A Scalable Petri Net Representation of Interaction Protocols for Overhearing*

Gery Gutnik and Gal Kaminka

The MAVERICK Group, Computer Science Department, Bar-Ilan University
52900 Ramat Gan, Israel
{gutnikg ,galk}@cs.biu.ac.il

Abstract. In open distributed multi-agent systems, agents often coordinate using standardized agent communications. Thus, representing agent conversations is an important aspect of multi-agent applications. Lately, Petri nets have been found to provide certain advantages comparing to other representation approaches. Radically different approaches using Petri nets to represent multi-agent interactions have been proposed, and yet relative strengths and weaknesses of these approaches have not been examined. Moreover, no approach was shown to provide a comprehensive coverage of advanced standardized communication aspects such as those found in FIPA interaction protocols. This paper presents (i) an analysis of existing Petri net representation approaches in terms of their scalability and appropriateness for different tasks; (ii) a novel scalable representation approach, particularly suited for monitoring open systems; and (iii) a skeletal procedure for semi-automatically converting FIPA interaction protocols to their Petri net representations. We argue that the representation we propose is comprehensive, in the sense that it can represent all FIPA interaction protocol features.

1 Introduction

Open distributed multi-agent systems often involve multiple, independently-built agents performing mutually dependent tasks. To allow different agents designs to be developed independently, without having to consider the internal design of other agents, the coordination of the activities is often accomplished using standardized inter-agent interactions, typically by communications. Indeed, the multi-agent community has been investing a significant effort in developing standard communication languages to facilitate sophisticated multi-agent systems (e.g., FIPA communication standards [4]). These languages define agent interaction protocols that rely on pre-defined communicative acts for a variety of system tasks, ranging from simple queries, to complex negotiations by auctions and bidding. For instance, FIPA Contract Net Protocol [4] defines possible sequences of concrete messages that allow the interacting agents to negotiate.

Ideally, interaction protocols should be represented in a way that allows performance analysis, validation and verification, automated monitoring, debugging, etc. Various formalisms have been proposed for such purposes. However, Petri nets

* This research was supported in part by BSF grant #2002401

have been shown to offer significant advantages in representing multi-agent interactions, compared to other approaches [2,8,9,10]. Specifically, Petri nets are useful in validation and testing, automated debugging and monitoring [13] and dynamic interpretation of interaction protocols [3].

Unfortunately, existing literature on using Petri nets to represent multi-agent interactions leaves open several questions. First, different approaches to represent multi-agent interactions have been introduced, and yet their relative strengths and weakness have not been investigated. Second, most previous investigations have not provided a systematic comprehensive coverage of all issues that arise in representing complex protocols such as the standardized FIPA interaction protocols.

This paper addresses these open challenges. We analyze and compare existing approaches to representing interactions using Petri nets (Section 3). This comparison is done based on the type of Petri net chosen, its choice of representing individual or joint states, and explicit representation of messages. We then present a novel scalable representation that uses Colored Petri nets in which places explicitly denote joint conversation states and messages (Sections 4). This representation can be used to cover essentially all features used in FIPA conversation standards, including interaction building blocks, communicative act attributes (such as message guards and cardinalities), protocol nesting and temporal aspects (e.g., deadlines and duration). Finally, we provide a skeletal algorithm for converting FIPA conversation protocols in AUML, i.e. *Agent UML*, (the chosen FIPA representation standard [4,11]) to Petri nets (Section 5). Section 6 concludes.

2 Background

We begin first with a brief overview of Petri nets, and then survey existing approaches that use Petri nets in representing multi-agent interactions.

Petri nets are a graphical representation for describing systems in which multiple concurrent states may exist. An early elaboration of Petri nets is called Place/Transition nets (*PT-nets*), while another high-level extension is called Colored Petri nets (*CP-nets*) [6].

A PT-net is a bipartite directed graph where each vertex is either a place (typically denoted by circles) or a transition (rectangles). Arcs are directed edges connecting places to transitions and vice versa. A place can contain tokens (small black dots). An assignment of tokens to places is called *a marking*. Arcs may have associated integer *expressions*, which determine the number of tokens associated with the corresponding arc. A transition is *enabled* if and only if the marking of its input places satisfies the appropriate arc expressions. It then *fires*, carrying tokens from its input places, per the output arc expressions, to its output places.

In CP-nets, tokens carry information, called *color* [6]. Token color may be simple or complex, e.g. a tuple. Each place contains only tokens of a specified color. CP-net arc expressions are also extended, to allow complex expressions over colored token variables associated with the corresponding arcs. CP-nets also use *transition guards,* boolean expressions over token color attributes, which determine transition firing.

CP-nets contain additional extensions, which can be useful in representing complex AUML features. Further detail can be found in [5].

We now turn to using Petri nets to explicitly represent multi-agent conversations. All Petri net representation approaches of this type use places to represent interaction states, and Petri net transitions to represent transitions between interaction states. Net marking represents the current state of interaction. However, previous investigations take different design choices within this general approach.

Individual roles and CP-nets. Most investigations choose to separately represent individual roles within the interaction, rather than represent joint interaction states. In this approach, separate places are used for separate roles in the interaction, and thus different markings distinguish a conversation state where one agent has sent a message, from a state where the other agent received it. Typically, the net for each individual role is built separately, and then these nets are either merged into a single net [2,8,9], or simply connected together using Petri net fusion places, or other means [3,14]. All these investigations use CP-nets to represent multi-agent interactions. As shown later in the paper, the use of token color allows compact representation of multiple conversations using the same net.

Joint-state representations using PT-nets. In contrast, a limited number of investigations model conversations using PT-nets with joint conversation states [10,13].[1] In joint state representations, each net place is at once a representative of the conversation state of all agents. Typically, markings represent only valid conversation states (thus the nets ignore transmission delay, etc.), and synchronization protocols are implicitly assumed to underlie the conversation, to make sure that the agents are synchronized [12].

3 Analysis of Key Representations

The survey of related work presented above indicates that previous investigations have introduced rather different approaches to the modeling of multi-agent interactions using Petri nets. This section offers a comparative analysis of these approaches on the basis of several criteria: scalability (Section 3.1), and suitability for monitoring tasks (Section 3.2).

3.1 Scalability

We have classified previous approaches based on (i) their representation of individual conversation states vs. joint states, and on (ii) their utilization of token color. We now show how these two independent features affect the scalability of the chosen representation in terms of the number of conversations.

In principle, for a conversation that has R roles, with M messages, a representation which explicitly differentiates the conversation state of each role would have O(MR)

[1] Though authors claim otherwise, they in fact ignore color, using CP-nets as if they were PT-nets. For instance, Nowostawski et al. [10] duplicate portions of Petri net to represent multiple conversations, rather than using color tokens within a single net.

places: For every message there would be two individual places for the sender (before sending, and after sending), and similarly two more for each receiver (before receiving and after receiving). All possible joint states (i.e. message sent and received, sent and not received, not sent but incorrectly believed to have been received, not sent and not received) can be represented. In cases where all joint states must be represented (including all erroneous states), this representation is preferable to an explicit joint-state representation which would require $O(M^R)$ places.

However, many applications only require representation of valid conversation states (message not sent and not received, or sent and received). For instance, the specification of the FIPA interaction protocols [4] implicitly assumes the use of underlying synchronization protocols to guarantee delivery of messages [12]. Under such assumption, for every message, there are only two joint states regardless of the number of roles: before the message is sent, and after the message is sent and received. The number of places representing joint conversation states grows (linearly) in this case only with the number of messages – $O(M)$.

We now turn to examining the use of color tokens. In principle, CP-nets and PT-nets are equivalent from a computational perspective [6], in much the same way the high level programming languages are no more powerful in principle than assembly. However, when representing conversations, a significant difference between PT-nets and CP-nets is their scalability. A PT-token corresponds to a single bit. The information it conveys is a function of the place it is marking. As a result, it is impossible to represent several concurrent conversations in the same PT-net, since the tokens representing the different states of the conversations may overwrite each other, or cause the net to fire erroneously. Therefore, representing C concurrent conversations–all of the same interaction protocol–would require $O(C)$ PT-nets.

In contrast, however, colored tokens can be differentiated, even when multiple tokens mark the same net. For instance, in the representation we present in Section 4, token colors carry information about the sender and receivers of messages, about the time in which the message was sent, etc. This information allows us to represent multiple concurrent conversations–of the same protocol–on a single CP-net structure. Note that we save only on the number of nets explicitly represented–the number of tokens for representing C conversations is $O(C)$ in either a PT-net or CP-net approach.

There are some additional differences between CP-nets and PT-nets, in terms of features that support representation of FIPA interaction protocols, such as guards, sequence expressions, cardinalities and timing [4]. Representation of FIPA attributes is straightforward using the additional information carried by token color (a more detailed discussion can be found in [5]).

Table 1. Scalability Comparison.

	PT-nets	CP-nets
Individual States	**Space: O(MRC)**	**Space: O(MR)** [2],[3],[8],[9],[14]
Joint States	**Space: O(MC)** [10],[13]	**Space: O(M)**

Based on the above, it is possible to make concrete predictions as to the scalability of different approaches with respect to the number of agents. Table 1 above shows the space complexity of different approaches, given that we model *C* conversations, each with a maximum of *R* roles, and *M* messages. The table also cites relevant investigations.

3.2 Monitoring Conversations

There are many different uses for a representation of an interaction: To monitor its progress, to detect faults [13], to verify or analyze its features, etc. We focus here on monitoring, and distinguish two settings, depending on the information available to the monitor.

In the first type of setting, the monitor, representing the conversation, has access to the state of the conversation in one or more of the participants, but not to the messages being exchanged. This would be the case, for instance, if a participant in a conversation is monitoring its own progress. In this case, the participant has access to its own conversation state, but likely, does not have direct knowledge on whether messages were sent or received by others. Therefore, messages are not explicitly represented, except as transitions that take the conversation from one place to another (regardless of whether these places are represented individually or jointly). By placing tokens in the appropriate conversation places, an agents' state can be inferred. Then, letting the corresponding transition fire implies the message being sent and received. Previous works that have taken this approach include [2,8,9].[2]

In the second type of settings, the monitor has knowledge of the messages being sent and received, but does not necessarily know the internal conversation state. It monitors conversations by tracking the messages (e.g., through overhearing [7]). This could be done either from an individual perspective, or in settings of a global monitor that does not have direct knowledge of the conversation state of each agent. However, this requires the use of separate message places. In this type of representation, a state place and a message place are connected via a transition to a new state. A monitoring agent in this case places a token in the appropriate message place whenever it intercepts a message. Together with conversation state places, these tokens allow the conversation to transition from one conversation state to a new conversation state only based on explicit knowledge of the message being sent or received. In principle, given the current state, the new conversation state can be inferred from "observing" a message. Previous work that has used explicit message places include [2,3,10,13,14].

4 Scalable Representation for Overhearing

In this section, we focus on developing a scalable representation for overhearing. The design choices are dictated by the insights gained in the previous section. Thus, the clear choice in terms of scalability is the approach combining CP-nets with places representing joint interaction states. In addition, since in overhearing we only expect

[2] In the same publication, Cost et al. [2] also use the other approach.

to have knowledge of messages being exchanged, we use explicit message places. Unfortunately, previous investigations did not explore this design, though the work in [13] explores similar ideas using PT-nets.

We now show how various simple and complex AUML interaction features, used in FIPA conversation standards [4], can be implemented using the proposed CP-net representation.

We begin by examining a simple agent conversation building block, corresponding to a FIPA asynchronous message, which we first show in AUML (Figure 1-a) and then using our CP-net representation (Figure 1-b). Here, $agent_1$ sends an asynchronous message *msg* to $agent_2$. In Figure 1-a, the *msg* communicative act is shown by the arrow connecting the lifelines of the corresponding agents. The stick arrowhead denotes that *msg* is passed asynchronously (see [1,4,11] for AUML details).

To represent the same conversation using a CP net, we first identify net places and transitions. The representation we develop uses two types of places, corresponding to messages and joint conversation states (as previously described). Figure 1-b shows the asynchronous message implementation using our CP-net model. This CP-net shows three places and one transition connecting them. The A_1B_1 and the A_2B_2 places are agent places, while the *msg* place is a message place. The *A* and *B* capital letters are used to denote the $agent_1$ and the $agent_2$ individual interaction states respectively. We have indicated the individual and the joint interaction states over the AUML diagram in Figure 1-a, however these details are omitted later on in the paper. The A_1B_1 place indicates a joint interaction state where $agent_1$ is ready to send the *msg* message to $agent_2$ (A_1) and $agent_2$ is waiting to receive the corresponding message (B_1). The *msg* message place corresponds to the *msg* sent and received. The interception of the *msg* (and placing a corresponding token) causes the agents to transition to the A_2B_2 place. This place corresponds to the joint interaction state in which $agent_1$ has already sent the *msg* communicative act to $agent_2$ (A_2) who has received it (B_2).

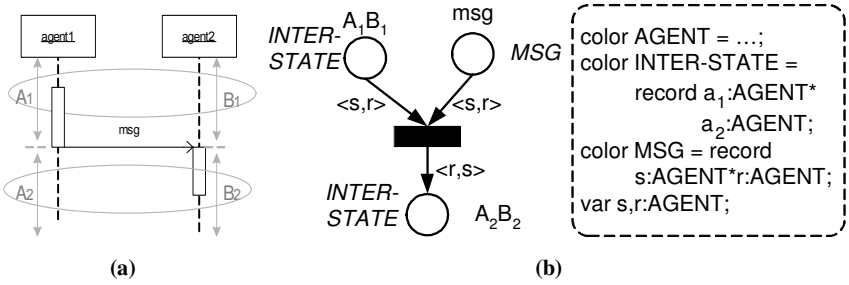

Figure 1. Asynchronous message interaction.
(a) AUML (b) CP-net representations.

The CP-net implementation in Figure 1-b introduces the use of token colors to represent additional information about agent interaction states and communicative acts of the corresponding interaction. The token color sets are defined in the net

declaration (dashed box in Figure 1-b). The syntax follows standard CP-notation [6]. The *AGENT* color is used to identify agents participating in the corresponding interaction. This color is further used to construct the two net compound color sets. The first color set is *INTER-STATE*. This color set is related to the net agent places and it is applied to represent agents corresponding to the appropriate joint interaction states. The *INTER-STATE* color token is a tuple (record) $<a_1,a_2>$, where a_1 and a_2 are *AGENT* color elements of the interacting agents. We apply the *INTER-STATE* color set to model concurrent conversations using the same CP-net. The second color set is *MSG*. The *MSG* color set describes interaction communicative acts and it is associated with the net message places. The *MSG* color token is a record $<s,r>$, where the s and r elements determine the sender and the receiver agents of the corresponding message.

Therefore, in Figure 1-b, the A_1B_1 and the A_2B_2 places are associated with the *INTER-STATE* color set, while the *msg* place is associated with the *MSG* color set. The place color set is written in italic capital letters next to the corresponding place. Furthermore, we use the s and r *AGENT* color type variables to denote the net arc expressions. Thus, given that the output arc expression of both the A_1B_1 and the *msg* places is $<s,r>$, the a_1 and a_2 elements of the agent place token must correspond to the s and r elements of the message place token. Consequently, the net transition occurs if and only if the addressed agents of the message correspond to the interacting agents.

Figures 2 through 4 show similar mappings between AUML representation of FIPA building blocks, and their CP-net equivalents. Figure 2 shows synchronous message passing, denoted through the filled solid arrowhead, meaning, that an acknowledgement of *msg* communicative act must always be received by $agent_1$ before the interaction protocol may proceed. Figure 3 shows a more complex interaction, called XOR-decision. In this interaction, the sender can send only one of the two possible messages to the designated recipients. The figure shows the use of a joint state for the three agents (the $A_1B_1C_1$ place). Figure 4 shows another complex interaction, the OR-parallel interaction, in which the sender can send one or two communicative acts (inclusively) to the designated recipients simulating an inclusive-or. As shown, $agent_1$ can send message msg_1 to $agent_2$ or message msg_2 to $agent_3$ or both.

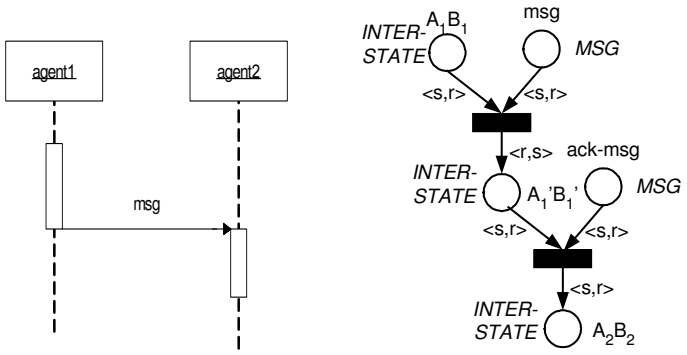

Figure 2. Synchronous message interaction.
(a) AUML (b) CP-net representations.

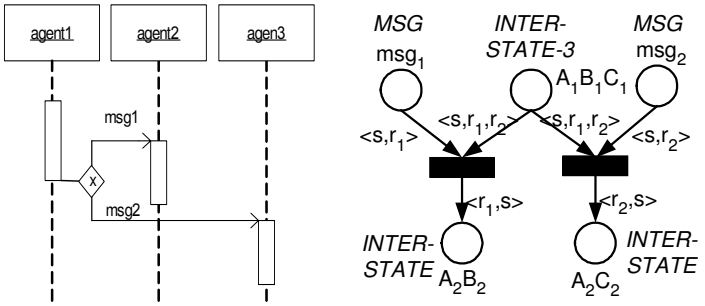

Figure 3. XOR-decision messages interaction.
(a) AUML (b) CP-net representations.

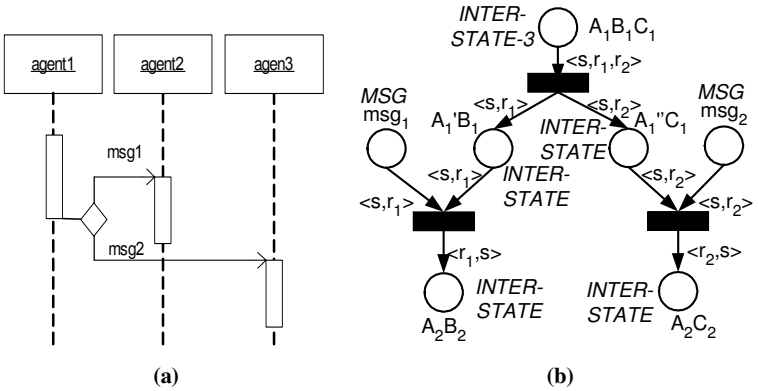

Figure 4. OR-parallel messages interaction.
(a) AUML (b) CP-net representations.

We now extend our technique to facilitate the implementation of additional interaction aspects useful in describing multi-agent conversation protocols. First, we use CP-nets to represent interaction message attributes used by FIPA conversation standards such as guards, sequence expressions, cardinalities, etc [4]. Second, we demonstrate representation of multiple agent concurrent conversations using the same CP-net.

Figure 5-a demonstrates a conditional agent interaction using AUML. This interaction is similar to Figure 1-a above, except for the use of the message guard-condition *[condition]*. Its semantics are that *msg* is sent if and only if the *condition* is true. Fortunately, message guard-conditions can be mapped directly to a CP-net transition guard (indicated next to the corresponding transition using square brackets in Figure 5-b). The transition guard guarantees that the transition is enabled if and only if the transition guard is true.

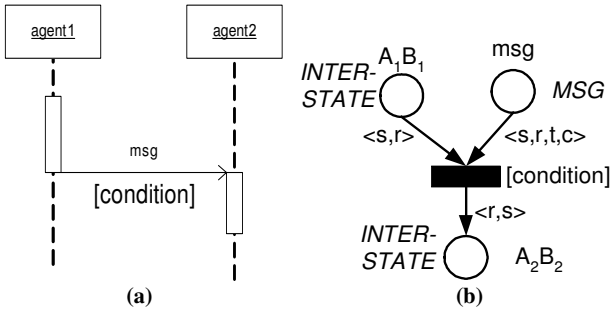

Figure 5. Message guard-condition.
(a) AUML (b) CP-net representations.

In Figure 5-b, we also demonstrate the CP-net implementation to message type and content attributes. For that purpose, we define two additional colors. The first, *TYPE* color, determine a message type, while the second, *CONTENT* color, represents message content. Furthermore, we extend the *MSG* color set, previously defined, to allow information passing between agents. Thus, the *MSG* color token is a record <*s,r,t,c*>, where the *s* and *r* elements has previous interpretation and the *t* and *c* elements define the message type and content.

Additional communicative act attributes include message sequence-expression and cardinality. In FIPA [4], sequence-expressions denote a constraint on the message sent from an agent: *m* denotes that the message is sent exactly *m* times; *n..m* denotes that the message is sent *n* up to *m* times; *{*}* denotes that the message is sent an arbitrary number of times.

In this paper, we focus on a non-FIPA extension commonly used–the broadcast sequence expression, which denotes the broadcast sending of a message to all recipients on a list. In Figure 6 we show its representation using CP-nets. For this purpose, we define an *INTER-STATE-CARD* color set. This color set is a tuple (<a_1,a_2>, *i*) consisting of two elements. The first tuple element is an *INTER-STATE* color element, which denotes the interacting agents as before. The second tuple element is an integer *i* that counts the number of messages already sent by an agent–message cardinality. This element is initially assigned to 0. The S_1R_1 place is of color *INTER-STATE-CARD*. Two additional colors are *BROADCAST-LIST* (defining the sender's list of receivers) and *TARGET* (index into this list).

The key novelty in Figure 6 is the use of the condition on the first transition, coupled with the arc looping back to S_1R_1. The initial marking of S_1R_1 is a single token (<*s,TARGET*(0)>,0), pointing at the first receiver on the broadcast list as the target, with message cardinality counter initiated to 0. On the other hand, the msg_1 message place initially contains multiple tokens. Each of these tokens represents the msg_1 message addressed to a designated receiver on the broadcast list. The S_1R_1 place token and the appropriate msg_1 place token together enable the corresponding transition. It fires, thus representing the sending of msg_1 to the first receiver on the broadcast list.

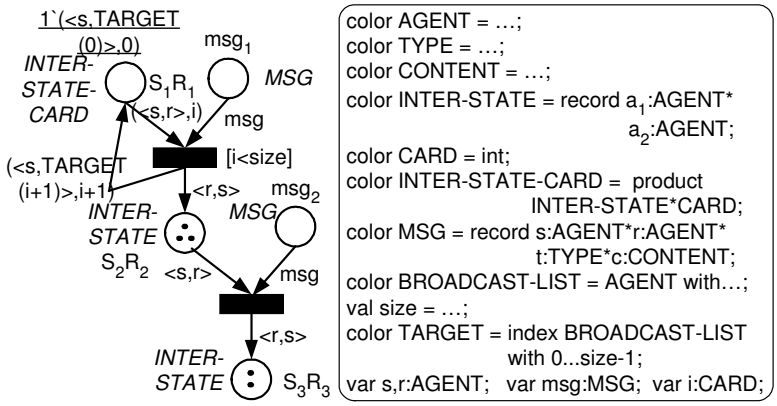

Figure 6. Broadcast in CP-net representation.

The arc looping back to S_1R_1 has an arc expression which increments the index *i*. Thus after the initial firing, a new token is placed in S_1R_1, pointing at the next recipient on the broadcast list. This recipient is matched with the appropriate token in the msg_1 place, and again the transition would fire, indicating transmission and receipt of msg_1 by the second receiver. The process continues while the condition on the transition holds, i.e., while the index *i* is smaller the size of the broadcast list.

The use of token color allows multiple conversations to be concurrently tracked using the same CP-net. For instance, in Figure 6, let the sender agent be called $agent_1$ and its broadcast list contain agents $agent_2$,…, $agent_6$. Suppose $agent_1$ has already sent msg_1 to all agents on the broadcast list, but has only received the msg_2 reply from $agent_3$, $agent_4$ and $agent_6$. The CP-net marking for this state would be: (i) The S_2R_2 place marked {<$agent_2$, $agent_1$>, <$agent_5$, $agent_1$>}; and (ii) the S_3R_3 place marked {<$agent_1$, $agent_3$>, <$agent_1$, $agent_4$>, <$agent_1$, $agent_6$>}. The different tokens, that are distinguishable because of the token color, differentiate concurrent conversations involving $agent_1$, using the same CP-net. This is a significant improvement over PT-net representations.

Due to space constraints, we cannot show how the proposed CP-net representation is amenable to represent *all* FIPA AUML building blocks (and additional features, such as deadlines and nested protocols). The reader is referred to [5] for such details.

5 Algorithm & Concluding Example

Previous investigations have explored various machine-readable Petri net representations. However, interaction protocols are typically specified in human-readable form (e.g., in AUML [1,11]). The question of how to systematically translate an interaction protocol specification into a machine-readable form has been previously ignored. We present a semi-automated procedure for transforming an AUML protocol diagram of two interacting agents to its CP-net representation. While not fully automated, we believe that it represents a significant step towards fully

automatic translation. We apply this algorithm on a complex multi-agent conversation protocol that involves many of the interaction aspects already discussed.

The procedure is presented in Figure 7. Its input is an AUML diagram, and its output is a corresponding CP-net representation using joint states and explicit message places. The CP-net is constructed in iterations: The algorithm essentially creates the conversation net by exploring the interaction protocol breadth-first, while avoiding cycles. Lines 1-2 create and initiate a queue and the output CP-net respectively. The queue, denoted by *S*, holds the initiating agent places for the current iteration. These places correspond to interaction states that initiate further conversation between the interacting agents. In lines 4-5, an initial agent place, A_1B_1, is created and inserted into the queue.

We enter the main loop in line 8 and set *curr* to the first initiating agent place in *S*. Lines 10-13 create the CP-net components of the current iteration. First, in line 10, message places, associated with *curr* agent place, are created using *CreateMessagePlaces*. These places correspond to communicative acts, which take agents from the joint interaction state *curr* to its successor(s). Then, in line 11, we create agent places that correspond to interaction state changes as a result of these messages associated with *curr* agent place. Then, in *CreateTransitionsAndArcs* (line 12), these places are connected through transitions and arcs, using the CP-net building blocks described previously, and in [5]. Finally, we add token color elements to the CP-net structure, implementing attributes using *FixColor* (line 13).

Algorithm *CreateConversationNet* (**input :** *AUML*,**output :** *CPN*)

```
1: S      ← new queue
2: CPN    ← new CP-net
3:
4: A1B1   ← new agent place with color information
5: S.enqueue (A1B1)
6:
7: while S not empty do
8:   curr     ← S.dequeue ()
9:
10:  MP       ← CreateMessagePlaces (AUML, curr)
11:  RP       ← CreateResultingAgentPlaces (AUML, curr, MP)
12:  (TR, AR) ← CreateTransitionsAndArcs (AUML, curr, MP, RP)
13:  FixColor (AUML, CPN, MP, RP, TR, AR)
14:
15:  foreach place p in RP
16:    if p was not created in current iteration
17:      continue
18:    if p is not terminating place
19:      S.enqueue (p)
20:  end foreach
21:
22:  CPN.places      = CPN.places ⋃ MP ⋃ RP
23:  CPN.transitions = CPN.transitions ⋃ TR
24:  CPN.arcs        = CPN.arcs ⋃ AR
25: end while
26:
27: return CPN
```

Figure 7. AUML to CPN Conversion Procedure.

Lines 15-20 determine agent places that are inserted into *S* for further iteration. Only non-terminating agent places, corresponding to non-terminal interaction states, are inserted into *S* (lines 18-19), with the exception of places that have already been handled (lines 16-17). Completing the iteration, the output CP-net, denoted by *CPN*, is updated according to the current iteration CP-net components in lines 22-24. The loop iterates as long as *S* contains places that have not been handled. Finally, the resulting CP-net is returned (line 27).

To demonstrate this algorithm, we now use it to construct a CP-net of the FIPA Contract Net Interaction Protocol [4] (shown in AUML in Figure 8). In this protocol, the *Initiator* agent issues *m* calls for proposals using a *cfp* message. By a given deadline, each of the *Participants* may send either a *refuse* message (terminating the interaction), or a *propose* message containing a counter-proposal. Once the deadline expires, the *Initiator* evaluates received proposals and selects agents to perform the requested task. Selected participants are sent an *accept-proposal* message, while others are sent a *reject-proposal*. Selected participants carry out their task, and upon completion, send either an *inform-done*, an *inform-result*, or a *failure* message.

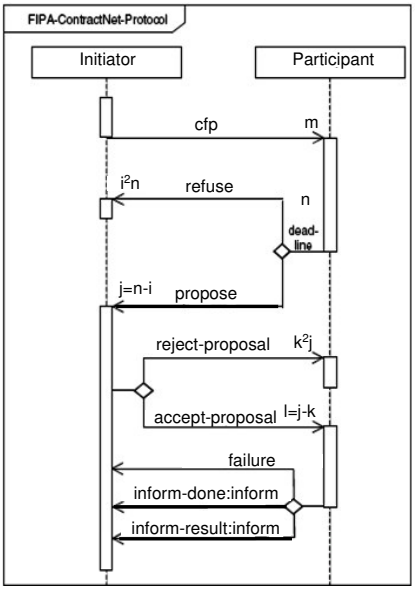

Figure 8. FIPA Contract Net using AUML.

We now use the algorithm introduced above to create a CP-net for this protocol, in four iterations of the main loop. The algorithm begins with the creation (and insertion into *S*) of the I_1P_1 place, of *INTER-STATE* color. Thus, in the first iteration, the *curr* variable is set to I_1P_1. The algorithm creates net places, which are associated with the I_1P_1 place, i.e. a *cfp* message place and an I_2P_2 resulting agent place. Then, the three places are connected using the asynchronous message building block shown in Figure 1-b. Next, the color sets of the corresponding places are determined, and the

algorithm also handles the broadcast sequence-expression attribute of the *cfp* message, as shown in Figure 6. Accordingly, the color set associated with I_1P_1 place, is changed to the *INTER-STATE-CARD* color set. The I_2P_2 is not a terminating place (*Initiator* is waiting for a response from *Participants*) and is thus inserted into the *S* queue.

In the second iteration, *curr* is set to the I_2P_2 place. A *Participant* can send either a *refuse* or a *propose* messages, and thus appropriate message places are created. Then, the I_3P_3 and I_4P_4 agent places, corresponding to the results of the messages, are created. The I_2P_2, *Refuse*, I_3P_3, *Propose* and I_4P_4 places are connected using the XOR-decision described in Figure 3-b. Then, the deadline sequence expression of both the *refuse* and the *propose* messages is implemented as shown in [5]. The I_3P_3 place (resulting from *refuse*) is a terminal interaction state, while the I_4P_4 place represents a non-terminal state. Thus, only I_4P_4 is inserted into *S*.

For lack of space, we now skip over the final two iterations of the main loop, to the resulting CP net (Figure 9). The only items of interest in these skipped iterations involve the creation of the guard conditions on the transitions (see Figure 5-b), and the abstraction of the two inform messages (*inform-done*, *inform-result*) into a single message place marked *inform*. A detailed discussion of their creation is provided in [5].

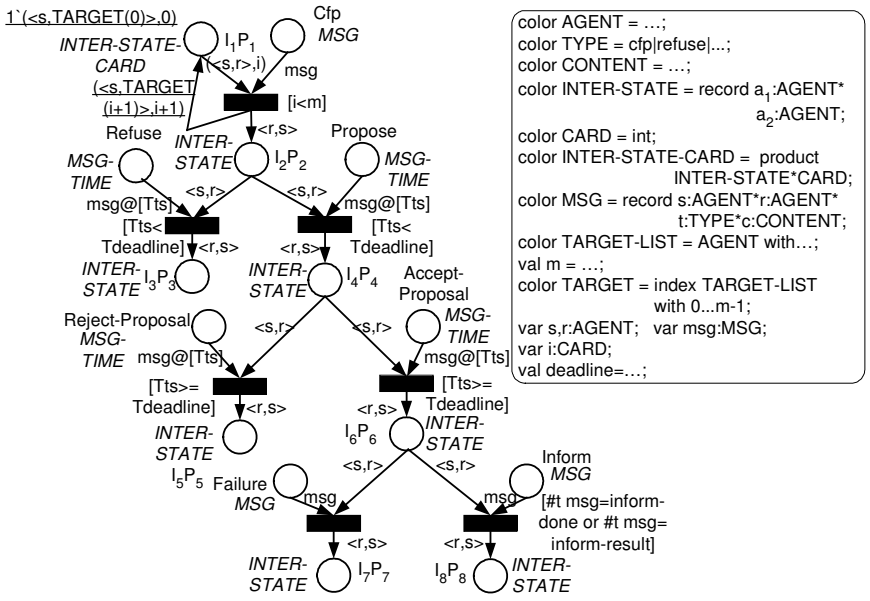

Figure 9. FIPA Contract Net using CP-net.

Although this procedure can convert many 2-agent protocols in AUML to their CP-net equivalents, it does not address the general *n*-agent case. We leave this development to future work.

6 Summary & Conclusions

Over recent years, increasing attention has been directed at representations of agent conversations. In particular, there is an increasing interest in using Petri nets to model multi-agent interactions [2,9,10,14]. Unfortunately, features of competing approaches with respect to scalability and suitability for different tasks have not been analyzed. Furthermore, no procedures were provided that guide the conversion of an interaction protocol given in AUML (the FIPA standard human-readable representation [4,11]) to any of the Petri net representations.

This paper sought to address these open questions. First, we analyzed key features in existing representation approaches. We have shown that (i) when representing valid conversations, a CP-net, where places denote joint conversation states, scales better than other approaches; (ii) message places are necessary for tracking conversations by overhearing. Unfortunately, previous work did not examine this combination of CP-nets with joint states and message places.

We therefore developed this representation to target scalable overhearing and monitoring tasks. We provided building blocks allowing this representation to model complex multi-agent conversations as defined by FIPA [4]. Finally, we have presented a skeleton semi-automated procedure for converting an AUML protocol diagrams to an equivalent CP-net, and demonstrated its use on a challenging FIPA conversation protocol.

We believe that the proposed technique can assist and motivate continuing research on representing conversations for tasks other than overhearing, e.g., debugging [13], automated monitoring [7], etc.

References

1. AUML site (2004). Agent UML, at www.auml.org.
2. Cost, R. S., Chen, Y., Finin, T., Labrou, Y. & Peng, Y. (2000). Using Coloured Petri Nets for a Conversation Modeling. In Dignum, F. & Greaves, M. (Eds.), Issues in Agent Communications, pp. 178-192. Springer-Verlag.
3. Cranefield S., Purvis M., Nowostawski M. & Hwang P. (2002). Ontologies for interaction protocols. In Proceedings of AAMAS-02.
4. FIPA Specifications (2004). FIPA Specifications, at www.fipa.org/specifications/index.html.
5. Gutnik, G. & Kaminka, G.A. (2004). A comprehensive Petri net representation for multi-agent conversations. MAVERICK Technical Report 2004/1, Bar-Ilan University, at www.cs.biu.ac.il/~maverick/tech-reports/.
6. Jensen, K. (1997). Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag.
7. Kaminka, G.A., Pynadath, D.V. & Tambe, M. (2002). Monitoring Teams by Overhearing: A Multi-Agent Plan-Recognition Approach. JAIR, 17, 83-135.
8. Lin, F., Norrie, D. H., Shen, W. & Kremer, R. (2000). A schema-based approach to specifying conversation policies. In Dignum, F. & Greaves, M. (Eds.), Issues in Agent Communications, pp. 193-204. Springer-Verlag.
9. Mazouzi, H., Fallah-Seghrouchni, A. E. & Haddad, S. (2002). Open protocol design for complex interactions in multi-agent systems. In Proceedings of AAMAS-02.

10. Nowostawski, M., Purvis, M. & Cranefield, S. (2001). A layered approach for modeling agent conversations. In Proceedings of Workshop on Infrastructure for Agents, MAS and Scalable MAS, pp. 163-170. Montreal, Canada.
11. Odell, J., Parunak, H. V. D. & Bauer, B. (2001). Agent UML: A formalism for specifying multi-agent interactions. In Ciancarini, P. & Wooldridge, M. (Eds.), Agent-Oriented Software Engineering, pp. 91-103. Springer-Verlag, Berlin.
12. Paurobally S., Cunningham J. & Jennings N. R. (2003). Ensuring consistency in the joint beliefs of interacting agents. In Proceedings of AAMAS-03.
13. Poutakidis, D., Padgham, L. & Winikoff, M. (2002). Debugging multi-agent systems using design artifacts. In Proceedings of AAMAS-02.
14. Purvis, M. K., Hwang, P., Cranefield, S. J. & Schievink, M. (2002). Interaction Protocols for a Network of Environmental Problem Solvers. In Proceedings of iEMSs-02.