Open Protocol Design for Complex Interactions in Multi-agent Systems

Hamza Mazouzi
LAMSADE
University of Paris Dauphine
Paris, France
mazouzi@lamsade.dauphine.fr

Amal El Fallah Seghrouchni
LIPN - UMR 7030
University of Paris 13
Villetaneuse, France
elfallah@lipn.univ-paris13.fr

Serge Haddad
LAMSADE
University of Paris Dauphine
Paris, France
haddad@lamsade.dauphine.fr

ABSTRACT

This paper proposes a generic approach for protocol engineering through the analysis, the specification, and the verification of such protocols when several agents are involved. This approach is three folds: 1) Starting from semi-formal specification by means of Protocol Diagrams (AUML), both formal specification of interaction protocols and their verification are allowed thanks to Colored Petri Nets (CPN); 2) Debugging and qualitative analysis of interactions are based on distributed observation associated with the true concurrency semantics (i.e. CPN unfolding) and ; 3) CPN formalism is extended to Recursive CPN (RCPN) with abstraction in order to deal with open protocols. The main interest of abstraction is the design of flexible protocols giving agents more autonomy during interaction. In addition, abstraction allows concise modeling and easier verification.

measures, performance measures]

Categories and Subject Descriptors

D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal methods, Validation*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Multiagent systems*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Pattern analysis*

General Terms

Design, Verification

Keywords

AUML, Colored Petri Nets, distributed observation, formal specification, protocol diagrams, true concurrency, validation

AAMAS'02 July 15-19, 2002, Bologna, Italy.

1. INTRODUCTION

In order to support organizational interaction, communication, and cooperation in Multi-agent System (MAS), many frameworks have been proposed to standardize interaction between agents. The most significant to date are KQML [16] and FIPA [8]. These frameworks develop a generic interaction language by specifying messages and protocols for inter-agent communication and cooperation. Nevertheless, few works tackle the issue of protocol engineering which typically comprises various stages including specification, verification, performance analysis, implementation, and testing. Recently, some approaches that covers some of these stages have been proposed. In [4] a formal study of the FIPA protocols is proposed. It focuses on the dynamic of interactions showing that some of them may lead to incorrect behavior (e.g. deadlock). In [2] a nice approach provides the specification of interaction protocols based on service concept. In [15], the authors propose an extension of AUML (Agent Unified Modeling Language) [19] to specify interaction protocols reducing the gap between informal specification of interaction and semi-formal one. Since the focus of these approaches was to provide protocols as a global structure and to manage sophisticated interaction, the protocol engineering issue remains a challenge for MAS research. Hence, despite the mentioned efforts, some shortcomings persist: 1) How to translate from informal or semi-formal specification into a formal one when designing interaction protocols? 2) Formal specification is required in order to allow validation of protocols: This rises three fundamental issues: what the relevant properties of protocols are, how they can be validated and when: upstream, during or downstream the interaction process? 3) How to evaluate the success of protocols in practice and to make agents able to explain their relationships within conversations or group utterances in order to develop a proper view of other agents? 4) Which compromise is possible between the autonomy of agents and their behavior following a given protocol?

This paper proposes a generic and global approach to answer the previous questions. Genericity here means that the approach is ACL independent even if our examples are based on FIPA-ACL introduced as shared background of MAS community. Global approach means that it covers all the relevant stages to be considered for the protocol engineering. Two main phases (see figure 1) should be distinguished in our approach: 1) Analysis and design phase detailed in

the second and the third sections: section two shows how to translate interaction protocols from semi-formal specification based on the AUML Protocol Diagrams [20] into their formal specification by means of Colored Petri Nets (CPN) [10]. Section three discusses the main properties to be satisfied to build correct interaction protocols and shows how these properties can be expressed and validated using the CPN formalism. 2) Execution phase corresponds to the

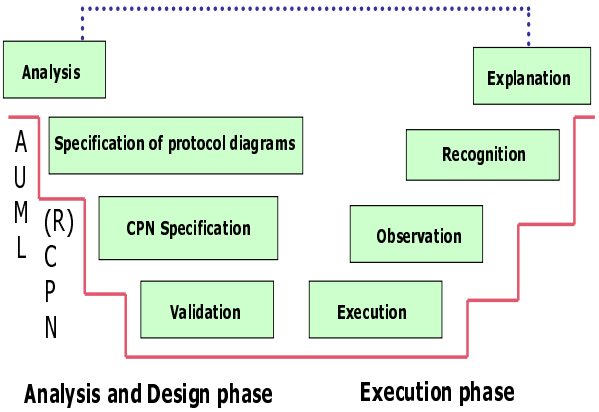

Figure 1: The main phases of our approach

study of the dynamic of interaction: section four introduces the pragmatics of interaction protocol design. It argues for a qualitative evaluation and debugging after interactions' occurrence. This evaluation is possible thanks to the distributed observation paradigm combined with the true concurrency semantics inherent to the CPN unfolding.
Section five introduces the abstraction and its dual operation i.e. refinement to meet the agents' autonomy. We show how the abstraction concept reinforces this aspect and increases the power of CPN formalism for modeling open protocols.
To conclude our presentation, section six outlines our perspectives towards the learning of interaction protocols.

2. INTERACTION PROTOCOL DESIGN

In the following, we provide a formal semantics for sequence diagrams of AUML, an extension of UML for agents. It refers to these sequence diagrams as Protocol Diagrams (PD) [19]. In fact, in [19] the authors do not define formal semantics for the communicative acts for AUML, but instead use the UML meta-model.

2.1 From analysis to formal specification

One of the most interesting semi-formal representation of protocols has been proposed by [20] namely protocol diagrams. Nevertheless, the protocols validation requires their formal specification. This section proposes meaningful rules to translate PD into CPN allowing thus their verification and validation.
Last years, few relevant works on the topic of conversion of UML models into Petri nets has been done [13, 22]. It seems that the UML integration into Petri nets and vice-versa becomes increasingly useful to the users of these two tools. Nevertheless, the study of the relationships between UML and Petri nets (i.e. the translation rules) is still necessary. The main expectation is to overcome the informal aspects of UML.

2.2 Formal Specification by means of CPN

2.2.1 *Why CPN formalism?*

Interaction modeling is often based on misfit formalisms such as the graphs 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 [21]) have also been used to describe conversations between agents. These models are suitable to specify the structure of the conversations when they appear as isolated communications. Nevertheless, they exhibit a poor capacity for computing complex protocols, basically because: 1) any graph state includes all the local states of the agent which leads to the combinatory explosion in the case of real and complex protocols; 2) most usual formalisms consider only sequential processes. Moreover, when these formalisms take into account temporal aspects (in the case of CPN, 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 facing the concurrency of interactions, which is one of the main features of MAS.
Our point of view is that it should be more judicious to resort to well-known and well-proved formalisms for concurrent systems, such as CPN [10] for at least three reasons: 1) they naturally take in charge concurrency; 2) they make the factorization process of treatments easy; and 3) they offer several methods for the analysis and the validation of the modeled protocols. Last years, other works followed [14, 1] and reinforced the choice of CPN as a formalism for the engineering of protocols. In addition, in section 5.3, we show how CPN formalism can be extended with abstraction and dynamic refinement to deal with open protocols.

2.2.2 *Example of protocol modeled by means of CPN*

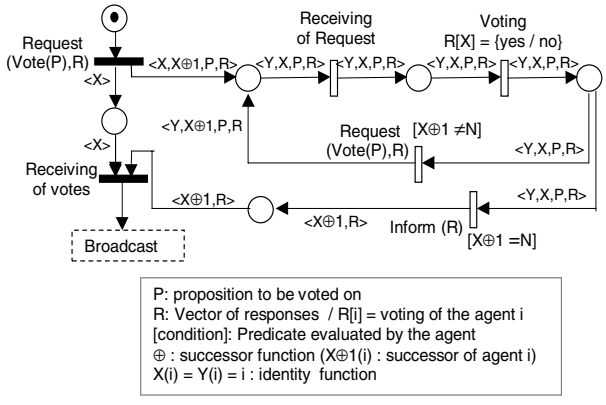

Figure 2: A circular voting protocol

Most of the protocols presented in MAS literature propose bilateral interaction that have been proved insufficient since the relevant cases involve groups of agents (e.g. protocols of election or voting).
However, such protocols are frequently useful in a large class of applications such as the coalition formation, the e-trading,

etc. In the following, we propose a new protocol called "Call For Voting" (CFV) and give its CPN model.
Let's assume that an agent A wants to join a group of agents $G = (A_1, .., A_N)$ located in a private place of e-market. The admission of such an agent requires an agreement of the group based on a vote within the group. The agent A addresses therefore a request to the group through its representative, who will initiate a voting procedure. Thereafter, he informs A that he has been accepted into the group (or not).
The voting procedure can take several forms. So, it is parameterized by the vote decision rules (e.g. majority, unanimity, etc.) as local to each group of agents. Also, the vote implementation should be circular, centralized, etc.. To better efficiency, we choose the majority circular vote against the centralized one, since it reduces the number of exchanged messages (N instead of *2*(N-1)* where N is the number of agents).
Figure 2 presents a CPN model of the circular voting protocol, where the representative of the group initiates a circular voting on the group of N agents by sending a *Request(action = Vote(P),R)* to his successor $(X \oplus 1)$. P describes the proposal submitted to the vote using a specification language (e.g. first order logic) and R is a vector which serves to record the votes ($R[i]$) of each agent A_i. At the reception of $R[i]$, A_i votes and transmits the request to its immediate successor. The protocol ends when A_{N-1} votes and send back an *inform* act to the representative agent which synthesizes the result of the vote and informs the other agents thanks to a broadcast protocol.

2.3 Guidelines for AUML protocol diagrams translation into CPN

The objective of this section is to propose some general guidelines which may be applied to formally specify AUML protocols endowing them with a formal semantics and also to the specification of other protocols for agent communication. Such a semantics will enable the designer to validate his/her specifications. More precisely, we focuse on the description of dynamic aspects of protocols using the CPN's elements (places, transitions, arcs, functions, variables and domains). The AUML objects become domains of colors and variables in the CPN models. For more details about CPN, the reader is invited to refer to [10]. Technically speaking, we introduce an equivalence between the modeling elements proposed in AUML diagrams and CPN. AUML diagrams represent scenarios of the dynamic behavior of protocols through the roles of agents and the communications but lack formal semantics required to validate protocols. The translation from AUML to CPN cannot be automated since the first is informal and the second is formal. What we can provide are guidelines to help the designer. In the following, we propose the operational semantics which associates a CPN with a PD.

2.3.1 The CPN structure

The structure of the CPN can be obtained from the elements of the PDs as follows:
- The "life line" of agent's role is represented implicitly by a sequence of places and transitions belonging to this role (browsed by tokens which symbolize identities of agents of every role). The net is constituted therefore by one sub-net (Petri net process) for each role acting during the interaction and these nets are connected by places that correspond to the exchanged messages.
- A message exchange between two roles is represented by a synchronization place and arcs. The first ongoing arc connects the transition of "message sending" to the "synchronization place" while the second outgoing arc connects this place to the "receiving message transition".
- We add to the CPN model a function of transition labeling in order to interpret the messages exchanged through "sending" and "receiving" transitions.
- We define transitions for the processing of the messages to specify the activities triggered following the reception of messages. The CPN could contain "internal transitions" the firing of which involves decision making methods. In AUML, it corresponds to a *bar* of activation. A causal order is defined between the transitions of reception, processing, and sending of messages and are connected sequentially through the intermediate state places and arcs.
- Every sub-net associated with a role includes a starting place, initially marked, and one or several ending transitions representing the agent's state, respectively at the beginning and at the end of the execution of the interaction protocol.
It often happens that one needs to model flows and control points like the mutual exclusion. For that, we will be brought to use some additional elements.

2.3.2 Domains, valuations and initial marking

Once the structure of the Petri net has been obtained, it remains necessary to manage the variables using data types and domains of colors, variables and guards. A domain of color, representing information to determine the state of the protocol, is associated with places in the Petri net model. We thus consider the variables identified in the general case as follows:
- Informations depicted by variables are mainly associated with places. Domains of the transitions are generally defined according to the domains of the results of functions evaluation of input arcs.
- Places derived from methods (processing transitions) are enriched by local variables such as result of a demand, return of a decision function, etc.
- Informations related to communication contains three parts: the source, the destination and the value of the message.

2.3.3 Translation rules of AUML elements into CPN

This section presents the main situations to be taken in charge by transformation rules.

R1: Choice or decision making
Figure 3 illustrates a possible transformation of the symbol of sending a message among a list (exclusive or), so that precisely one communicative act is sent. With each type of message is associated a transition and a function on its input arc. The function plays the role of a filter, i.e. it control the firing of the transition corresponding to the message type. Figure 4 shows how the symbol of decision is translated into CPN. This symbol means what communicative act (zero or more) will be sent.

R2: Concurrency or parallelism
Figure 5 models concurrency (parallel case or multi-threading) of sending performatives by means of CPN.

R3: Synchronous and asynchronous communication

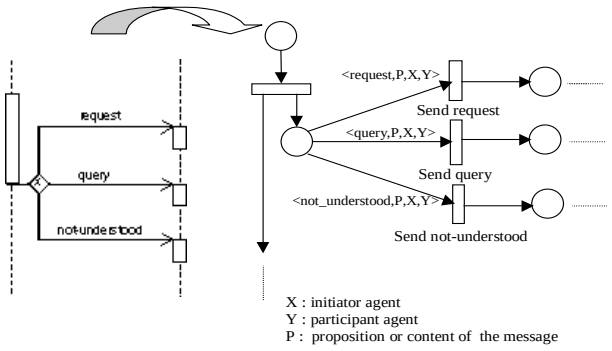

Figure 3: CPN model of "exclusive or"

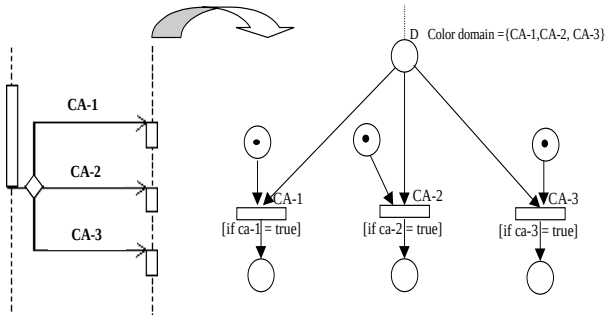

Figure 4: CPN model of "inclusive or"

Asynchronous and respectively synchronous messages are translated into one of CPNs given in figure 6, respectively sides (a) and (b).

R4: Basic interaction
Figure 7 illustrates the transformation of the sending of a request and the receiving of an answer into CPN.

R5: Cardinality of messages
The cardinality of a message in AUML translates the number of senders (n) and recipients (m) of a message. The associated notation is to mention cardinalities at the beginning and at the end (extremity) of the arrow. The transformation into CPN of this representation specifies the number of initiating and participating agents through the domains of the places. Figure 8 illustrates an example of a message with cardinality.

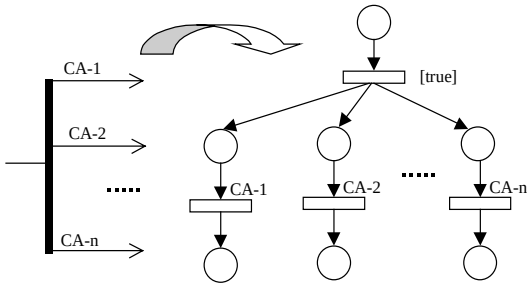

Figure 5: CPN model of parallelism

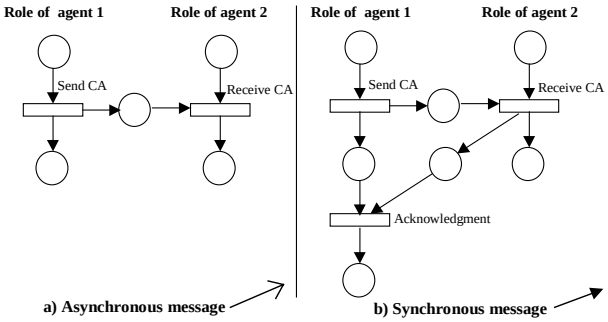

Figure 6: CPN of (a)synchronous messages

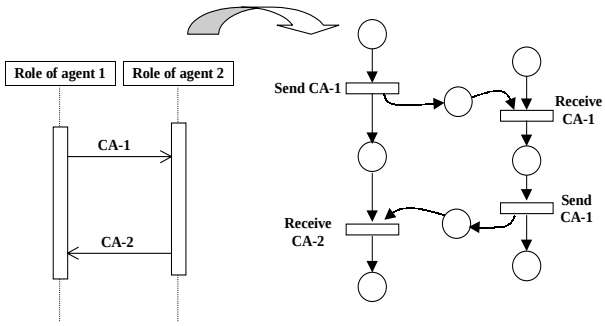

Figure 7: CPN of sending/reception message

R6: Repetition or loop
A loop in a part of AUML specification is represented by an arrow and an expression of guard or an end condition. In CPN, the loop is specified in the same way except that the end condition is a guard expression associated with the transition that starts the loop. A loop has a beginning and "ending transitions" that are connected by a place and two arcs (from the end transition to the place and from the place to the "beginning transition").

2.3.4 Example of Translation

Figure 9 presents the transformation of the FIPA-request-when [8] protocol into CPN. The CPN such as it is specified doesn't support the overall semantics of a FIPA-request-when interaction. Indeed, PDs of AUML don't represent the semantics of orders on the receptions of the messages, nor the alternatives or possible competitions between activities

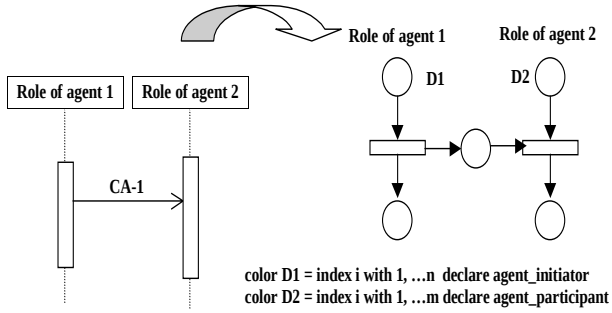

Figure 8: Message cardinality

associated with the processing of messages. Moreover, we note through the CPN that the behavior of the participating agent is correct. Nevertheless, concerning the initiator, there is no distinction between failure or success of interaction situations. The activities related to the reception of messages in the PD are connected without distinction between acts that succeed or not. The designer interprets this by a logical continuity of the interaction whereas on the reception of the negative answer, the agent must suspend the interaction in this protocol.

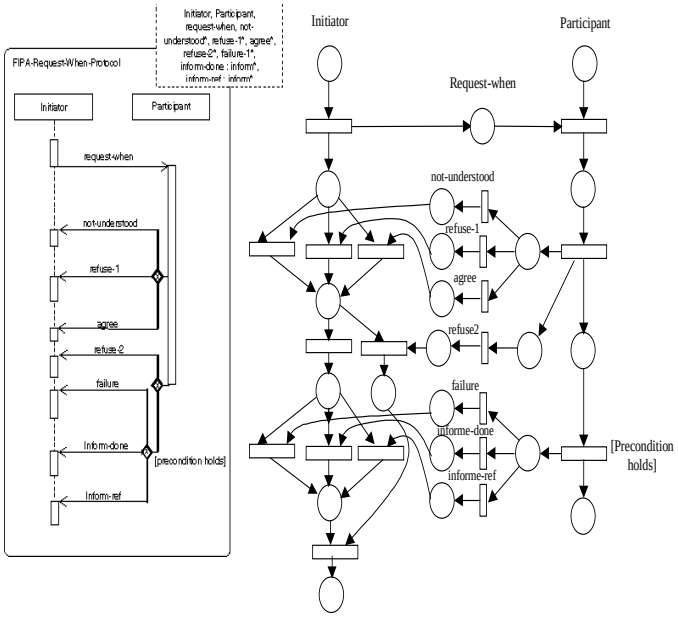

Figure 9: A translation of the protocol FIPA-request-when diagram into CPN

3. INTERACTION PROTOCOL VALIDATION

After replacing each modeling element of PD by the corresponding segment CPN (developed in section *2.3.3*), the simulation tools of Petri net analysis should be applied to validate the dynamic behavior of the protocol. Creating a CPN protocol provides a model that explicitly identifies roles and their state behavior. Furthermore, the model can serve as an executable specification through the net simulation. The reachability graph is then developed to verify and identify properties of the protocol. In some cases, it is interesting to establish some properties (e.g. eventual reception of a message modeled through a transition firing), while in other cases it is relevant to invalidate others (e.g. conflicting use of resource).

To analyze CPN, we can first unfold the CPN into an ordinary Petri net representing the equivalent behavior (under the hypothesis of finite domains of colors and then apply standard net analysis techniques and algorithms. In this section, we discuss some important verifiable properties of CPN protocols and their interpretation in MAS.

When a protocol is modeled by means of CPN, two kinds of properties should be verified: 1) Structural properties are related to the CPN topology. They are marking-independent and help to build correct protocols during the specification phase. The main interest is to free the designer from setting the number of agents, of resources, etc., at earlier stages ; 2) Behavioral properties concern the qualitative behavior once the initial marking of the protocol is fixed. In the following some relevant properties are discussed in terms of CPN and their incidence on the interaction protocols:

- *Structural liveness* of a CPN protocol guarantees the existence of an initial state such that for any accessible state, at least one operation is executed.
- *Cyclic occurrences* guarantee the existence of a control so that all the operations can be executed an infinity of times. This is not often desirable in protocols.
- *Consistency* guarantees the existence of a control such as all the operations are executable. This enables an agent to come back periodically to his initial state (or some home states if any).
- *Boundedness* guarantees that the occurrences of the modeled objects remains limited for any initial state (i.e. the messages are continuously treated and do not accumulate in the net). In fact, the CPN places often convey agents' identities and messages. Hence, this property enable a ceaseless growth of sending of messages without their processing.
- *Accessibility* guarantees the controllability of the interaction. It makes it possible to lead a conversation to a desired state starting from the initial state.
- *Absence of deadlock* guarantees that given an initial state, at least one operation can be carried out whatever the state reached during a conversation.
- *Liveness* is even stronger than the absence of deadlock. From the protocol point of view, it guarantees that for any initial state, all the operations (independently) can be always be executed whatever the agent's decision. Moreover, in our model we added a semantics for the possible results of the conversations by means of final transitions.

4. STUDY OF INTERACTION DYNAMICS

In [5] we have proposed an efficient mechanism to study the dynamic of interaction, which corresponds to the execution phase (see figure 1).

4.1 Main steps of the execution phase

This phase relies upon four steps: 1) First, the execution of the MAS incorporates an *on-line* distributed observation mechanism which captures the traces of the relevant events underlying the agents' interactions; namely sending/receiving messages related to the interaction protocols. 2) The second step exploits the obtained traces, builds the global causal graph (GCG) of all events [4] underlying MAS execution. This is ensured *off-line* and based on logical clocks proposed by J. Fidge [7]. 3) The third step is the recognition of interactions. It is based on a pattern matching algorithm as detailed in [5] and briefly presented in section 4.2 . The algorithm is jointly based on the GCG and the CPN models used as filters (CPN_patterns) (see Figure 10). 4) The last step exploits the outputs of our algorithm in order to explain the behavior of interacting agents.

4.2 Pattern matching algorithm based on unfolding Petri nets

Our aim is to represent two aspects in our model: The first expresses serial and concurrent events to be observed, i.e., the interaction states achieved by agents; the second aspect describes the causally precedence relation that exists among communicative acts occurred during computation.

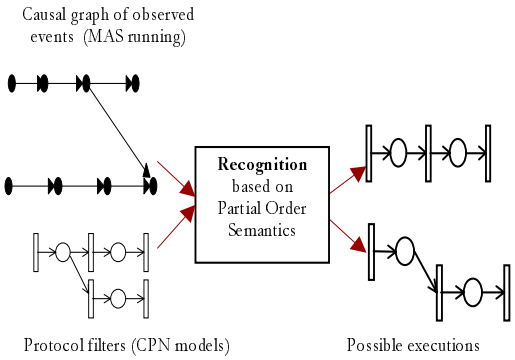

Figure 10: Pattern matching algorithm

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 unfolding of Petri nets [18]. 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.

4.3 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 the possible executions in figure 10). 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 with the firings (with respect 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 [17, 6] 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 number of state is not faced since the unfolding we look for corresponds to a specific protocol computation and necessarily is finite.

4.4 Recognition process

Our algorithm starts from the GCG and tries to recognize the protocol(s) that have been executed. Each protocol model is a CPN with which several computations may be associated. Hence, our goal is 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 [18] which enable to associate a set of unfolding nets with a given protocol modeled as CPN. For technical details of our algorithm, the reader is invited to see [5]. In the following, we present an example of unfolding and recognition principles.

4.4.1 Recognition process example

Let us consider two protocols (cf. figure 11) which provide the same service (sending query messages to agents and receiving 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 sequential. One can easily verify in $Protocol_2$ that, except for the first firing of T_1 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 .
Techniques of partial orders offer a suitable framework for the analysis of interaction dynamics. One can verify during the development of different executions of the protocols that all executions produced by the $Protocole_2$ can be generated by the $Protocole_1$. Knowing that the two protocols can produce a same execution, filtering is assured thanks to the causal graph whose semantics will permit to keep only protocols which can produce such a causal graph.
Let us now observe a computation of one of the two protocols given through a causal graph CG in (figure 12.b). The algorithm presented below develops all the possible process nets (see figure 12.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 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 CG (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 the fired 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.

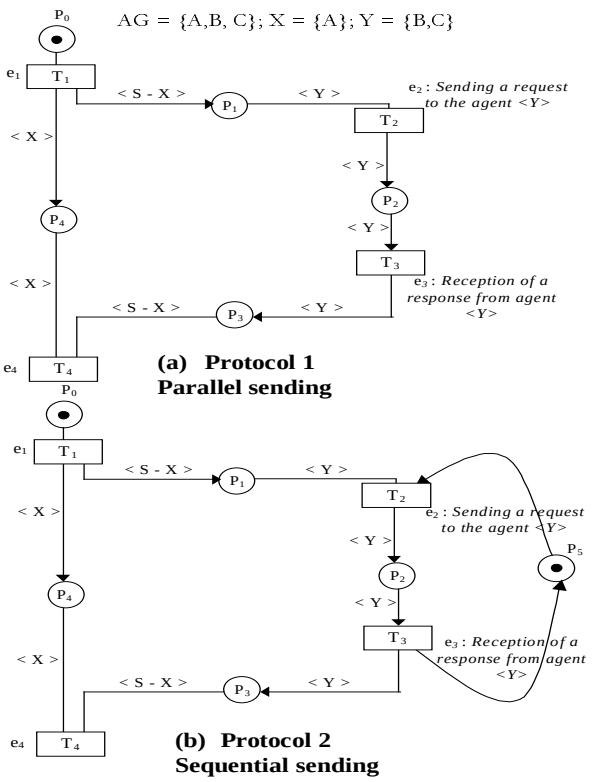

Figure 11: Two CPN protocols to be recognized

4.5 Explanation and analysis

Various points of analysis can be deduced from the algorithm. It allows to:

- recognize the protocol which has produced a given interaction,
- distinguish in term of events different interactions within an execution,
- know the different states visited by agents during a conversation,
- know the finality of a conversation (success, failure) and to analyze the historic of such situation through its causal graph,
- validate *a posteriori* the protocols using experiments and debugging tools, etc.

5. DESIGN OF OPEN PROTOCOLS

Interaction protocols can range from negotiation schemes to simple requests for a task. While referring to FIPA-ACL, the distinction between primitive and composite communicative acts make protocols specification increasingly complex and consequently provides robustness and more expressiveness for large scale of application.

5.1 Open protocols based on abstraction

The main contribution of CPN as illustrated above is modeling protocols in which elementary actions associated with irreducible tasks (primitive acts) and complex tasks

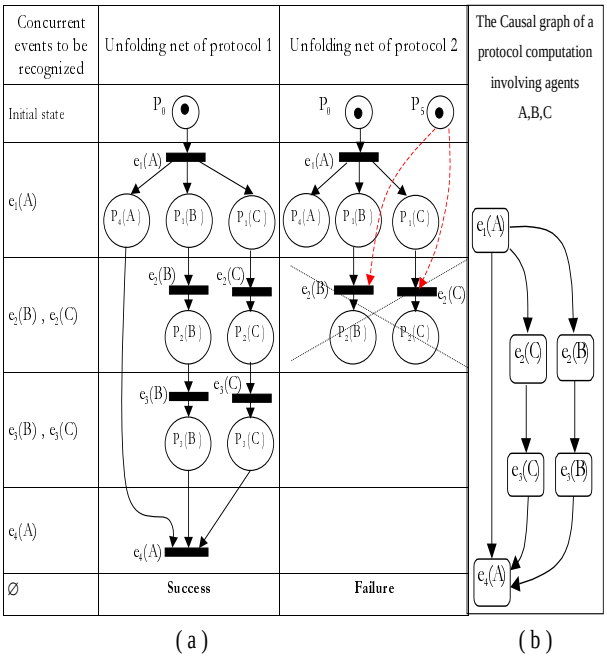

Figure 12: The unfolding process of the two protocols according to the CG

(composite acts) are not distinguished. The Recursive Colored Petri Net (RCPN) formalism, introduced in this paper, can be viewed as an extension of recursive Petri nets, defined by [9], which have been successfully used for specifying plans of agents in MAS [3]. We complement this work to offer a power of expressivity in modeling protocols. It permits to consider an action of a protocol as another sub-protocol or a composite act which leads to provide an abstraction and dynamics in the structure of the conversation. In addition, the hierarchical aspect of RCPN supports the dynamic refinement of transitions and allows a protocol to be considered at different levels of abstraction.
The recursive Petri net formalism we have introduced overcomes some limitations of usual categories of Petri nets [12] (e.g. ordinary Petri Nets, High-Level Petri Nets (HLPN) and even Hierarchical HLPN(HHLPN)) that are apparent if one considers a Petri net as an interaction protocol:

- transitions firings are instantaneous whilst an action lasts some time,
- (HLPN only) transitions are elementary actions as one needs to see an action as an abstraction of a protocol,
- (HHLPN) transition is a syntactical abstraction (i.e. for *a priori* medeling only) and there is no clear end to its firing, while dynamicity is required in the structure of the net (i.e. refinement during execution) as provided by our RCPN.

The abstraction concept is well known as an elegant way that helps to provide: 1) concise modeling: when the model size is too big to be represented in exhaustive way. This could be the case when modeling complex interactions between cognitive agents; 2) easier verification: when the model used is limited in terms of verification tools. In this case, the designer should translate the initial model to an equivalent

one which preserves the expected properties and provides suitable tools/techniques for their verification.
A fortiori, as interactions in MAS could be complex, the protocols to be designed should benefit from abstraction both from modeling and verification perspectives. Moreover, to build open protocols, abstraction is also required. The main goal is to find an acceptable compromise between the concept of protocol - as a well structured exchange between agents - while giving interacting agents some degree of autonomy and freedom in the protocol execution.

5.2 Our Model for open Protocols

Our model considers a protocol as a collection of actions which can be performed sequentially or concurrently in some specific order by a set of agents that work in a distributed and asynchronous way.
Furthermore, these actions concern communicative acts and local processing of the underlying contents. The way in which the agent processes the contents of the message in conformance to its mental state is not taken into account by our formalism.
A protocol involves both elementary actions (sending and receiving performatives, performing atomic actions) and complex actions (i.e., composite communicative acts). Semantically, there are two types of actions:
• an *elementary action*, associated with an irreducible task (processing the content of a primitive communicative act) and which can be performed without any decomposition,
• an *abstract action*, the execution of which requires its substitution (i.e. refinement) by a new sub-protocol which can be simply composed by another communicative act or a sequence or parallel actions (formulated by the operators ";" and "|" in [8]),
Methods: Intuitively, a method may be viewed as the way to perform an action. A method requires: a label, a list of formal parameters to be linked when the method is executed, a set of Pre-conditions (i.e. necessary conditions of the method execution), and a set of Post-conditions (i.e. the conditions satisfied after the method execution). Note that, the Pre- and Post-conditions correspond to what may be satisfied, when the variables and the expressions of the Petri net are bounded. Pre- and Post-Conditions may be compared to the rational effects and feasibility's pre-conditions (as in FIPA specification).
Depending on the action definition, a method may be elementary or abstract according to the action type. An elementary method calls for a sub-routine in order to execute the associated elementary action. An abstract method calls for a sub-protocol corresponding to the refinement of the associated abstract action.

5.2.1 Example

Let us consider the protocol FIPA-Request applied to the following situation: An agent X, desires to join several working groups represented by their responsible (representative of the groups) A, B, C, etc.. X sends his Request (*Action Send request of admission*). The protocol skeleton is given in figure 13. Each responsible receives the request (*Reception Request*) and handles the request (*Action Request Handling*). Obviously, each responsible may have a proper method of handling this request within his group. To take into account this variety, the handling action is represented through an

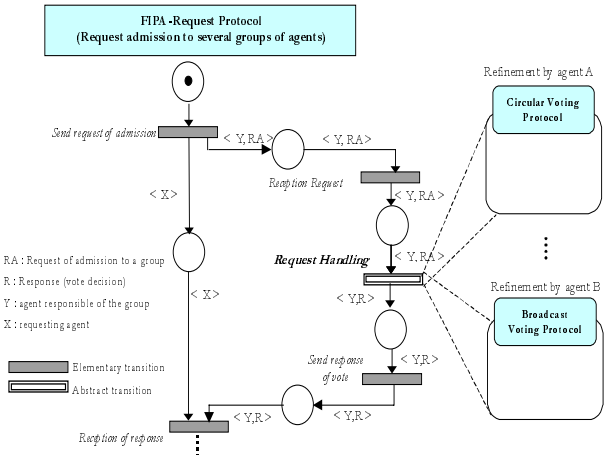

Figure 13: RCPN modeling of vote requesting

abstract transition, the refinement of which could be context dependent, and consequently, the RCPN enables to associate several refinements with an abstract transition. For instance, one possible refinement could be the circular voting protocol given in figure 2. The main protocol imposes only that each responsible provides an answer to the agent X.

5.3 Recursive Colored Petri Nets (RCPN)

This section presents both syntactical aspects and the semantics of RCPN.

5.3.1 Syntax of RCPN

A recursive CPN is defined as follows: $RCPN = (CPN, \Omega, \Upsilon)$ where CPN is a colored Petri net $(\sum, P, T, C, G, F, M_0)$ as defined in [10].
The set of transitions T is typed as follows:
. A transition of T can be either elementary or abstract, the sets of them are respectively denoted by T_{elem} and T_{abs} (with $T = T_{elem} \biguplus T_{abs}$ where $\biguplus$ denotes the disjoint union).
. Ω is a labeling function which associates to each abstract transition an initial marking according to the parameters of the transition.
. Υ is an effective semi-linear set of final markings (can be specified with any usual syntax notation).

5.3.2 Semantics of RCPN

An operational semantics of RCPN is given in terms of states and change of states. A state of a RCPN is a tree where each node is labeled by a marked RCPN and an arc corresponds to the abstract transition and the colored token by which the transition has been fired.
The intuitive interpretation of a state is the following:

- The root of the tree corresponds to an initial protocol,
- Each edge represents a firing of abstract transition according to a color where the extremity of the edge denotes the unfolded net,
- This structure represents the concurrent behavior of a RCPN generating thus a tree "of abstract transition calls" relatively to each token of the domain.

Extended marking

An *extended marking* "tr" of an RCPN noted $N = (\sum, P, T, C, G, F, M_0, \Omega, \Upsilon)$ is a labeled tree $tr = (S, M, E, A)$ where:
- S is the set of nodes where each node s represents a RCPN,
- M is a marking function from $S \rightarrow Bag(C(p))$ (the marking of RCPN),
- $E \subseteq S \times S$ is the set of edges,
- A is a function from $E \longrightarrow \cup(t, c)$, where $t \in T_{abs}$ and $c \in C(t)$. An edge is labeled by an abstract transition and a token element of the domain of the transition (labeling of the edges both by the abstract transition and the token having permitted its firing).

A marked RCPN (N, tr_0) is a RCPN N associated to an initial extended marking tr_0. This initial extended marking is usually a tree reduced to a unique node.
We note by $s_0(tr)$ the root of the extended marking tr. An empty tree is denoted by $\perp$. All colored marking m can be seen like extended marking, denoted by $\lceil m \rceil$, consisting in only one node in the tree.
For a node s of an extended marking, we note by $Pred(s)$ its predecessors in the tree (defined only if s is different from the root) and by $succ(s)$ the set of its direct and indirect successors including s ($\forall s \in S, Succ(s) = s' \in S | (s, s') \in E^*$ where E^* denotes the reflexive and transitive closure of E).
An elementary step of a RCPN may be either the firing of a transition or the closing of a sub-tree (also called a *cut* and noted by τ).
The finite markings defined for a RCPN correspond to cuts of steps (τ) of its extended marking.
As reasoning on the extended marking tree we give thereafter the firing rules of RCPN.

The firing rules

A transition t is fireable for a color c from a node s of an extended marking tr (noted by $tr \xrightarrow{t,c,s}$) iff $\forall p \in P : M(s)(p) \geq F(p,t)\langle c \rangle$ and a cut of a step is fireable from a node s (noted by $tr \xrightarrow{\tau,c,s}$) iff $M(s) \in \Upsilon$.
The firing of an elementary step t from a node s of an extended marking $tr = (S, M, E, A)$ leads to the extended marking $tr' = (S', M', E', A')$ (noted by $tr \xrightarrow{\tau,c,s} tr'$) depending on the type of t.

- t is an *elementary transition* ($t \in T_{elem}$).
  The thread associated to s fires such a transition with a color c as for a CPN. The structure of the tree is unchanged. Only the current marking of s is updated.
  . $S' = S$
  . $\forall s' \in S \setminus \{s\}, M'(s') = M(s')$
  . $\forall p \in P, M'(s)(p) = M(s)(p) - F(p,t)\langle c \rangle + F(t,p)\langle c \rangle$
  . $E' = E$
  . $\forall e \in E, A'(e) = A(e)$

- t is an *abstract transition* ($t \in T_{abs}$).
  The thread associated to s consumes the input tokens of t. It generates a new thread s' with initial marking the starting marking of t. Let us note that the identifier s' is a fresh identifier absent in S.
  . $S' = S \cup \{s'\}$
  . $\forall s'' \in S \setminus \{s\}, M'(s'') = M(s'')$
  . $\forall p \in P, M'(s)(p) = M(s)(p) - F(p,t)\langle c \rangle$
  . $M'(s') = \Omega(t)$
  . $E' = E \cup \{(s, s')\}$
  . $\forall e \in E, A'(e) = A(e)$
  . $A'((s, s')) = (t, c)$

- t is a *cut step* ($t \in \tau$).
  If the thread is associated with the root of the tree and that one of final markings is reached, the reduction leads to the empty tree. In the other case, the subtree rooted at this thread is pruned and the output tokens of the abstract transition which gave birth to the thread are added to the marking of its father.
  . $S' = S \setminus Succ(s)$
  . $\forall s' \in S' \setminus \{pred(s)\}, M'(s') = M(s')$
  . $\forall p \in P, M'(pred(s))(p) = M(pred(s))(p) + F(p,t)\langle c \rangle)$
  . $E' = E \cap (S' \times S')$
  . $\forall e \in E', A'(e) = A(e)$

5.3.3 Some analysis issues

The analysis and the verification of an RCPN could be based on two methods. The first one is indirect i.e., it relies on the translation of the RCPN into CPN model: the unfolding RCPN substitutes each abstract transition by the protocols which refine it and connect them to the upstream places. Then, the resulting CPN is modified by creating non-deterministic functions on the choice of one of the refining sub-protocols (for instance, the refinement may corresponds to a local decision of the agent). Hence, we can apply the usual CPN techniques for analysis and validation.
The second method is said direct, because it defines how to build the reachability graph directly from an RCPN Protocol thanks to the firing rules of RCPN transitions (if it is permitted by the net size). Then we can directly identify many useful properties, since we will have an explicit state-space view of the protocol.

6. CONCLUSION AND FUTURE WORK

This paper proposes a generic approach for the protocol engineering in the case of complex interactions and open protocols. A formal translation of interaction protocols from AUML Protocol Diagrams into CPN is proposed enabling their formal analysis and validation. The study of the dynamic of interactions is also described through the execution phase. Our approach also proposes the RCPN formalism which offers the following main advantages:
- The formalism is domain and language independent and supports complex protocols with different levels of abstraction.
- It permits protocol reuse allowing agents to adapt the interacting process, in the case of similar situations according, to the execution context (library of abstract protocols could be the basic building blocks of the new protocols).
- The qualitative and quantitative analysis of protocols by means of large number of formal analysis methods to prove properties (the CPNs can be submitted to automated analysis by Petri net analysis tools, such as Design/CPN [11].
- The protocol size remains controllable during the specification phase and its complexity is tractable for the validation step.

Our future work intends to exploit the analysis results of the interactions to learn interaction protocols.

7. REFERENCES

[1] R. S. Cost, Y. Chen, T. Finin, Y. Labrou, and Y. Peng. Modeling agent conversations with colored Petri nets. *In proc. of the* 3^{rd} *International Conference on Autonomous Agents (Agents'99), Workshop on Agent Conversation Policies, Seattle, Washington*, May 1999.

[2] M. d'Inverno, D. Kinny, and M. Luck. Interaction protocols in Agentis. *In Proc. of International Conference on Multi Agent Systems (ICMAS'98)*, 1998.

[3] A. El-Fallah-Seghrouchni and S. Haddad. A recursive model for distributed planning. *In proc. of the* 2^{nd} *International Conference on Multi-Agent Systems, Kyoto, Japon*, 1996.

[4] A. El-Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. A formal study of interactions in multi-agent systems. *In Proc. of the* 14^{th} *International Conference on Computers and Their Applications (ISCA CATA-99), ISBN: 1-880843-27-7, Cancun, Mexico*, pages 240–245, April 1999.

[5] A. El-Fallah-Seghrouchni, S. Haddad, and H. Mazouzi. Protocol engineering for multi-agent interaction. *F.J. Garijo, M. Boman (eds.), Proc. of Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'99). LNAI, vol. 1647, Springer Verlag.*, 1999.

[6] J. Esparza, S. Romer, and W. Volger. An improvement of mcmillan's unfolding algorithm. *In Proc. of the* 2^{nd} *International Workshop on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96), Springer Verlag, Passau, Germany*, 1055 of LNCS:87–106, March 1996.

[7] J. Fidge. Timestamps in message passing systems that preserve the partial order ring. *In Proc.* 11^{th} *Australian Computer Science Conference*, pages 55–66, february 1988.

[8] FIPA. Foundation for intelligent physical agents. *FIPA 97 Specification. Part 2, Agent Communication Language, http://www.fipa.org*, 1997.

[9] S. Haddad and D. Poitrenaud. Theoretical aspects of recursive Petri nets. *In Proc.* 20^{th} *International Conference on Applications and Theory of Petri nets, Williamsburg, VA, USA*, 1639 of Lecture Notes in Computer Science:228–147, 1999.

[10] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, volume 1. Springer-Verlag, Basic Concepts of Monographs in Theoretical Computer Science, 1992.

[11] K. Jensen. Design/cpn, version 4.0. *http://www.daimi.au.dk/designCPN/, University of Aarhus, Denmark,*, 1999.

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

[13] P. King and R. Pooley. Using UML to derive stochastic petri net models. *In Proc. of the* 5^{th} *UK Performance Engineering Workshop (UKPEW '99)*, pages 45–56, 1999.

[14] J. Koning, G. Francois, and Y. Demazeau. Formalization and pre-validation for interaction protocols in multi agent systems. *In* 13^{th} *European Conference on Artificial Intelligence, Brighton, 1998*, 1998.

[15] J. Koning, M. P. Huget, J. Wei, and X. Wang. Extended modeling languages for interaction protocol design. *In Proc. of Agent-Oriented Software Engineering (AOSE'2001), Montreal, Canada*, pages 93–100, 2001.

[16] KQML. Specification of the kqml agent-communication language. Technical report, DARPA Knowledge Sharing Initiative External Interfaces Working Group, http://www.cs.umbc.edu/agents/kse/kqml/, 1993.

[17] K. McMillan. On-the-fly verification with stubborn sets. *In Proc. of Computer Aided Verification, Springer Verlag, Montreal*, 663 of LNCS:164–175, June 1992.

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

[19] J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. *In Proc. of Agent-Oriented Information Systems (AOIS) Workshop at AAAI*, 2000.

[20] J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. *In Proc. of the* 1^{st} *International Workshop on Agent Oriented Software Engineering (AOSE), Paolo Ciancarini and Michael Wooldridge eds., Berlin*, pages 121–140, 2001.

[21] V. Parunak. Visualizing agent conversations: Using enhanced Dooley graphs for agent design and analysis. *In Proc. of International Conference on Multi Agent Systems (ICMAS'96), AAAI press.*, pages 275–282, 1996.

[22] R. G. Pettit and H. Gomaa. Validation of dynamic behavior in UML using colored Petri nets. *In Proc. of UML'2000*, 2000.