problem solving, interaction involves complex strategies of cooperation in the sense that they are non-deterministic, hard to be interpreted and sometimes not completely reproducible nor predictable. The balance between the autonomy of agents, endowed with more or less intelligent behavior, and their convergence towards a global goal may be considered as the most significant characterization of interaction. Several types of interaction has been considered by researchers [5] and analyzed through different components including the nature of the goals, the access to resources, and the skills of agents. Consequently, a first typology of interacting situations may be identified: autonomy, a simple cooperation, congestion, coordinated cooperation, individual or collective competition, individual or collective conflict, etc. Moreover, interacting situations may be analyzed through hierarchical point of view. In fact, a complex interacting situation is composed of several elementary situations. In other words, macro-situations may be distinguished during the analyze of the global activity of the agents while micro-situations may be observed at the most fine levels of details.

2.2 Interaction languages in multi-agent systems

To support inter-organizational interaction, communication and cooperation in multi-agents systems, many frameworks towards standardization for formalizing the flow of interaction between agents have been proposed [7, 8, 17]. These frameworks intend to develop a generic interaction language by specifying messages and protocols for inter-agent communication and cooperation. One of the major interests of interaction languages is to reduce the communication cost by avoiding an exhaustive description of the ad hoc messages and to offer a large scale of protocols [10]. Such languages focus especially on how to describe exhaustively the speech acts [1, 19] both from the syntactic and semantic point of views 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, mainly developed in USA, includes Knowledge Sharing Effort (KSE) that outputs specification for the Knowledge Querying and Manipulation Language (KQML) [8, 11] and the Knowledge Interchange Format (KIF), and specifications of ontologies. KQML proposes an extensible set of performatives, which defines the permissible operations that agents may attempt on each other's knowledge and goal stores. KQML is based on the speech act theory introduced at first by Austin [1] and developed later by Searle [19] in order to allow cognitive agents to cooperate. Based on the possibility to encode explicitly in the messages themselves illocutionary acts in terms of messages or *performatives*, it lies on the mental states of the agents [10, 3, 5].
Nevertheless, some observations about KQML has been recently pointed out i.e some performatives are ambiguous, while others are not really performatives at all, and there are no performatives that commit an agent to do something. An other criticism that can be made to KQML is its deficiency regarding to a clear semantics independent of the agents' structure [3]. In fact, KQML offers possibilities for isolated communications when handling of complex interactions needs sophisticated protocols. Consequently, the communication specification through KQML imposes agents to shape their behavior according to an architecture that implements and supports a theory based on mental states.
In [10], an interaction language has been proposed. It introduces generic protocols of interaction devoted to be instanciated depending on the social behaviors to be implemented. The major interest of such language consists in the fact that it encapsulates the application level. It offers several protocols of communication as well as the multi-agent language. Nevertheless, this language raises the modeling problem. In fact, the specification of protocols by means of the finite state automaton induces some deficiencies when we have to study complex interactions.
More recently, the international collaboration of member organizations, which are active companies and universities in the agent field within FIPA (*Foundation for Intelligent Physical Agents*), proposes and specifies some standards for the agent technology and especially, an agent communication language namely ACL (*Agent Communication Language*) [7]. ACL is also based on the speech act theory: the messages are considered as acts or communicative acts reflecting the act (action) expected by the speaker as result of sending message. ACL recovers the syntactical idea of KQML which allows to build interactions enriched by a powerful semantics of performatives and consequently enables the expression of a set of high level protocols and primitives to control the information exchange between agents. Although KQML and ACL of FIPA are similar at the syntactical level, they suggest substantially differing views on the issue of agent communication.

2.3 Modeling Interactions

In order to model interactions, usually expressed by means of interaction protocols, several formalisms have been used. For instance, the model proposed in [17] is a graph of predefined states where an agent evolves according to the kind of recieved messages. Other models like automata, graph or more specific graphs like the Dooly-graph [16] are used to describe the conversations between agents. These models of representation are very practical when they are used to precise some conversational structures and especially when they appear as isolated communications. The problem to be faced with such formalisms, is their poor capacity of computing (i.e. handling protocols) or of the representation of complex protocols. From one hand, the finite number of the graph states reduces the capacity to represent real and complex protocols. From the other hand, the most formalisms used take in charge only sequential processes. Hence, these models are very limited when it is useful to take into account the concurrency which is the key stone of multi-agent systems since agents are often involved simultaneously in several conversations or more generally in several interactions.
From our point of view, it should be useful to formally describe interactions by means of models that are suitable to specify concurrent systems such as colored Petri nets (CPN) [9] which naturally represent the concurrency and make easy the processing factorization.

3 Conceptual framework

Agents that operate in a multi-agent system need an efficient strategy to handle their concurrent activities in a shared and dynamic environment. Searching for an optimal interaction strategy is a hard problem to be solved by an agent because it depends mostly on the dynamic behavior of the other agents. One way to deal with this problem is to endow the agents with the ability to adapt their strategies according to their experience.
Our framework hypothesis are as follows:

- The multi-agent systems considered are composed of a set of cognitive agents distributed on different sites and maybe running concurrently.

- The agents communicate exclusively by asynchronous messages.

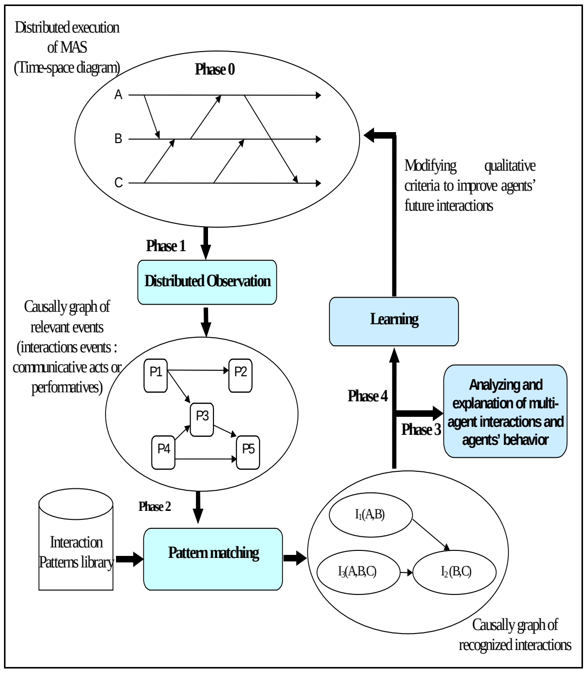

Distributed execution of MAS (Time-space diagram)
Phase 0
A
B
C
Modifying qualitative criteria to improve agents' future interactions
Phase 1
Distributed Observation
Causally graph of relevant events (interactions events : communicative acts or performatives)
P1
P2
P3
P4
P5
Learning
Phase 4
Phase 3
Analyzing and explanation of multi-agent interactions and agents' behavior
Phase 2
Interaction Patterns library
Pattern matching
I1(A,B)
I3(A,B,C)
I2(B,C)
Causally graph of recognized interactions

Figure 1: A structured approach for multi-agent system design

- The observation is distributed in the sense that each agent observes and traces observable events (corresponding to the emissions/receptions of the agent messages).

- The analysis and learning process is ensured *off-line* by a single agent which centralizes the traces of the other agents.

Hence, our approach may be summarized as follows (cf. Figure 1) : starting from a distributed execution (i.e. phase 0), the first phase corresponds to the distributed observation in order to capture the traces of events underlying to interactions.
The second phase corresponds to the recognition process of interactions based on a pattern matching mechanism applied to the causally graph. Let be remarked that the filters used to recognize the patterns of interaction are defined as CPN which model interaction protocols.
The third phase exploits the traces of interactions, obtained by observation, in order to explain the behavior of the agent.
The fourth phase will not be detailed here. It is based on a central agent, which learns and deduces a qualitative criteria for the improvement of the behavior of the other agents.

4 Causal dependencies in multi-agent interactions

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 execution are relevant (e.g., send and receive messages). An observer of a multi-agent system may be any entity that attempts to watch the system "live", while the computation is in progress, or examine a post-mortem events' log or trace. In all the cases, it is necessary to inform the observer whenever interesting events occur.
In multi-agent systems, three types of event are involved during a computation, namely, *internal* event, *message send* event, and *message receive* event. An internal event only affects locally the agent at which it occurs and are linearly ordered by the order of their occurrence. Moreover, send and receive events signify the flow of information between agents and establish causal dependency from the sender agent to the receiver agent. It follows that the execution of the multi-agent application results in a set of distributed events produced by the agents. The *causal precedence* relation induces a partial order on the events of a distributed computation.
Thus, Causality (or the causal precedence relation) among events is a powerful concept in reasoning, analyzing, and drawing inferences about a computation. The knowledge of the causal precedence relation among agents helps studying the variety of interaction happened during multi-agent system computation.

4.1 A model of distributed computation

In our framework, we consider a distributed execution as composed of a set of n asynchronous agents $A_1, A_2, \ldots, A_n$ that communicate by message passing over a communication network [18]. In addition, the agents do not share a global memory or global clock.
At the most abstract level, execution of a multi-agent system can be defined as a set of events denoted E. The events happened at the same agent are naturally ordered. Thus, there is a *total* order of events in a sequential system. Observe that one of the major difficulties in distributed agents is that the relation between events is only a *partial* order.
Leslie Lamport [12] recognized the importance of the ordering relation between events in a distributed system. He postulated that this relation (which he called happened-before) is transitive and not reflexive. We describe this relation in the next section.
Each agent A_i generates an execution trace, which is a finite sequence of local atomic events. There are two kinds of events produced by the multi-agents' execution :

- interaction events, i.e. sending and receiving messages.
- internal events of the agents i.e., all events other than sending and receiving messages (updating internal state of the agent, performing local actions etc.).

4.2 Events' causality in interactions

The time concept in distributed systems may be usefully used in many ways, especially for ordering events. We say that an event a happened before an event b if the physical time of the event a is less than the time at which b is happened. In the absence of precisely synchronous clocks in distributed systems, the use of logical clocks is useful for keeping information about causality rather than physical time. Other aspect is that many applications require identifying "*cause and effect*" relationships in event occurrences. In any case, it is necessary to construct mechanisms that give information about *causally precedes* relation among the events in the distributed computation.
This relation can be defined as:

- *Locally precedes* relation between events of a single agent,
- *Immediately precedes* relation between coupled events e and f of exchange messages if e is the send event of the message and f is the receive event of the same message.

Now, the causally precedes relation denoted by "$\rightarrow$" can be defined as the transitive closure of the union of *locally precedes* and *immediately precedes* relations.
Clearly, for any two events e_1 and e_2 in a distributed execution, $e_1 \rightarrow e_2$ or $e_2 \rightarrow e_1$, or $e_1 || e_2$ (concurrent events).

This relation can be represented with an oriented graph $G = (X, U)$ denoted causal dependency graph where:

- X is the set of nodes representing the events,
- $U \in X \times X$ is the set of oriented edges in the form (x, y) which indicates that x *precedes causally* y.

Thus, the graph G associated with the partial order relation "$\rightarrow$" in the domain E is defined as follow:

$$\forall x, y \in E : x \rightarrow y \Leftrightarrow (x, y) \in U$$

Finally, the causality can be accurately captured bu logical clocks. Many logical clocks are implemented, depending on the level of information required, intending to obey the fundamental monotonic property; that is, if an event a causally affect an event b, then the timestamp (date assigned to an event) of a is smaller than the timestamp of b. In order to know exactly the causally *precedes relation* between events, we use the vector clocks developed by [6, 13].

5 Interaction Model

Many details of the behavior of agent-based systems cannot be predicted analytically, but must be observed. A *causally precedes graph* of the interactions that emerge can provide the basis of a number of quantitative measures that are relevant to the system strategies. Similarly, useful measures and meaningful aspects of the dynamic of the conversations can be derived from the causally graph (e.g., deriving from the lengths of various paths reply cycles and series of performatives).

5.1 Distributed Observation of multi-agent interactions

The point of interactions is that a conversation, can be explored both from the perspective of the performatives it contains (usefully distinguished as events) and states through which it passes. The performatives' causally graph representation enables us to deduce causally relation between interaction instances. At run-time, an agent may be involved in several protocol instances. In order to record the state of every instance of a protocol, to which the agent is committed, the agent must record all the necessary data regarding to the protocol as well as the events' timestamps. In the following sections, we consider only complex interactions since the elementary ones may be viewed as trivial cases of interaction. The following example shows how to build the causally graph from a communication protocol namely the FIPA contract-net [7].

Example. FIPA-Contract-Net Protocol
This section presents a version of the widely used *Contract Net Protocol*, originally developed by Smith and Davis [21]. FIPA-Contract-Net is a minor modification of the original contract net protocol in that it adds *rejection* and *confirmation* communicative acts. In the contract net protocol, one agent takes the role of *manager*. The manager wishes to have some task performed by one or more other agents, and further

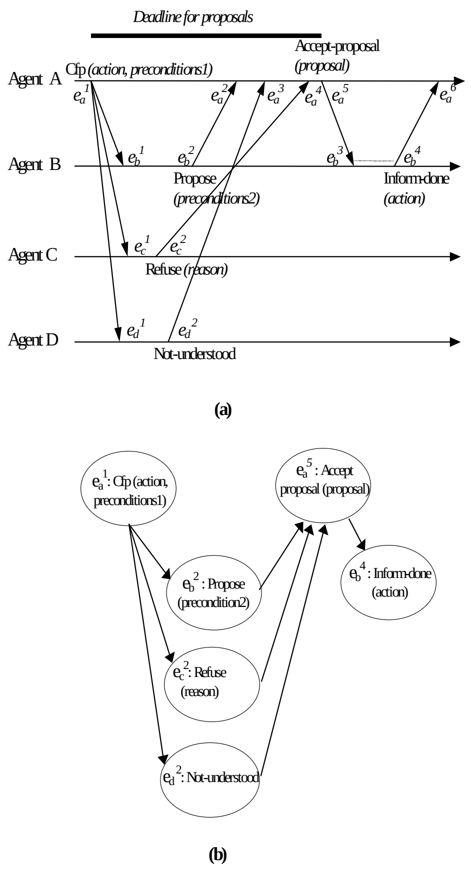


Figure 2: (a) A possible execution of FIPA-Contract-Net protocol ; (b) Causally graph inherent to performatives of FIPA-Contract-Net protocol

wishes to optimize a function that characterizes the task (commonly expressed as the price, time to completion, etc.). The manager demands *proposals* from other agents by issuing a *call for proposals*, which specifies the task and any conditions the manager is placing upon the execution of the task. Agents receiving the call for proposals are viewed as potential *contractors*, and are able to generate proposals to perform the task as *propose* acts including the preconditions which may be the price, time when the task will be done, etc. Alternatively, the contractor may *refuse* to propose. Once the manager receives back replies from all of the contractors, it evaluates the proposals and makes its choice of which agents will perform the task. One, several, or no agents may be chosen. The agents of the selected proposal(s) will be sent an acceptance message, the others will receive a notice of rejection. The proposals are assumed to be binding on the contractor, so that once the manager accepts the proposal the contractor acquires a commitment to perform the task. Once the contractor has completed the task, it sends a completion message to the manager. Note that the protocol requires the manager to know when it has received all replies. In the case that a contractor fails to reply with either a *propose* or a *refuse*, the manager may potentially be left waiting indefinitely. To guard against this, the *cfp* includes a deadline by which replies should be received by the manager. Proposals received after the deadline are automatically rejected, with the given reason that the proposal was late.

Figure 2 illustrates the causally precedes relation between observable events through an execution of FIPA-Contract-Net protocol. The causal graph associated to the performatives' occurrence is constructed by using vectors clocks (timestamps) of Fidge and Mattern [6, 13](cf. Appendix) .

5.2 Modeling interactions by means of Colored Petri Nets

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. We consider interaction protocols as the basic ones from which complex and high-level interactions can be built [2]. As argued in section 2.3, we adopt the CPN formalism to handle interactions. Next, we present the syntactical and semantical features of CPN [9] illustrated through a significant example, FIPA-Query-Protocol [7] (see figure 3).

5.3 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.

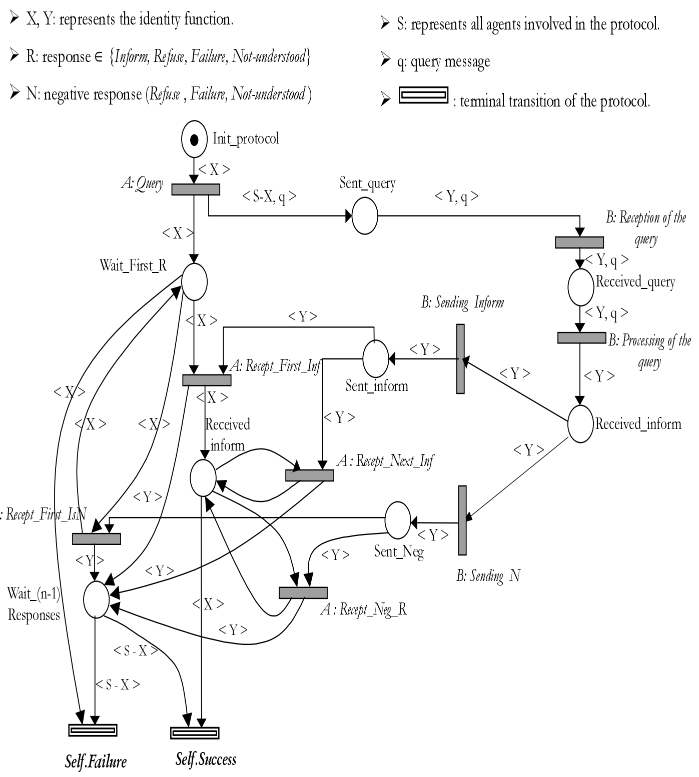


Figure 3: The CPN model of FIPA-Query-Protocol

- 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).

5.4 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. $AG \setminus \{A_i\}$.
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.

6 Pattern matching based on unfolding Petri Nets

Our aim is to represent two aspects in our model: The first expresses serial and concurrence events to be observed, i.e., the interaction states achieved by agents; the second aspect describes the causally precedence that exists among communicative acts occurred during computation.
The next stage consists in carrying out interaction protocols by recognizing them. The recognition of interactions is provided by a pattern matching algorithm (or filtering) where filters are available as CPN protocols library.
Our algorithm is based on the partial-order semantics of Petri Nets and well-known as unfoldings of Petri Nets [15]. 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.

6.1 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 5). 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 [14] [4] 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.

6.2 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,

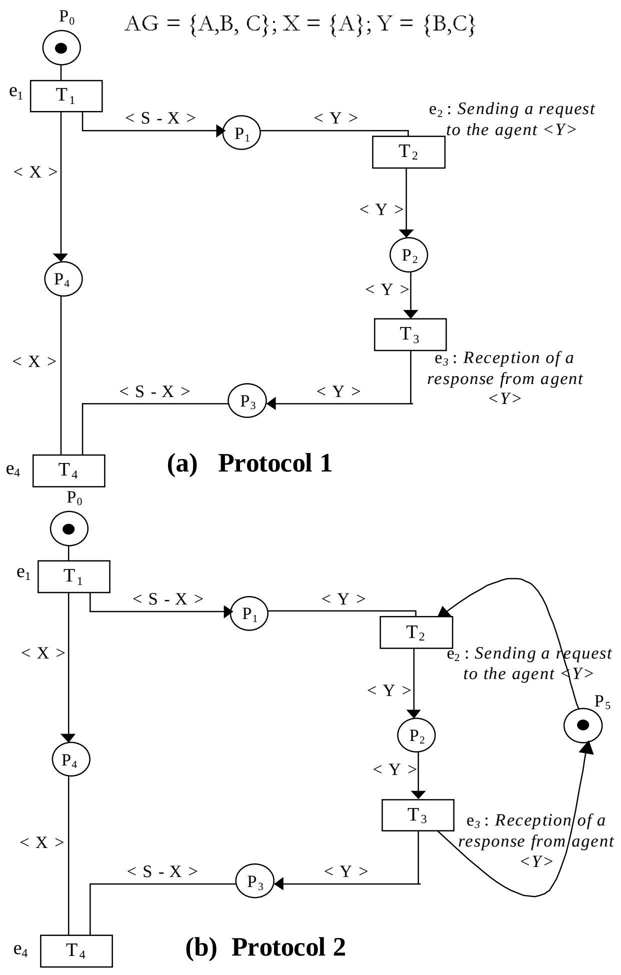

AG = {A,B, C}; X = {A}; Y = {B,C}
P0
e1
T1
< S - X >
P1
< Y >
e2 : Sending a request to the agent <Y>
T2
< X >
< Y >
P2
P4
< Y >
T3
e3 : Reception of a response from agent <Y>
< X >
< S - X >
P3
< Y >
e4
T4
(a) Protocol 1
P0
e1
T1
< S - X >
P1
< Y >
T2
e2 : Sending a request to the agent <Y>
< X >
< Y >
P5
P2
P4
< Y >
T3
e3 : Reception of a response from agent <Y>
< X >
< S - X >
P3
< Y >
e4
T4
(b) Protocol 2

Figure 4: Two CPN protocols to be recognized

in several interactions [?]. 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 ??). 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 4) 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 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 ($e_1(A)$)).

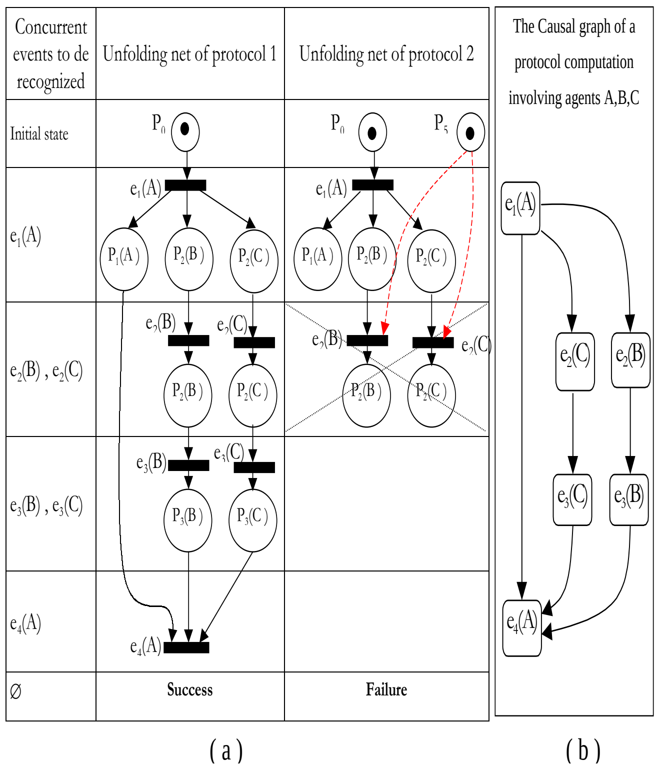
Step 1-2: For each of the expected events (here $e_1(A)$), the algorithm tries to recognize while firing the transitions labeled by these events concurrently in the two CPNs. In our example, only the transition T_1 labeled by $e_1(A)$ is fired both in $Protocol_1$ and $Protocol_2$. Consequently, the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



Concurrent events to de recognized
Unfolding net of protocol 1
Unfolding net of protocol 2
The Causal graph of a protocol computation involving agents A,B,C
Initial state
$e_1(A)$
$e_2(B)$, $e_2(C)$
$e_3(B)$, $e_3(C)$
$e_4(A)$
∅
Success
Failure
(a)
(b)

Figure 5: The unfolding process of the two protocols according to the CG

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 $\setminus \{\sum_i\}$: *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 $\setminus \{\sum_i\}$: *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 $\setminus \{\sum_i\}$: *The unfolding of* $\sum_i$ *is not maximal*
            end_if
      end_if
End
```

7 Conclusion and future work

In this paper, we have proposed an approach based on distributed observation and a formal method for designing interaction protocols in multi-agent systems. It provides a very powerful and useful way to analyze and explain progress, that exhibits interesting properties. The concept of causality between events is fundamental to design and analyze distributed computation. Represented through partial order, the causality may be implemented using the timestamps tools. When applied to the communication events, causality enables to emphasize the correlation between both elementary and complex interactions in order to recognize and evaluate the interacting situations. This paper also proposes an original approach for modeling, studying and analyzing interactions by means of Colored Petri

Nets.
This formalism is suitable for concurrent activities and it offers several formal methods to study the dynamic of complex systems. In addition, our approach is generic and may be applied to a large scale of communication protocols and languages.
The analysis of multi-agent system interactions, in particular the observation of the process of concurrent execution, allows not only the detection of the success/failure situations but also to explain the reasons of such situations. In fact, the explanation lies on the causally graph and allows the backtracking if necessary.
Our future work, intends to use the results of the distributed observation and consequently the evaluation that it provides regarding to a set of interactions, in the following way: a learner agent recovers these results, generates a qualitative criteria to be communicated to other agents in order to enrich their social knowledge and to improve their future interactions.

References

[1] J.L. Austin. How to do things with words. *Oxford University Press*, 1962.

[2] H. Bachatne, M. Coriat, and El Fallah Seghrouchni. Using software engineering principles to design intelligent cooperative systems. *In the Proc. of SEKE'93 (KSI Press. San Fransisco, USA*, 1993.

[3] P.R. Cohen and H.J. Levesque. Communicative actions for artificial agent. *Proceedings of the First International Conference On Multi-agent Systems (ICMAS'95), San Francisco, CA*, 1995.

[4] J. Esparza, S. Romer, and W. Volge. An improvement of mcmillan's unfolding algorithm. *Proc. of the second international workshop TACAS'96, volume 1055 of LNCS, pp. 87-106, Passau, Germany, Springer Verlag.*, 1996.

[5] J. Ferber. *Les systmes Multi-Agents. Vers une Intelligence Collective.* Inter-Edition, 1995.

[6] J. Fidge. Timestamps in message passing systems that preserve the partial ordering. *In Proc. 11th Australian Computer Science Conference*, pages 55–66, February 1988.

[7] Foundation for Intelligent Physical Agents. Fipa 97 specification. part 2, agent communication language. 1997.

[8] DARPA Knowledge Sharing Initiative External Interfaces Working Group. Specification of the kqml agent-communication language - plus example agent policies and architectures. 1993.

[9] K. Jensen and G. Rozenberg. *High Level Petri Nets, Theory and Applications.* Springer-Verlag, 1991.

[10] J.L Koning, Y. Demazeau, B. Esfandiari, and J. Quinqueton. Quelques perspectives d'utilisation des langages et protocoles d'interaction dans le contexte des telecommunications. *3mes Journes Francophones sur l'Intelligence Artificielle Distribue et les Systmes Multi-Agents, Chambry, France*, 1995.

[11] Y. Labrou and T. Finin. A proposal for a new kqml specification. *Technical Report CS-97-03*, 1997.

[12] L. Lamport. Time, clocks, and the ordering of events, in distributed system. *Communication of the ACM*, 21(7):558–565, 1978.

[13] F. Mattern. Virtual time and global states of distibuted systems. *Proc. of the Workshop on Parallel and Distributed Algorithmes,Bonas, North Holland*, 1988.

[14] K.L. McMillan. On-the-fly verification with stubborn sets. *Proc. of Computer Aided Verification, vol. 663 of LNCS, PP. 164-175, Montreal, Springer Verlag*, June 1992.

[15] M. Niellsen, G. Plotkin, and G. Winskel. Petri nets, event structures and domains. *Theoretical Computer Science (13)1, pp. 85-108*, 1980.

[16] V. Parunak. Visualizing agent conversations: Using enhanced dooley graphs for agent design and analysis. *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, pages 275–282, 1996.

[17] P. Populaire, Y. Demazeau, O. Boissier, and J. Sichman. Description et implmentation de protocoles de communication en univers multi-agents. *1res Journes Francophones Intelligence Artificielle Distribue et systmes multi-agents, Toulouse*, April 1993.

[18] Michel Raynal. Logical time : A way to capture causality in distributed systems. *Rapport de recherche INRIA*, (2472), 1995.

[19] J.R. Searle. Speech acts. *Cambridge Universit Press*, 1969.

[20] A. El Fallah Seghrouchni and H. Mazouzi. Proto col engineering for multi-agent interaction. *F.J Garijo, M. Boman (eds.), Multi-Agent Systen Engineering, Proceedings of the 9th Europeai Workshop on Modelling Autonomous Agents ii a Multi-Agent World, MAAMAW'99. Lectur Notes in AI, vol. 1647, Springer Verlag.*, 1999.

[21] R.G. Smith. The contract net protocol : High level communication and control in a distributec problem solver. *IEEE Trans. On Computers* 29(12):1104–1113, 1980.

Appendix : Vector clocks

The partial order between observed events is recon structed through timing information associated witl each event captured by the observer. However, in th system of vector clocks, the time domain is representec by a set of n-dimensional non-negative integer vectors Each process P_i maintains a vector $V_i[1..n]$ where $V_i[i$ is the local logical clock of Pi and describes the logica time progress at process P_i. $V_i[j]$ represents proces P_i's latest knowledge of process P_j local time.
Each process P_i executes the following algorithm:

```
Data :
$V_i[1..n]$ vector of integers ;
Initialisation :
$\forall j \in 1..n : V_i[j] := 0$ ;
Rule for an internal event x produced by $P_i$ :
$V_i[i] := V_i[i] + 1$ ;
Rule for a send event ( message from $P_i$ to $P_k$ ) :
$V_i[1..n]$ is included in the message ;
Rule for a receive event ( message from $P_k$ to $P_i$ in
cluding $V_k$ ):
$\forall j \in 1..n$ : Do
If $V_i[j] < V_k[j]$ then $V_i[j] := V_k[j]$ ;
```

This algorithm is described by the initial condi tions and the actions taken for each event. The al gorithm also satisfies the following property : for an two events x and y ,

$$x \to y \Leftrightarrow \mathit{timestamp}(x) < \mathit{timestamp}(y)$$

The following three relations are defined to compar two vector timestamps, v_x and v_y:

$$v_x \le v_y \Leftrightarrow \forall i : v_x[i] \le v_y[i]$$

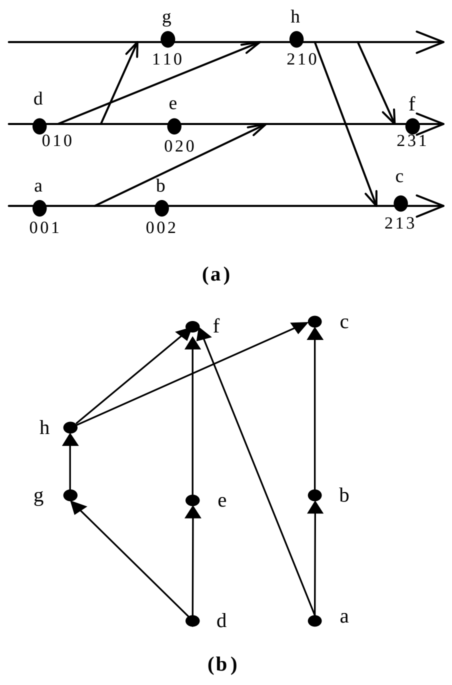


Figure 6: (a) A simple execution of the vector clock algorithm; (b) Associated causally precedes relation

$$v_x < v_y \Leftrightarrow v_x \le v_y \; and \; \exists i : v_x[i] < v_y[i]$$

$$v_x \parallel v_y \Leftrightarrow \; not \, (v_x < v_y) \; and \; not \, (v_y < v_x)$$

An example is given in Figure 7.