A Formal Study of Interactions in Multi-agent Systems

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

S. Haddad and H. Mazouzi
LAMSADE - Université paris Dauphine
Place du Maréchal de Lattre de Tassigny
75775 Paris Cédex 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 provides an interesting framework to formalize complex interaction between agents. 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.

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

1 The interaction issue

This paper proposes an approach to study agents' interactions as essential components of the dynamic of multi-agent systems. The benefit we expect is 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) as an efficient formalism to specify, model and study several kind of communication protocols.

1.1 Interacting situations

In a multi-agent system, agents are often led to communicate together in order to cooperate, to achieve a common tasks and/or goals, to exchange data, knowledge, and plans, etc. A common paradigm for agents' communication is the message. A such communication is the necessary mean of cooperation and required a shared background of agents and skills as complex as perception, learning, planning and reasoning. Face to the distributed problem solving, many kinds of interaction appear and involve 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 of 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. Many types of interaction has been considered by researchers [3] and analyzed through several 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 when micro-situations may be observed at the most fine levels of details.

1.2 Interaction languages in multi-agents systems

To support inter-organizational interaction, communication and cooperation in multi-agents systems, many proposals towards standardization for formalizing the flow of interaction between agents have been made these last years. 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 [8]. Such languages focus especially on how to describe exhaustively the speech acts 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.

At the United States, a standard of communication language has been developed. Hence, Knowledge Sharing Effort (KSE) outputs specification for the Knowledge Querying and Manipulation Language (KQML) [6] and [9] 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 [Austin, 62] and developed later by Searle [15]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.

Nevertheless, KQML has recently been pointed out that some perfomatives are ambiguous, while others are not really performatives at all, and there are no performatives that commit an agent to do something [2]. An other criticism that can be made to KQML is its deficiency regarding to a clear semantics independent of the agents' structure. In fact, KQML offers possibilities for isolated communications when the handling of complex interactions needs sophisticated protocols. Consequently, the communication specification through KQML imposes to agents to shape their behavior according to an architecture that implement and support a theory based on mental states [8, 2, 3]. In [8], 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*) [5]. 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 his 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.

1.3 Modeling Interactions

An interaction protocol aims to restrict the different interactions an agent can make with other agents regarding the problems to solve. The specification of interaction protocols is non-trivial process. Several formalisms have been introduced to model interaction protocols. In particular, several models around on the graph model. For instance, the model proposed in [13] 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 the Dooly-graph [12] 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 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 used formalisms take in charge only sequential processes. Moreover, when these formalisms (e.g. Dooly-graph) take into account the temporal aspect, they assume the existence of a global clock. 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 col-

ored Petri nets (CPN) [7] which naturally represent the concurrency and make easy the processing factorization.
This paper is organized as follows: section 2 describes our general framework as well as our hypothesis and our aims namely the modeling and the analyzing of interactions. Section 3 briefly presents the paradigm of distributed observation through the causality concept and the technique we adopt (i.e. Vector clocks) to carry out the causally graph. Section 4 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 5 focuses on the recognition process, which lead to extract interactions from causally graph. Section 6 concludes and lights up our future work.

2 Approach

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 approach is structured as follows: 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 that model protocols extracted from ACL on studying and analyzing interactions.
The third phase exploits the traces of interactions (obtained by observation) in order to explain the behavior of the agent regarding its interactions.
The phase four 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.

Our Hypothesis:

- The multi-agent systems considered are composed of a set of cognitive agents distributed on different sites and may be run concurrently.
- The agents have access to only their local memory and exclusively communicate by asynchronous messages.
- Each agent has a local strategy to solve its own problems, can reason, etc.
- The observation is distributed in the sense that each agent has a local module, which observes and traces the events corresponding to the emissions/receptions of the agent messages.
- The analyze is ensured *off-line* by the central agent which centralizes the traces of the other agents.
- The learning process is also ensured by the central agent.

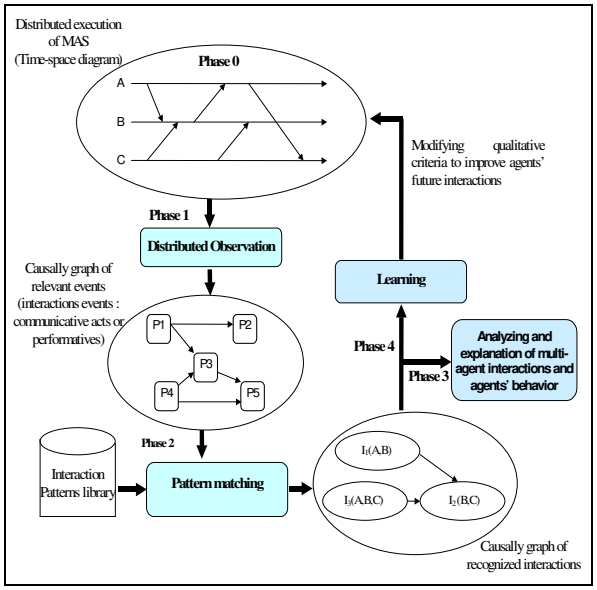

Figure 1: A structured approach for multi-agent system design

3 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 event 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 observing and studies variety of interaction happened in multi-agent system execution.

3.1 A model of distributed executions

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 [14]. In addition, the agents do not share a global memory or global clock. Agent execution and a message transfer are asynchronous - a process may execute an event spontaneously and an agent sending message does not wait for the delivery of the message to be complete.
At the most abstract level, execution of a multi-agent system can be defined as a set of events. The events happened at the same agent are naturally ordered. Thus, there is a *total* order of events in a sequential system. On the other hand, one of the major difficulties in distributed agents is that the relation between events is only a *partial* order.
Leslie Lamport [10] 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. We denote E the union of all events occurred in the multi-agent execution. There are two kinds of event:

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

3.2 Events causality in interactions

The concept of time in distributed systems may be usefully used in many ways, especially for ordering events. We say that event a happened before event b if the physical time of the event a is less than the time at which b is happened. In 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 execution.
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 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.
If $e_1 \rightarrow e_2$, then event e_2 is directly or transitively dependent on event e_1. if not $(e_1 \rightarrow e_2)$ and not $(e_2 \rightarrow e_1)$, then events e_1 and e_2 are said to be concurrent and are denoted $e_1 \| e_2$. 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 $e_1 \| e_2$.

To illustrate the concept of causality, the figure 2. shows the time diagram of a distributed execution (interaction) involving three agents. A horizontal line represents the progress of the agents; a dot indicates an event; a slant arrow indicates a message transfer. The examples we present below are inspired from communication language ACL of FIPA since it offers a clear and rich semantic of communication acts and it supports a set of significant interaction protocols (c.f. Appendix 2)

Example 1 : FIPA-Request-When protocol [5]
The FIPA-request-when protocol is simply an expression of the full-intended meaning of the *request-when* action. The requesting agent uses the request-when

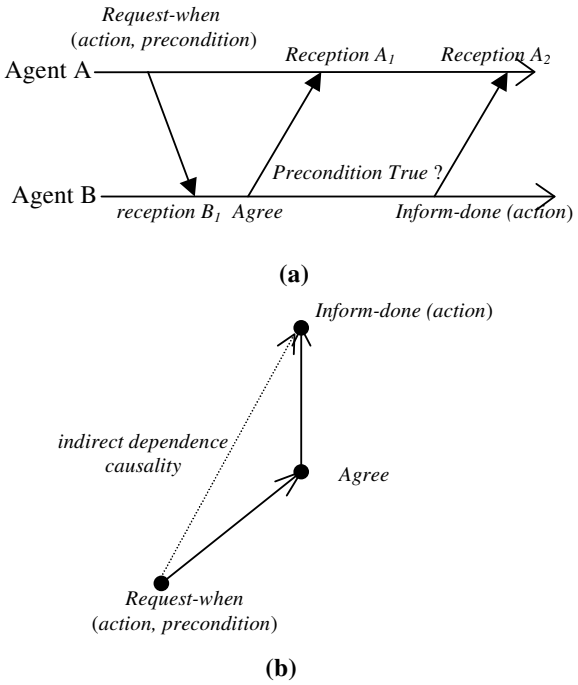

Figure 2: a) The time diagram of FIPA-Request-when protocol execution (with acceptation) ; b) Causally precedes graph related to the performatives

action to seek from the requested agent that it performs some action in the future once a given precondition becomes true. If the requested agent understands the request and does not refuse, it will wait until the precondition occurs then perform the action, after which it will notify the requester that the action has been performed. Note that this protocol is somewhat redundant in the case that the action requested involves notifying the requesting agent anyway. If it subsequently becomes impossible for the requested agent to perform the action, it will send a refuse request to the original requestor.

This example illustrates the causally precedes relation between observable events through the execution of FIPA-request-when protocol. The entire events of an agent are totally ordered (e.g., at agent B: *Agree* $\rightarrow$ *Inform-done (action))*. The send messages event precedes causally their reception events (e.g., *Request-when (action, precondition)* $\rightarrow$ *Receive* B_1). Thus we can deduce the causal order between any couple of events by transitivity. (e.g. *Request-when (action, precondition)* $\rightarrow$ *Inform-done (action)*).

3.3 Causally precedes graph

A causally precedes graph is a very intuitive way of visualizing computation and communications in multi-agent systems. It is powerful and can be analyzed for properties that reflect errors in agent's interactions. Causal order can be represented with an oriented graph $G = (X, U)$ denoted causal dependency graph where:

- X is the set of nodes representing the events in the graph,
- $U \in X \times X$ 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:

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

This graph inherits from the partial order relation the properties of non-reflexivity, transitivity, and anti-symmetry. These three properties induct an a-cyclic graph. We note that this graph is a complete graph of the causality relation and it can be reduced to a minimal graph by eliminating the edges associated with the closer transitive relation of "$\rightarrow$".

3.4 Logical Clock to capture interaction causality

In multi-agent system, progress is made in spurts and interaction between agents occurs in spurts and exhibits concurrency; consequently, it turns out that multi-agent execution and causality relation between events can be accurately captured by logical clocks rather then physical clocks which are not precisely synchronized in distributed systems. In a system of logical clocks, every agent has a logical clock that is advanced using a set of rules. To each event is assigned a timestamp and the causality relation between events can be generally inferred from their timestamps.
Depending on the level of information required, many logical clocks are implemented intending to obey the fundamental monotonic property; that is, if an event a causally affect an event b, then the timestamp 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 independently by [4] and [11]. The main interest of this implementation is that it satisfies an isomorphism between the set of the partially ordered events produced

by the distributed computation and their timestamps. Recall that relation "$\rightarrow$" induces a partial order, if two events x and y have timestamps v_x and v_y, respectively, then:

$$x \rightarrow y \Leftrightarrow v_x < v_y$$

$$x \| y \Leftrightarrow \ not\,(v_x < v_y)\ and not\ (v_y < v_x)$$

This equivalence provides a very powerful, useful, and interesting property of vector clocks.

4 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).

4.1 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 provides us to deduct 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 interactions. The next example shows how to build the causally graph from a communication protocol namely the FIPA contract-net.

Example 2. FIPA-Contract-Net Protocol [5]
This section presents a version of the widely used *Contract Net Protocol*, originally developed by Smith and Davis [16]. 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 wishes to optimize a function that characterizes the task. This characteristic is commonly expressed as the *price*, in some domain specific way, but could also be soonest time to completion, fair distribution of tasks, etc. The manager solicits *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. The contractor's proposal includes the preconditions that the contractor is setting out for the task, 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 3. illustrates the causally relation performed in one possible execution of the protocol. It is constructed by using vectors clocks (timestamps) of Fidge and Mattern associated to the performatives occurrence. The table on bellow presents timestamps associated to the events of FIPA-Contract-Net protocol execution illustrated in figure 3.a (cf. Appendix 1). Event e_b^2 (performative *Propose* of agent B) precedes causally e_a^5 (performative *Accept-Proposal* of agent A) because their timestamps respectively are comparable (i.e. (1,2,0,0) $\leq$ (5,2,2,2)). If there is no comparison between two vector timestamps then the events associated are concurrent (i.e. independent). We find this with performatives *Propose*, *Refuse* and *Not-understood*.

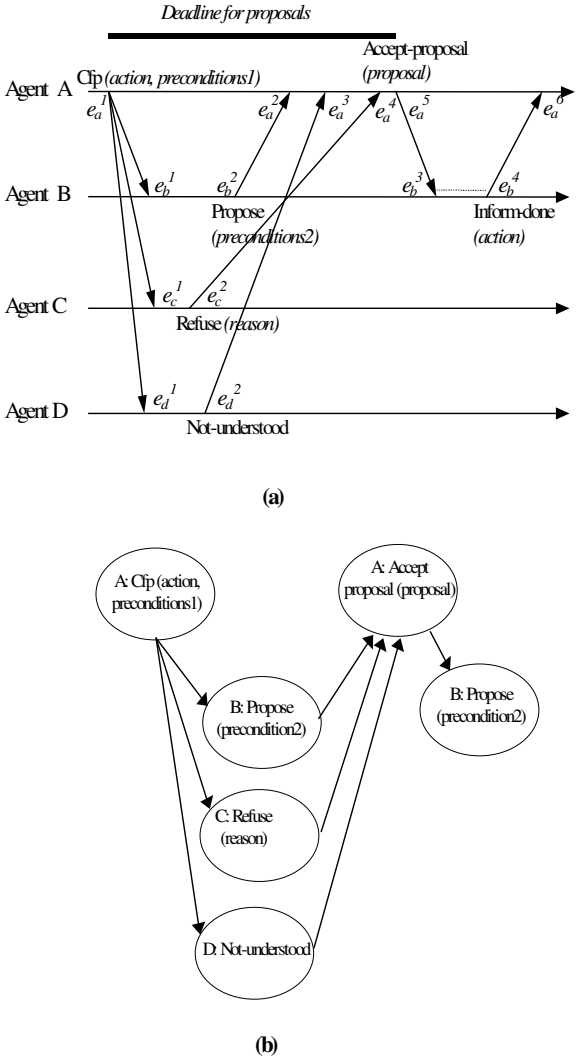

Figure 3: (a) A possible execution of FIPA-Contract-Net protocol ; (b) Causally graph inherent to performatives of FIPA-Contract-Net protocol

$Agent_A$	$Agent_B$	$Agent_C$	$Agent_D$
$e_a^1(1,0,0,0)$	$e_b^1(1,1,0,0)$	$e_c^1(1,0,1,0)$	$e_d^1(1,0,0,1)$
$e_a^2(2,2,0,0)$	$e_b^2(1,2,0,0)$	$e_c^2(1,0,2,0)$	$e_d^2(1,0,0,2)$
$e_a^3(3,2,0,2)$	$e_b^3(5,3,2,2)$	-	-
$e_a^4(4,2,2,2)$	$e_b^4(5,4,2,2)$	-	-
$e_a^5(5,2,2,2)$	-	-	-
$e_a^6(6,4,2,2)$	-	-	-

4.2 Validation of interaction protocols

A computational specification of complex interactions is the description of a combination of exchanging performatives. As we deal with the validation of this combination, we focus on the formalization of the agents' interactions. So we identify some interaction patterns, i.e. protocols of interactions commonly used in multi-agents systems. Hence, we use a formalism that enables validation and verification activities. We choose the Colored Petri nets formalism (CPN) since it is well suited to modeling agent's interactions. It is also richer than others generally used to formalize multi-agent's interaction since its provides verification techniques.
We consider interaction protocols [1] as a basic ones from which complex and high-level interactions can be built. For this, we identify interaction protocols and specify them with CPN formalism enabling to prove their quality and offering an external view of their execution.

4.3 Designing interactions with Colored Petri Nets

Colored Petri Nets (CPN) is a graphical oriented formalism for design, specification, simulation and verification of concurrent systems. It is in particular well suited for systems where communication, synchronization and resource sharing are important. Typical examples of application areas are communication protocols, distributed systems, imbedded systems, automated production systems and workflow analysis.
Colored Petri Nets have got their name because they allow the use of tokens that carry data values and can be distinguished from each other- in contrast to the tokens of low-level Petri nets, which by convention are drown as black. The circles are called places. They describe the states of the interaction. The rectangles are called transitions. They describe the actions. The arc expressions describe how the state of the CP-net changes when the transitions are fired. Each place contains a set of markers called tokens carrying a data

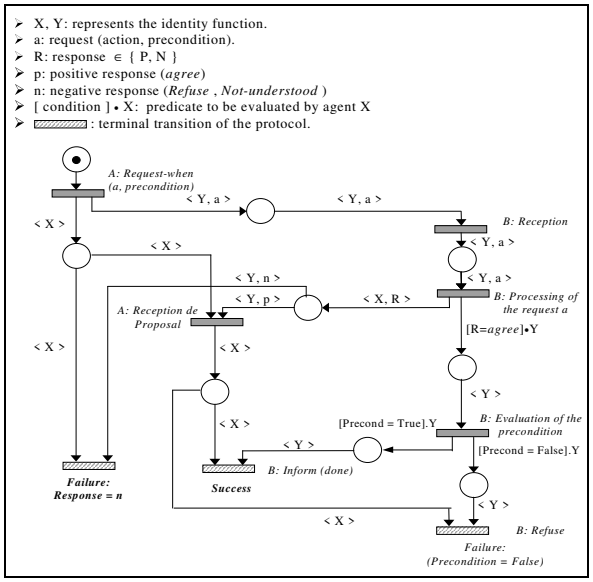

Figure 4: The CPN model of FIPA-Request-when protocol

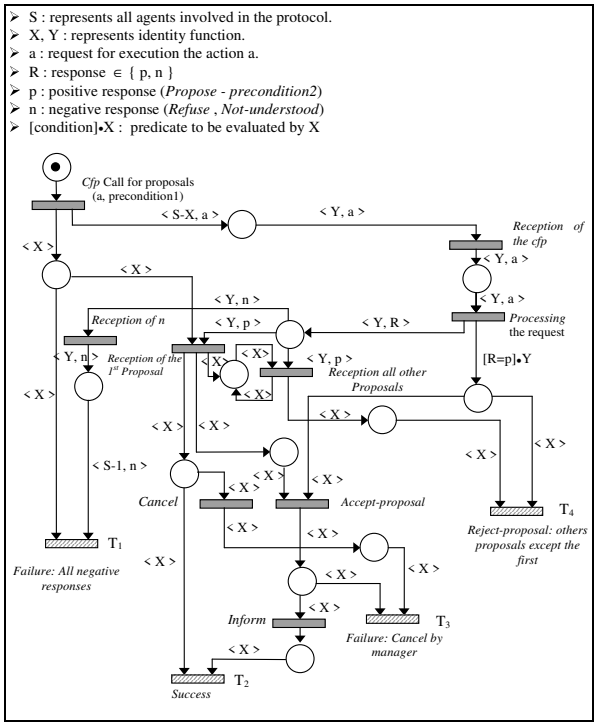

Figure 5: The CPN model of FIPA-Contract-Net protocol

value, which belongs to a given type. To be fired a transition must have sufficient tokens on its input places, and these tokens must take values that match the arc expressions.

The following examples illustrate the CPN modeling of two interaction protocols. The first expresses a simple interaction between two agents while the second involves several agents.

Example 1: FIPA-Request-When Protocol

The CPN of the figure 4. models the FIPA-request-when protocol (cf. example 1. §3.2) and describes the states according to the interaction of the agents A and B. The two agents executes their own part of the protocol designed by their own tokens. We distinguish three possible executions while attending final transitions: a success, a failure when an not-understanding or a refuse message is generated by the receiver or a failure while the necessary preconditions are not satisfied.

Example 2 : FIPA-Contract-Net Protocol

This example illustrates the effective interest of CPN for modeling interactions which involve several agents participation. This protocol (c.f. example 2. §4.1) allows an agent (manager) to choose an other agent (i.e. contractor) which will achieve an action. Mainly, manager sends a broadcast message request for an action (a) execution. The potentially contractor agents send their responses (R) which can be positive (p) or negative (n). We assume that the manager accepts the first positive response and rejects others whatever their type. Notes that the manager can also abort the engaged negotiation by a *cancel* performative. We distinguish through the CPN many possible executions:

- All the contractors refuse to give a proposal ⇒ *protocol failure* (transition T_1)
- At least one positive proposal (p) is generated:
  - If the manager cancels (cancel) the protocol during ⇒ *protocol failure* (transition T_3)
  - If the proposal is the first positive received by the manager then it accepts the proposal (*accept-proposal*) ⇒ *protocol success* (transition T_2)
  - For each other proposals, the manager sends reject messages (*reject-proposal*) according to the transition T_4.

The CPN above synthesizes the set of the observed events of the protocol and the causally relation between them through places and transitions.

5 Pattern Matching

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 curred during computation.

Causally graph is built in two phases: the f captures event's causality by considering perfc events and thus constructing the causally g cluding performatives events, while the secon out interaction protocols by recognizing the an algorithm which is not developed here. can moreover exploit this to construct an tion graph which represents the *interactions' graph*. The recognition of interactions is prov pattern matching (or filtering) with CPN prot brary (c.f. figure 1.).

Example : Figure 6 shows how to built and tc two interaction patterns by using the causall The *Request-When* and *Query-if* interactions a case of independent causally interactions w can not define order between them.

6 Conclusion and future work

In this paper, we have proposed an approac on distributed observation and a formal metho signing interaction protocols in multi-agent s It provides a very powerful, useful, and int property observation way to analyze and exp teractions progress. The concept of causa tween events is fundamental to design and ana tributed computation.

Represented through partial order, the causal be implemented using the timestamps tools. W plied to the communication events, causality to emphasize the correlation between both ele and complex interactions in order to recogr evaluate the interacting situations. This pa proposes an original approach for modeling, st analyze interactions by means of colored Petr This formalism is suitable for concurrent activi it offers several formal tools to study the dyr complex systems. In addition, our approach is and may be applied to a large scale of commu protocols and languages.

In fine, the analyze of multi-agent system inte and in particular the observation of the proces current execution allows not only the detectio

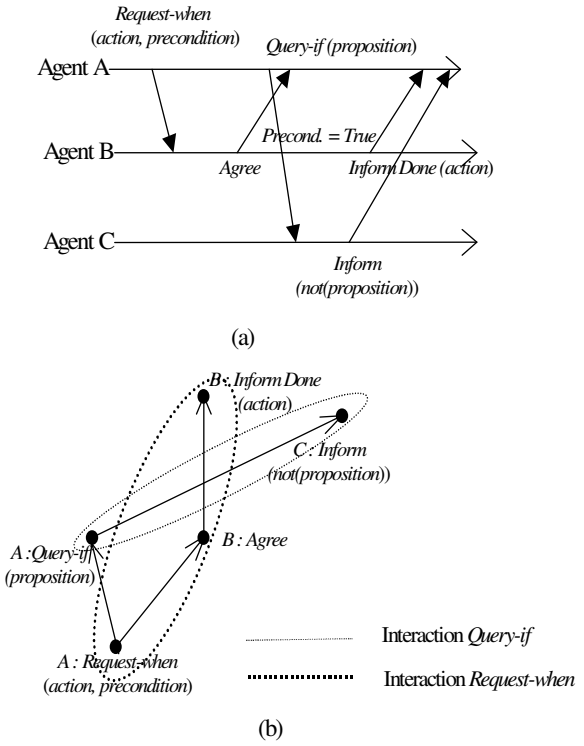

Figure 6: (a) Phase 1:Time-space diagram of two concurrent interactions FIPA-Request-when et FIPA-query-if; (b) Phase 2: Associated causally precedes graph.

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, intend to use the result of the distributed observation and consequently the evaluation that it provides regarding to a set of interactions. The main idea is the following: the learner agent recovers these results, generates a qualitative criteria to be communicated to agents in order to enrich their social knowledge and to improve their future interactions.

Acknowledgements

This work is supported by the "Action Intégrée" of Franco-Moroccan under the contract 97/074/SI/R1. The project is "NetWork design and applications: Video-Conference application".

References

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

[2] 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.

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

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

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

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

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

[8] 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. *3èmes Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents, Chambéry, France*, 1995.

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

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

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

[12] 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.

[13] P. Populaire, Y. Demazeau, O. Boissier, and J. Sichman. Description et implémentation de protocoles de communication en univers multi-agents. *1ères Journées Francophones Intelligence Artificielle Distribuée et systèmes multi-agents, Toulouse*, April 1993.

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

[15] J.R. Searle. Speech acts. *Cambridge University Press*, 1969.

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

Appendix 1 : Vector clocks

The partial order between observed events is reconstructed through timing information associated with each event captured the observer. However, in the system of vector clocks, the time domain is represented 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 and satisfie the following property : for any two events x and y
$x \rightarrow y \Leftrightarrow timestamp(x) < timestamp(y)$
The following three relation are defined to compar two vector timestamps, v_x and v_y:

$$v_x \leq v_y \Leftrightarrow \forall i : v_x[i] \leq v_y[i]$$
$$v_x < v_y \Leftrightarrow v_x \pounds v_y \; and \; \exists i : v_x[i] < v_y[i]$$
$$v_x \parallel v_y \Leftrightarrow not\,(v_x < v_y) \; and \; not\,(v_y < v_x)$$

An example is given in Figure 7.

Appendix 2: FIPA Interactions protocols [5]

This section details graphically a number the FIPA protocols specification used in the paper.

Protocol Description Notation

The following notation is used to describe the stan dard interaction protocols in a convenient manner:

- Boxes with double edges represent communicativ actions.
- White boxes represent actions performed by ini tiator.
- Shaded boxes are performed by the other partic ipant(s) in the protocol.
- *Italic text* with no box represents a comment.

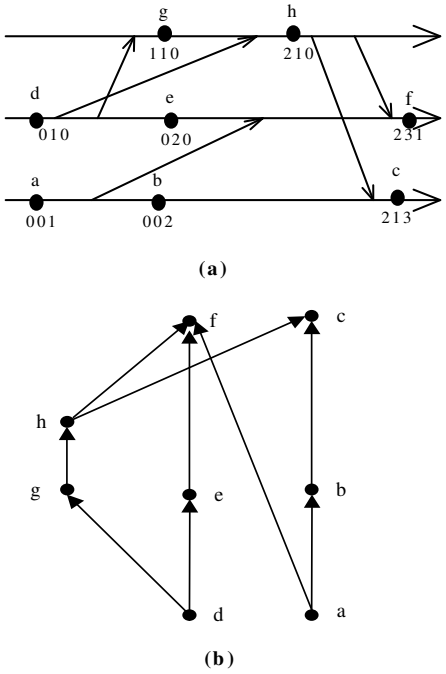

Figure 7: (a) A sample execution of the vector clock

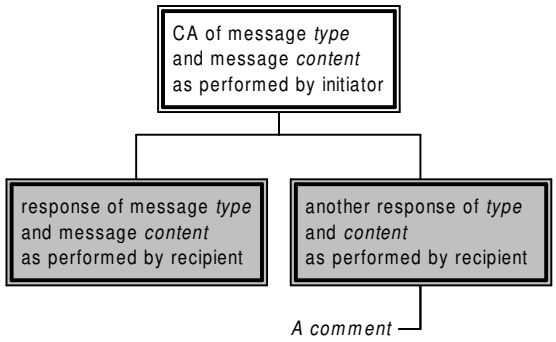

Figure 8: Example of graphical description of proto-

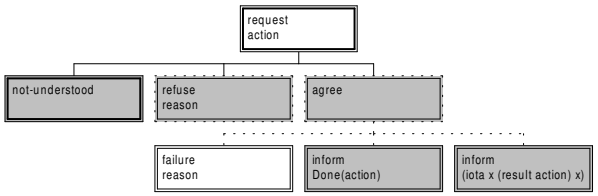

Figure 9: - FIPA-Request Protocol

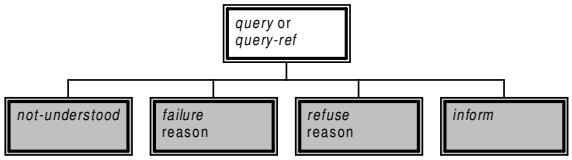

Figure 10: - FIPA-query Protocol

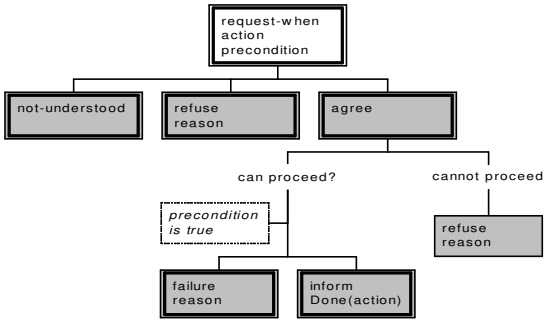

Figure 11: - FIPA-request-when Protocol