Journal of Artificial Intelligence Research 25 (2006) 349-387 Submitted 06/05; published 03/06

Representing Conversations for Scalable Overhearing

Gery Gutnik GUTNIKG@CS.BIU.AC.IL
Gal A. Kaminka GALK@CS.BIU.AC.IL
Computer Science Department
Bar Ilan University
Ramat Gan 52900, Israel

Abstract

Open distributed multi-agent systems are gaining interest in the academic community and in industry. In such open settings, agents are often coordinated using standardized agent conversation protocols. The representation of such protocols (for analysis, validation, monitoring, etc) is an important aspect of multi-agent applications. Recently, Petri nets have been shown to be an interesting approach to such representation, and radically different approaches using Petri nets have been proposed. However, their relative strengths and weaknesses have not been examined. Moreover, their scalability and suitability for different tasks have not been addressed. This paper addresses both these challenges. First, we analyze existing Petri net representations in terms of their scalability and appropriateness for overhearing, an important task in monitoring open multi-agent systems. Then, building on the insights gained, we introduce a novel representation using Colored Petri nets that explicitly represent legal joint conversation states and messages. This representation approach offers significant improvements in scalability and is particularly suitable for overhearing. Furthermore, we show that this new representation offers a comprehensive coverage of all conversation features of FIPA conversation standards. We also present a procedure for transforming AUML conversation protocol diagrams (a standard human-readable representation), to our Colored Petri net representation.

1. Introduction

Open distributed multi-agent systems (MAS) are composed of multiple, independently-built agents that carry out mutually-dependent tasks. In order to allow inter-operability of agents of different designs and implementation, the agents often coordinate using standardized interaction protocols, or conversations. Indeed, the multi-agent community has been investing a significant effort in developing standardized *Agent Communication Languages* (ACL) to facilitate sophisticated multi-agent systems (Finin, Labrou, & Mayfield, 1997; Kone, Shimazu, & Nakajima, 2000; ChaibDraa, 2002; FIPA site, 2003). Such standards define communicative acts, and on top of them, *interaction protocols*, ranging from simple queries as to the state of another agent, to complex negotiations by auctions or bidding on contracts. For instance, the *FIPA Contract Net Interaction Protocol* (FIPA Specifications, 2003b) defines a concrete set of message sequences that allows the interacting agents to use the contract net protocol for negotiations.

Various formalisms have been proposed to describe such standards (e.g., Smith & Cohen, 1996; Parunak, 1996; Odell, Parunak, & Bauer, 2000, 2001b; AUML site, 2003). In particular, AUML–*Agent Unified Modelling Language*–is currently used in the FIPA-ACL standards

(FIPA Specifications, 2003a, 2003b, 2003c, 2003d; Odell, Parunak, & Bauer, 2001a) [1]. UML 2.0 (AUML site, 2003), a new emerging standard influenced by AUML, has the potential to become the FIPA-ACL standard (and a forthcoming IEEE standard) in the future. However, for the moment, a large set of FIPA specifications remains formalized using AUML. While AUML is intended for human readability and visualization, interaction protocols should ideally be represented in a way that is amenable to automated analysis, validation and verification, online monitoring, etc.

Lately, there is increasing interest in using Petri nets (Petri Nets site, 2003) in modelling multi-agent interaction protocols (Cost, 1999; Cost, Chen, Finin, Labrou, & Peng, 1999, 2000; Lin, Norrie, Shen, & Kremer, 2000; Nowostawski, Purvis, & Cranefield, 2001; Purvis, Hwang, Purvis, Cranefield, & Schievink, 2002; Cranefield, Purvis, Nowostawski, & Hwang, 2002; Ramos, Frausto, & Camargo, 2002; Mazouzi, Fallah-Seghrouchni, & Haddad, 2002; Poutakidis, Padgham, & Winikoff, 2002). There is broad literature on using Petri nets to analyze the various aspects of distributed systems (e.g. in deadlock detection as shown by Khomenco & Koutny, 2000), and there has been recent work on specific uses of Petri nets in multi-agent systems, e.g., in validation and testing (Desel, Oberweis, & Zimmer, 1997), in automated debugging and monitoring (Poutakidis et al., 2002), in dynamic interpretation of interaction protocols (Cranefield et al., 2002; de Silva, Winikoff, & Liu, 2003), in modelling agents behavior induced by their participation in a conversation (Ling & Loke, 2003) and in interaction protocols refinement allowing modular construction of complex conversations (Hameurlain, 2003).

However, key questions remain open on the use of Petri nets for conversation representation. First, while radically different approaches to representation using Petri nets have been proposed, their relative strengths and weaknesses have not been investigated. Second, many investigations have only addressed restricted subsets of the features needed in representing complex conversations such as those standardized by FIPA (see detailed discussion of previous work in Section 2). Finally, no procedures have been proposed for translating human-readable AUML protocol descriptions into the corresponding machine-readable Petri nets.

This paper addresses these open challenges in the context of scalable *overhearing*. Here, an overhearing agent passively tracks many concurrent conversations involving multiple participants, based solely on their exchanged messages, while not being a participant in any of the overheard conversations itself (Novick & Ward, 1993; Busetta, Serafini, Singh, & Zini, 2001; Kaminka, Pynadath, & Tambe, 2002; Poutakidis et al., 2002; Busetta, Dona, & Nori, 2002; Legras, 2002; Gutnik & Kaminka, 2004a; Rossi & Busetta, 2004). Overhearing is useful in visualization and progress monitoring (Kaminka et al., 2002), in detecting failures in interactions (Poutakidis et al., 2002), in maintaining organizational and situational awareness (Novick & Ward, 1993; Legras, 2002; Rossi & Busetta, 2004) and in non-obtrusively identifying opportunities for offering assistance (Busetta et al., 2001, 2002). For instance, an overhearing agent may monitor the conversation of a contractor agent engaged in multiple contract-net protocols with different bidders and bid callers, in order to detect failures.

We begin with an analysis of Petri net representations, with respect to scalability and overhearing. We classify representation choices along two dimensions affecting scalability:

1. (FIPA Specifications, 2003c) is currently deprecated. However, we use this specification since it describes many important features needed in modelling multi-agent interactions.

(i) the technique used to represent multiple concurrent conversations; and (ii) the choice of representing either individual or joint interaction states. We show that while the run-time complexity of monitoring conversations using different approaches is the same, choices along these two dimensions have significantly different space requirements, and thus some choices are more scalable (in the number of conversations) than others. We also argue that representations suitable for overhearing require the use of explicit message places, though only a subset of previously-explored techniques utilized those.

Building on the insights gained, the paper presents a novel representation that uses Colored Petri nets (CP-nets) in which places explicitly denote messages, and valid joint conversation states. This representation is particularly suited for overhearing as the number of conversations is scaled-up. We show how this representation can be used to represent essentially all features of FIPA AUML conversation standards, including simple and complex interaction building blocks, communicative act attributes such as message guards and cardinalities, nesting, and temporal aspects such as deadlines and duration.

To realize the advantages of machine-readable representations, such as for debugging (Poutakidis et al., 2002), existing human-readable protocol descriptions must be converted to their corresponding Petri net representations. As a final contribution in this paper, we provide a skeleton semi-automated procedure for converting FIPA conversation protocols in AUML to Petri nets, and demonstrate its use on a complex FIPA protocol. While this procedure is not fully automated, it takes a first step towards addressing this open challenge.

This paper is organized as follows. Section 2 presents the motivation for our work. Sections 3 through 6 then present the proposed representation addressing all FIPA conversation features including basic interaction building blocks (Section 3), message attributes (Section 4), nested & interleaved interactions (Section 5), and temporal aspects (Section 6). Section 7 ties these features together: It presents a skeleton algorithm for transforming an AUML protocol diagram to its Petri net representation, and demonstrates its use on a challenging FIPA conversation protocol. Section 8 concludes. The paper rounds up with three appendixes. The first provides a quick review of Petri nets. Then, to complete coverage of FIPA interactions, Appendix B provides additional interaction building blocks. Appendix C presents a Petri net of a complex conversation protocol, which integrates many of the features of the developed representation technique.

2. Representations for Scalable Overhearing

Overhearing involves monitoring conversations as they progress, by tracking messages that are exchanged between participants (Gutnik & Kaminka, 2004a). We are interested in representations that can facilitate *scalable* overhearing, tracking many concurrent conversations, between many agents. We focus on open settings, where the complex internal state and control logic of agents is not known in advance, and therefore exclude discussions of Petri net representations which explicitly model agent internals (e.g., Moldt & Wienberg, 1997; Xu & Shatz, 2001). Instead, we treat agents as black boxes, and consider representations that commit only to the agent's conversation state (i.e., its role and progress in the conversation).

The suitability of a representation for scalable overhearing is affected by several facets. First, since overhearing is based on tracking messages, the representation must be able to explicitly represent the passing of a message (communicative act) from one agent to another

(Section 2.1). Second, the representation must facilitate tracking of multiple concurrent conversations. While the tracking runtime is bounded from below by the number of messages (since in any case, all messages are overheard and processed), space requirements may differ significantly (see Sections 2.2–2.3).

2.1 Message-monitoring versus state-monitoring

We distinguish two settings for tracking the progress of conversations, depending on the information available to the tracking agent. In the first type of setting, which we refer to as *state monitoring*, the tracking agent has access to the *internal* state of the conversation in one or more of the participants, but not necessarily to the messages being exchanged. The other settings involves *message monitoring*, where the tracking agent has access only to the messages being exchanged (which are *externally observable*), but cannot directly observe the internal state of the conversation in each participant. Overhearing is a form of message monitoring.

Representations that support state monitoring use places to denote the conversation states of the participants. Tokens placed in these places (the net marking) denote the current state. The sending or receiving of a message by a participant is not explicitly represented, and is instead implied by moving tokens (through transition firings) to the new state places. Thus, such a representation essentially assumes that the internal conversation state of participants is directly observable by the monitoring agent. Previous work utilizing state monitoring includes work by Cost (1999), Cost et al. (1999, 2000), Lin et al. (2000), Mazouzi et al. (2002), Ramos et al. (2002).

The representation we present in this paper is intended for overhearing tasks, and cannot assume that the conversation states of overheard agents are observable. Instead, it must support message monitoring, where in addition to using tokens in state places (to denote current conversation state), the representation uses *message places*, where tokens are placed when a corresponding message is overheard. A conversation-state place and a message place are connected via a transition to a state place denoting the new conversation state. Tokens placed in these originating places–indicating a message was received at an appropriate conversation state–will cause the transition to fire, and for the tokens to be placed in the new conversation state place. Thus the new conversation state is inferred from "observing" a message. Previous investigations, that have used explicit message places, include work by Cost (1999), Cost et al. (1999, 2000), Nowostawski et al. (2001), Purvis et al. (2002), Cranefield et al. (2002), Poutakidis et al. (2002)[2]. These are discussed in depth below.

2.2 Representing a Single Conversation

Two representation variants are popular within those that utilize conversation places (in addition to message places): *Individual state representations* use separate places and tokens for the state of each participant (each role). Thus, the overall state of the conversation is represented by different tokens marking multiple places. *Joint state representations* use a single place for each joint conversation state of all participants. The placement of a token

2. Cost (1999), Cost et al. (1999, 2000) present examples of both state- and message- monitoring representations.

within such a place represents the overhearing agent's belief that the participants are in the appropriate joint state.

Most previous representations use individual states. In these, different markings distinguish a conversation state where one agent has sent a message, from a state where the other agent received it. The net for each conversation role is essentially built separately, and is merged with the other nets, or connected to them via fusion places or similar means.

Cost (1999), Cost et al. (1999, 2000) have used CP-nets with individual state places for representing KQML and FIPA interaction protocols. Transitions represent message events, and CP-net features, such as token colors and arc expressions, are used to represent AUML message attributes and sequence expressions. The authors also point out that deadlines (a temporal aspect of interaction) can be modelled, but no implementation details are provided. Cost (1999) also proposed using hierarchical CP-nets to represent hierarchical multi-agent conversations.

Purvis et al. (2002), Cranefield et al. (2002) represented conversation roles as separate CP-nets, where places denote both interaction messages and states, while transitions represent operations performed on the corresponding communicative acts such as send, receive, and process. Special in/out places are used to pass net tokens between the different CP-nets, through special get/put transitions, simulating the actual transmission of the corresponding communicative acts.

In principle, individual-state representations require two places in each role, for every message. For a given 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 conversation states–valid or not–can be represented. For a single message and two roles, there are two places for each role (four places total), and four possible conversation states: message sent and received, sent and not received, not sent but incorrectly believed to have been received, not sent and not received. These states can be represented by different markings. For instance, a conversation state where the message has been sent but not received is denoted by a token in the *'after-sending'* place of the *sender* and another token in the *'before-receiving'* place of the *receiver*. This is summarized in the following proposition:

Proposition 1 *Given a conversation with R roles and a total of M possible messages, an individual state representation has space complexity of $O(MR)$.*

While the representations above all represent each role's conversation state separately, many applications of overhearing only require representation of valid conversation states (message not sent and not received, or sent and received). Indeed, specifications for interaction protocols often assume the use of underlying synchronization protocols to guarantee delivery of messages (Paurobally & Cunningham, 2003; Paurobally, Cunningham, & Jennings, 2003). Under such an assumption, for every message, there are only two joint states regardless of the number of roles. For example, for a single message and three roles–a sender and two receivers, there are two places and two possible markings: A token in a *before sending/receiving* place represents a conversation state where the message has not yet been sent by the *sender* (and the two *receivers* are waiting for it), while a token in a *after sending/receiving* place denotes that the message has been sent and received by both *receivers*.

Nowostawski et al. (2001) utilize CP-nets where places denote joint conversation states. They also utilize places representing communicative acts. Poutakidis et al. (2002) proposed a representation based on Place-Transition nets (PT-nets)–a more restricted representation of Petri nets that has no color. They presented several interaction building blocks, which could then fit together to model additional conversation protocols. In general, the following proposition holds with respect to such representations:

Proposition 2 *Given a conversation with R roles and a total of M possible messages, a joint state representation that represents only legal states has space complexity of $O(M)$.*

The condition of representing only *valid* states is critical to the complexity analysis. If all joint conversation states–valid and invalid–are to be represented, the space complexity would be $O(M^R)$. In such a case, an individual-state representation would have an advantage. This would be the case, for instance, if we do not assume the use of synchronization protocols, e.g., where the overhearing agent may wish to track the exact system state even while a message is underway (i.e., sent and not yet received).

2.3 Representing Multiple Concurrent Conversations

Propositions 1 and 2 above address the space complexity of representing a single conversation. However, in large scale systems an overhearing agent may be required to monitor multiple conversations in parallel. For instance, an overhearing agent may be monitoring a middle agent that is carrying multiple parallel instances of a single interaction protocol with multiple partners, e.g., brokering (FIPA Specifications, 2003a).

Some previous investigations propose to duplicate the appropriate Petri net representation for each monitored conversation (Nowostawski et al., 2001; Poutakidis et al., 2002). In this approach, every conversation is tracked by a separate Petri-net, and thus the number of Petri nets (and their associated tokens) grows with the number of conversations (Proposition 3). For instance, Nowostawski et al. (2001) shows an example where a contract-net protocol is carried out with three different contractors, using three duplicate CP-nets. This is captured in the following proposition:

Proposition 3 *A representation that creates multiple instances of a conversation Petri net to represent C conversations, requires $O(C)$ net structures, and $O(C)$ bits for all tokens.*

Other investigations take a different approach, in which a single CP-net structure is used to monitor all conversations of the same protocol. The tokens associated with conversations are differentiated by their token color (Cost, 1999; Cost et al., 1999, 2000; Lin et al., 2000; Mazouzi et al., 2002; Cranefield et al., 2002; Purvis et al., 2002; Ramos et al., 2002). For example, by assigning each token a color of the tuple type $\langle sender, receiver \rangle$, an agent can differentiate multiple tokens in the same place and thus track conversations of different pairs of agents[3]. Color tokens use multiple bits per token; up to $\log C$ bits are required to differentiate C conversations. Therefore, the number of bits required to track C conversations using C tokens is $C \log C$. This leads to the following proposition.

3. See Section 4 to distinguish between different conversations by the same agents.

Proposition 4 *A representation that uses color tokens to represent C multiple instances of a conversation, requires $O(1)$ net structures, and $O(C \log C)$ bits for all tokens.*

Due to the constants involved, the space requirements of Proposition 3 are in practice much more expensive than those of Proposition 4. Proposition 3 refers to the creation of $O(C)$ Petri networks, each with duplicated place and transition data structures. In contrast, Proposition 4 refers to bits required for representing C color tokens on a single CP net. Moreover, in most practical settings, a sufficiently large constant bound on the number of conversations may be found, which will essentially reduce the $O(\log C)$ factor to $O(1)$.

Based on Propositions 1–4, it is possible to make concrete predictions as to the scalability of different approaches with respect to the number of conversations, roles. Table 1 shows the space complexity of different approaches when modelling C conversations of the same protocol, each with a maximum of R roles, and M messages, under the assumption of underlying synchronization protocols. The table also cites relevant previous work.

	Representing Multiple Conversations (of Same Protocol)	
	Multiple CP- or PT-nets (Proposition 3)	**Using color tokens, single CP-net** (Proposition 4)
Individual States (Proposition 1)	**Space:** $O(MRC)$	**Space:** $O(MR + C \log C)$ Cost (1999), Cost et al. (1999, 2000), Lin et al. (2000), Cranefield et al. (2002), Purvis et al. (2002), Ramos et al. (2002), Mazouzi et al. (2002)
Joint States (Proposition 2)	**Space:** $O(MC)$ Nowostawski et al. (2001), Poutakidis et al. (2002)	**Space:** $O(M + C \log C)$ This paper

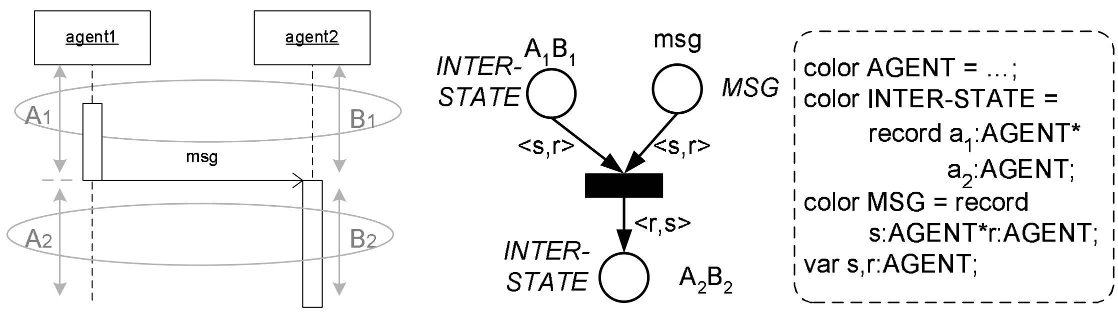
Table 1: Scalability of different representations

Building on the insights gained from Table 1, we propose a representation using CP-nets where places explicitly represent joint conversation states (corresponding to the lower-right cell in Table 1), and tokens color is used to distinguish concurrent conversations (as in the upper-right cell in Table 1). As such, it is related to the works that have these features, but as the table demonstrates, is a novel synthesis.

Our representation uses similar structures to those found in the works of Nowostawski et al. (2001) and Poutakidis et al. (2002). However, in contrast to these previous investigations, we rely on token color in CP-nets to model concurrent conversations, with space complexity $O(M + C \log C)$. We also show (Sections 3–6) how it can be used to cover a variety of conversation features not covered by these investigations. These features include representation of a full set of FIPA interaction building blocks, communicative act attributes (such as message guards, sequence expressions, etc.), compact modelling of concurrent conversations, nested and interleaved interactions, and temporal aspects.

3. Representing Simple & Complex Interaction Building Blocks

This section introduces the fundamentals of our representation, and demonstrates how various simple and complex AUML interaction messages, used in FIPA conversation standards (FIPA Specifications, 2003c), can be implemented using the proposed CP-net representation. We begin with a simple conversation, shown in Figure 1-a using an AUML protocol diagram. Here, $agent_1$ sends an asynchronous message msg to $agent_2$.



(a) AUML representation (b) CP-net representation

Figure 1: Asynchronous message interaction.

To represent agent conversation protocols, we define two types of places, corresponding to messages and conversation states. The first type of net places, called *message places*, is used to describe conversation communicative acts. Tokens placed in message places indicate that the associated communicative act has been overheard. The second type of net places, *agent places*, is associated with the valid *joint* conversation states of the interacting agents. Tokens placed in agent places indicate the current joint state of the conversation within the interaction protocol.

Transitions represent the transmission and receipt of communicative acts between agents. Assuming underlying synchronization protocols, a transition always originates within a joint-state place and a message place, and targets a joint conversation state (more than one is possible–see below). Normally, the current conversation state is known (marked with a token), and must wait the overhearing of the matching message (denoted with a token at the connected message place). When this token is marked, the transition fires, automatically marking the new conversation state.

Figure 1-b presents CP net representation of the earlier example of Figure 1-a. The CP-net in Figure 1-b has 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, but omit these annotations in later figures). Thus, the A_1B_1 place indicates a joint interaction state where $agent_1$ is ready to send the msg communicative act 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 communicative act sent between the two agents. Thus, the transmission of the msg communicative act 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) and $agent_2$ has received it (B_2).

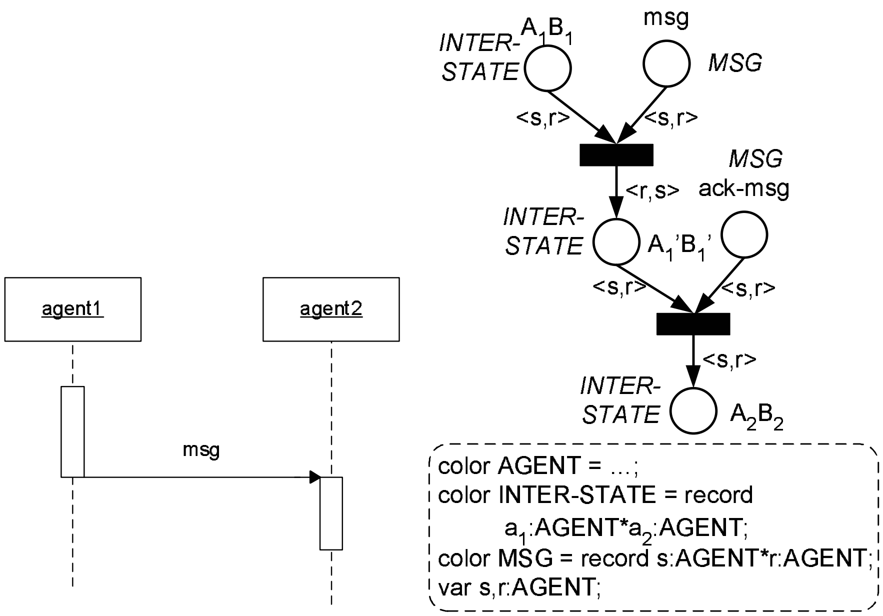
The CP-net implementation in Figure 1-b also introduces the use of token colors to represent additional information about interaction states and communicative acts. The token color sets are defined in the net declaration, i.e. the dashed box in Figure 1-b. The syntax follows the standard CPN ML notation (Wikstrom, 1987; Milner, Harper, & Tofte, 1990; Jensen, 1997a). The $AGENT$ color identifies the agents participating in the interaction, and is used to construct the two compound color sets.

The $INTER\text{-}STATE$ color set is associated with agent places, and represents agents in the appropriate joint interaction states. It is a record $\langle a_1, a_2 \rangle$, where a_1 and a_2 are $AGENT$ color elements distinguishing the interacting agents. We apply the $INTER\text{-}STATE$ color set to model multiple concurrent conversations using the same CP-net. The second color set is MSG, describing interaction communicative acts and associated with message places. The MSG color token is a record $\langle a_s, a_r \rangle$, where a_s and a_r correspond to the sender and the receiver agents of the associated communicative act. In both cases, additional elements, such as conversation identification, may be used. See Section 4 for additional details.

In Figure 1-b, the A_1B_1 and the A_2B_2 places are associated with the $INTER\text{-}STATE$ 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 $\langle s, r \rangle$, the s and r 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 agents of the message correspond to the interacting agents. The A_2B_2 place input arc expression is $\langle r, s \rangle$ following the underlying intuition that $agent_2$ is going to send the next interaction communicative act.

Figure 2-a shows an AUML representation of another interaction building block, synchronous message passing, denoted by the filled solid arrowhead. Here, the msg communicative act is sent synchronously from $agent_1$ to $agent_2$, meaning that an acknowledgement on msg communicative act must always be received by $agent_1$ before the interaction may proceed.

The corresponding CP-net representation is shown in Figure 2-b. The interaction starts in the A_1B_1 place and terminates in the A_2B_2 place. The A_1B_1 place represents a joint interaction state where $agent_1$ is ready to send the msg communicative act to $agent_2$ (A_1) and $agent_2$ is waiting to receive the corresponding message (B_1). The A_2B_2 place denotes a joint interaction state, in which $agent_1$ has already sent the msg communicative act to $agent_2$ (A_2) and $agent_2$ has received it (B_2). However, since the CP-net diagram represents synchronous message passing, the msg communicative act transmission cannot cause the agents to transition directly from the A_1B_1 place to the A_2B_2 place. We therefore define an intermediate $A_1'B_1'$ agent place. This place represents a joint interaction state where $agent_2$ has received the msg communicative act and is ready to send an acknowledgement on it (B_1'), while $agent_1$ is waiting for that acknowledgement (A_1'). Taken together, the msg communicative act causes the agents to transition from the A_1B_1 place to the $A_1'B_1'$ place, while the acknowledgement on the msg message causes the agents to transition from the $A_1'B_1'$ place to the A_2B_2 place.

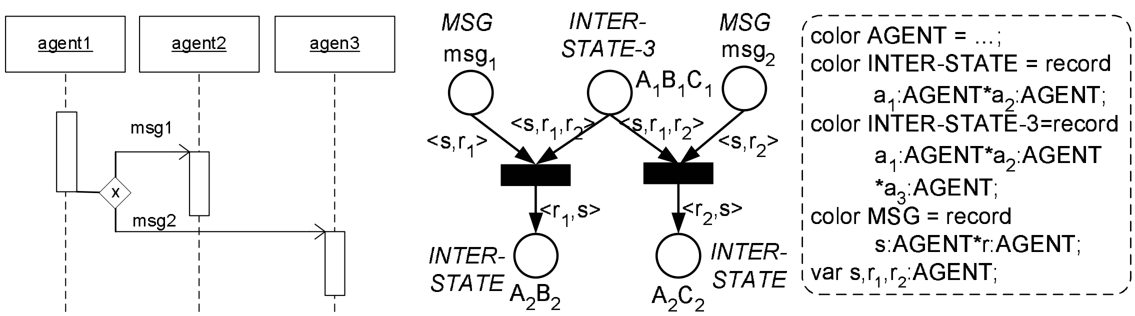


(a) AUML representation (b) CP-net representation

Figure 2: Synchronous message interaction.

Transitions in a typical multi-agent interaction protocols are composed of interaction building blocks, two of which have been presented above. Additional interaction building-blocks, which are fairly straightforward (or have appeared in previous work, e.g., Poutakidis et al., 2002) are presented in Appendix B. In the remainder of this section, we present two complex interactions building blocks that are generally common in multi-agent interactions: XOR-decision and OR-parallel.

We begin with the XOR-decision interaction. The AUML representation to this building block is shown in Figure 3-a. The sender agent $agent_1$ can either send message msg_1 to $agent_2$ or message msg_2 to $agent_3$, but it can not send both msg_1 and msg_2. The non-filled diamond with an 'x' inside is the AUML notation for this constraint.



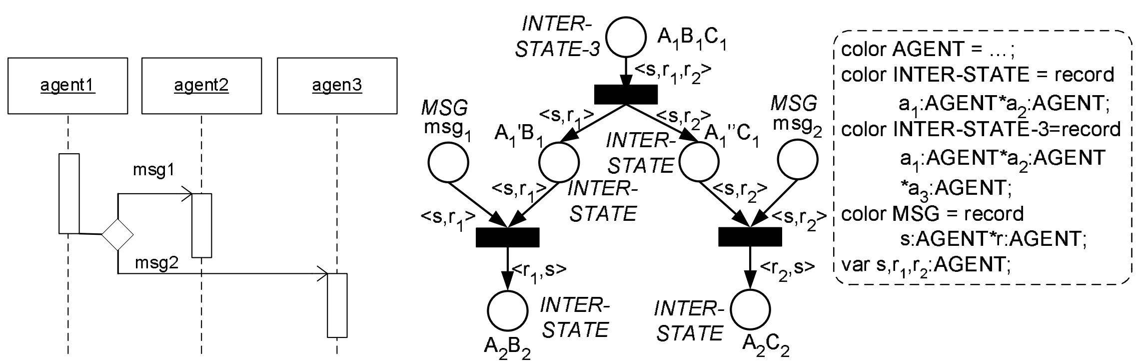
(a) AUML representation (b) CP-net representation

Figure 3: XOR-decision messages interaction.

Figure 3-b shows the corresponding CP-net. Again, the A, B and C capital letters are used to denote the interaction states of $agent_1$, $agent_2$ and $agent_3$, respectively. The

interaction starts from the $A_1B_1C_1$ place and terminates either in the A_2B_2 place or in the A_2C_2 place. The $A_1B_1C_1$ place represents a joint interaction state where $agent_1$ is ready to send either the msg_1 communicative act to $agent_2$ or the msg_2 communicative act to $agent_3$ (A_1); and $agent_2$ and $agent_3$ are waiting to receive the corresponding msg_1/msg_2 message (B_1/C_1). To represent the $A_1B_1C_1$ place color set, we extend the *INTER-STATE* color set to denote a joint interaction state of three interacting agents, i.e. using the *INTER-STATE-3* color set. The msg_1 communicative act causes the agents to transition to A_2B_2 place. The A_2B_2 place represents a joint interaction state where $agent_1$ has sent the msg_1 message (A_2), and $agent_2$ has received it (B_2). Similarly, the msg_2 communicative act causes agents $agent_1$ and $agent_3$ to transition to A_2C_2 place. Exclusiveness is achieved since the single agent token in $A_1B_1C_1$ place can be used either for activating the $A_1B_1C_1 \rightarrow A_2B_2$ transition or for activating the $A_1B_1C_1 \rightarrow A_2C_2$ transition, but not both.

A similar complex interaction is the OR-parallel messages interaction. Its AUML representation is presented in Figure 4-a. The sender agent, $agent_1$, can send message msg_1 to $agent_2$ or message msg_2 to $agent_3$, or both. The non-filled diamond is the AUML notation for this constraint.



(a) AUML representation (b) CP-net representation

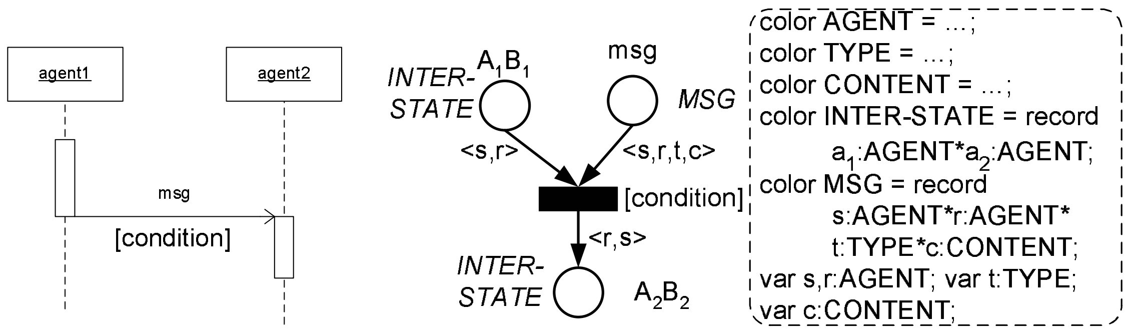
Figure 4: OR-parallel messages interaction.

Figure 4-b shows the CP-net representation of the OR-parallel interaction. The interaction starts from the $A_1B_1C_1$ place but it can be terminated in the A_2B_2 place, or in the A_2C_2 place, or in both. To represent this inclusiveness of the interaction protocol, we define two intermediate places, the A'_1B_1 place and the A''_1C_1 place. The A'_1B_1 place represents a joint interaction state where $agent_1$ is ready to send the msg_1 communicative act to $agent_2$ (A'_1) and $agent_2$ is waiting to receive the message (B_1). The A''_1C_1 place has similar meaning, but with respect to $agent_3$. As normally done in Petri nets, the transition connecting the $A_1B_1C_1$ place to the intermediate places duplicates any single token in $A_1B_1C_1$ place into two tokens going into the A'_1B_1 and the A''_1C_1 places. Consequently, the two parts of the OR-parallel interaction can be independently executed.

4. Representing Interaction Attributes

We now extend our representation to allow additional interaction aspects, useful in describing multi-agent conversation protocols. First, we show how to represent interaction message attributes, such as guards, sequence expressions, cardinalities and content (FIPA Specifications, 2003c). We then explore in depth the representation of multiple concurrent conversations (on the same CP net).

Figure 5-a shows a simple agent interaction using an AUML protocol diagram. This interaction is similar to the one presented in Figure 1-a in the previous section. However, Figure 5-a uses an AUML message guard-condition–marked as [*condition*]–that has the following semantics: the communicative act is sent from $agent_1$ to $agent_2$ if and only if the *condition* is true.



(a) AUML representation

(b) CP-net representation

Figure 5: Message guard-condition

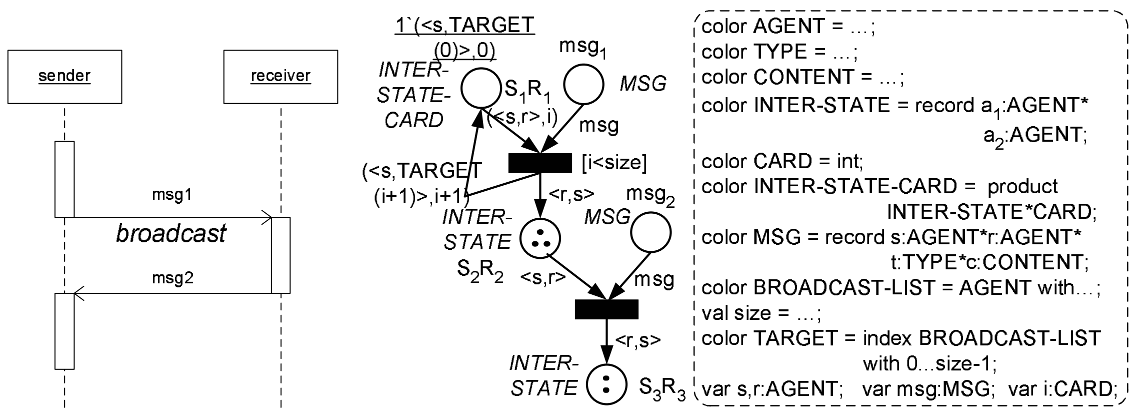
The guard-condition implementation in our Petri net representation uses transition guards (Figure 5-b), a native feature for CP nets. The AUML guard condition is mapped directly to the CP-net transition guard. The CP-net transition guard is indicated on the net inscription next to the corresponding transition using square brackets. The transition guard guarantees that the transition is enabled if and only if the transition guard is true.

In Figure 5-b, we also extend the color of tokens to include information about the communicative act being used and its content. We extend the MSG color set definition to a record $\langle s, r, t, c\rangle$, where the s and r elements has the same interpretation as in previous section (sender and receiver), and the t and c elements define the message type and content, respectively. The t element is of a new color $TYPE$, which determines communicative act types. The c element is of a new color $CONTENT$, which represents communicative act content and argument list (e.g. reply-to, reply-by and etc).

The addition of new elements also allows for additional potential uses. For instance, to facilitate representation of multiple concurrent conversations between the same agents (s and r), it is possible to add a conversation identification field to both the MSG and *INTER-STATE* colors. For simplicity, we refrain from doing so in the examples in this paper.

Two additional AUML communicative act attributes that can be modelled in the CP representation are message sequence-expression and message cardinality. The sequence-expressions denote a constraint on the message sent from sender agent. There are a number of sequence-expressions defined by FIPA conversation standards (FIPA Specifications, 2003c): m denotes that the message is sent exactly m times; $n..m$ denotes that the message is sent anywhere from n up to m times; $*$ denotes that the message is sent an arbitrary number of times. An additional important sequence expression is *broadcast*, i.e. message is sent to all other agents.

We now explain the representation of sequence-expressions in CP-nets, using broadcast as an example (Figure 6-b). Other sequence expressions are easily derived from this example. We define an *INTER-STATE-CARD* color set. This color set is a tuple $(\langle a_1, a_2 \rangle, i)$ consisting of two elements. The first tuple element is an *INTER-STATE* color element, which denotes the interacting agents as previously defined. The second tuple element is an integer that counts the number of messages already sent by an agent, i.e. the message cardinality. This element is initially assigned to 0. The *INTER-STATE-CARD* color set is applied to the S_1R_1 place, where the S and R capital letters are used to denote the *sender* and the *receiver* individual interaction states respectively and the S_1R_1 indicates the initial joint interaction state of the interacting agents. The two additional colors, used in Figure 6-b, are the *BROADCAST-LIST* and the *TARGET* colors. The *BROADCAST-LIST* color defines the sender broadcast list of the designated receivers, assuming that the *sender* must have such a list to carry out its role. The *TARGET* color defines indexes into this broadcast list.



(a) AUML representation (b) CP-net representation

Figure 6: Broadcast sequence expression.

According to the broadcast sequence-expression semantics, the *sender* agent sends the same msg_1 communicative act to all the *receivers* on the *broadcast list*. The CP-net introduced in Figure 6-b models this behavior.[4] The interaction starts from the S_1R_1 place, representing the joint interaction state where *sender* is ready to send the msg_1 communicative act to *receiver* (S_1) and *receiver* is waiting to receive the corresponding msg_1 message (R_1). The S_1R_1 place initial marking is a single token, set by the initialization expression (underlined, next to the corresponding place). The initialization expression $1`(\langle s, TARGET(0)\rangle, 0)$–given in standard CPN ML notation–determines that the S_1R_1 place's initial marking is a multi-set containing a single token $(\langle s, TARGET(0)\rangle, 0)$. Thus, the first designated *receiver* is assigned to be the agent with index 0 on the broadcast list, and the message cardinality counter is initiated to 0.

4. We implement broadcast as an iterative procedure sending the corresponding communicative act separately to all designated recipients.

The msg_1 message place initially contains multiple tokens. Each of these tokens represents the msg_1 communicative act addressed to a different designated *receiver* on the *broadcast list*. In Figure 6-b, the initialization expression, corresponding to the msg_1 message place, has been omitted. The S_1R_1 place token and the appropriate msg_1 place token together enable the corresponding transition. Consequently, the transition may fire and thus the msg_1 communicative act transmission is simulated.

The msg_1 communicative act is sent incrementally to every designated *receiver* on the *broadcast list*. The incoming arc expression $(\langle s, r\rangle, i)$ is incremented by the transition to the outgoing $(\langle s, TARGET(i+1)\rangle, i+1)$ arc expression, causing the *receiver* agent with index $i+1$ on the *broadcast list* to be selected. The transition guard constraint $i < size$, i.e. $i < |broadcast\ list|$, ensures that the msg_1 message is sent no more than $|broadcast\ list|$ times. The msg_1 communicative act causes the agents to transition to the S_2R_2 place. This place represents a joint interaction state in which sender has already sent the msg_1 communicative act to *receiver* and is now waiting to receive the msg_2 message (S_2) and *receiver* has received the msg_1 message and is ready to send the msg_2 communicative act to sender (R_2). Finally, the msg_2 message causes the agents to transition to the S_3R_3 place. The S_3R_3 place denotes a joint interaction state where sender has received the msg_2 communicative act from *receiver* and terminated (S_3), while *receiver* has already sent the msg_2 message to *sender* and terminated as well (R_3).

We use Figure 6-b to demonstrate the use of token color to represent multiple concurrent conversations using the same CP-net. For instance, let us assume that the *sender* agent is called $agent_1$ and its *broadcast list* contains the following agents: $agent_2$, $agent_3$, $agent_4$, $agent_5$ and $agent_6$. We will also assume that the $agent_1$ has already sent the msg_1 communicative act to all agents on the *broadcast list*. However, it has only received the msg_2 reply message from $agent_3$ and $agent_6$. Thus, the CP-net current marking for the complete interaction protocol is described as follows: the S_2R_2 place is marked by $\langle agent_2, agent_1\rangle$, $\langle agent_4, agent_1\rangle$, $\langle agent_5, agent_1\rangle$, while the S_3R_3 place contains the tokens $\langle agent_1, agent_3\rangle$ and $\langle agent_1, agent_6\rangle$.

An Example. We now construct a CP-net representation of the *FIPA Query Interaction Protocol* (FIPA Specifications, 2003d), shown in AUML form in Figure 7, to demonstrate how the building blocks presented in Sections 3 and 4 can be put together. In this interaction protocol, the *Initiator* requests the *Participant* to perform an inform action using one of two query communicative acts, *query-if* or *query-ref*. The *Participant* processes the query and makes a decision whether to *accept* or $refuse$ the query request. The *Initiator* may request the *Participant* to respond with either an *accept* or $refuse$ message, and for simplicity, we will assume that this is always the case. In case the *query* request has been accepted, the *Participant* informs the *Initiator* on the query results. If the *Participant* fails, then it communicates a $failure$. In a successful response, the *Participant* replies with one of two versions of inform (*inform-t/f* or *inform-result*) depending on the type of initial query request.

The CP-net representation of the *FIPA Query Interaction Protocol* is presented in Figure 8. The interaction starts in the I_1P_1 place (we use the I and the P capital letters to denote the *Initiator* and the *Participant* roles). The I_1P_1 place represents a joint interaction state where (i) the *Initiator* agent is ready to send either the *query-if* communicative act, or the *query-ref* message, to *Participant* (I_1); and (ii) *Participant* is wait-

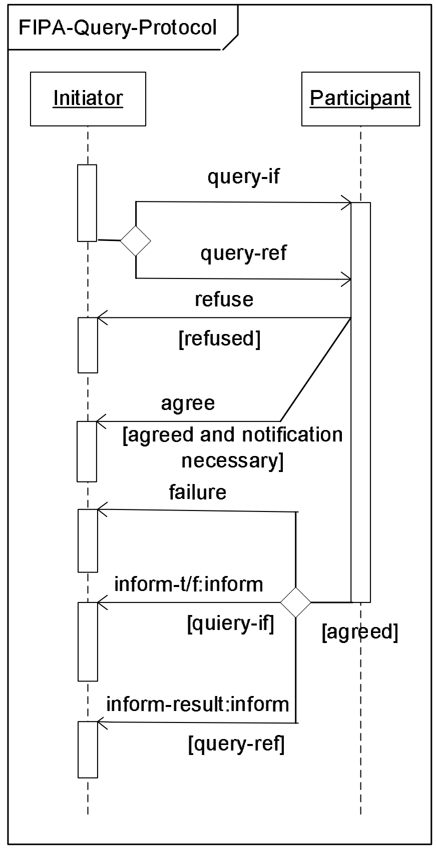


Figure 7: FIPA Query Interaction Protocol - AUML representation.

ing to receive the corresponding message (P_1). The *Initiator* can send either a *query-if* or a *query-ref* communicative act. We assume that these acts belong to the same class, the *query* communicative act class. Thus, we implement both messages using a single *Query* message place, and check the message type using the following transition guard: [*#t msg = query-if* or *#t msg = query-ref*]. The query communicative act causes the interacting agents to transition to the I_2P_2 place. This place represents a joint interaction state in which *Initiator* has sent the *query* communicative act and is waiting to receive a response message (I_2), and *Participant* has received the *query* communicative act and deciding whether to send an *agree* or a *refuse* response message to *Initiator* (P_2). The *refuse* communicative act causes the agents to transition to I_3P_3 place, while the *agree* message causes the agents to transition to I_4P_4 place.

The *Participant* decision on whether to send an *agree* or a *refuse* communicative act is represented using the XOR-decision building block introduced earlier (Figure 3-b). The I_3P_3 place represents a joint interaction state where *Initiator* has received a *refuse* communicative act and terminated (I_3) and *Participant* has sent a *refuse* message and terminated as well (P_3). The I_4P_4 place represents a joint interaction state in which *Initiator* has received an *agree* communicative act and is now waiting for further response from

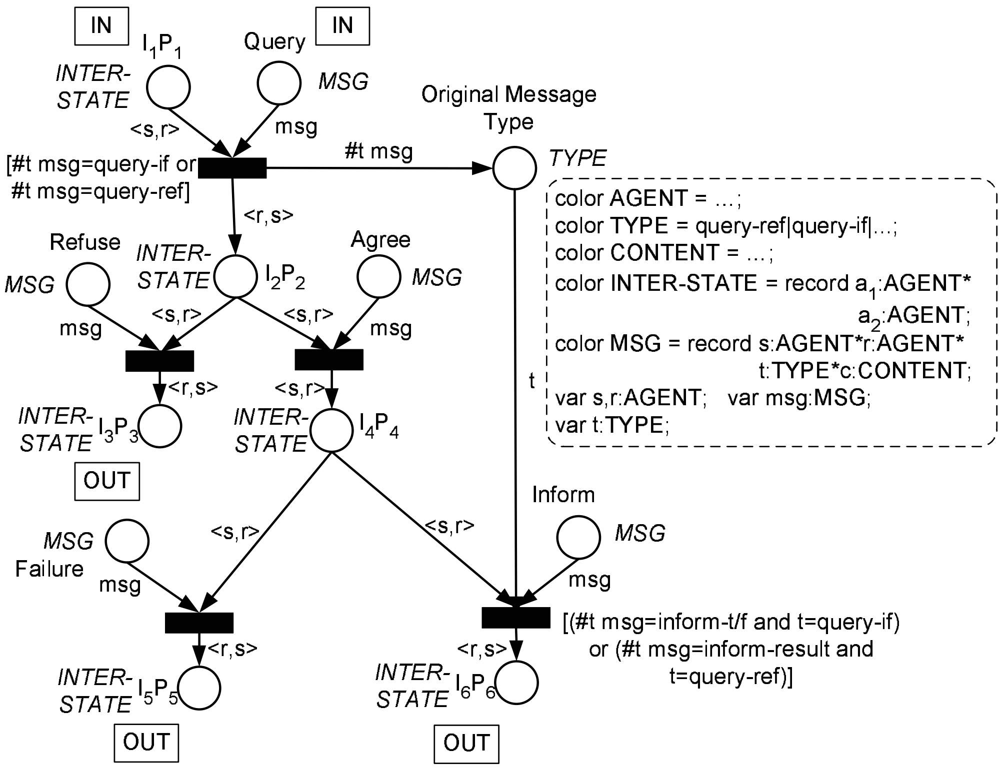


Figure 8: FIPA Query Interaction Protocol - CP-net representation.

Participant (I_4) and *Participant* has sent an *agree* message and is now deciding which response to send to *Initiator* (P_4). At this point, the *Participant* agent may send one of the following communicative acts: *inform-t/f*, *inform-result* and *failure*. The choice is represented using another XOR-decision building block, where the *inform-t/f* and *inform-result* communicative acts are represented using a single *Inform* message place. The *failure* communicative act causes a transition to the I_5P_5 place, while the *inform* message causes a transition to the I_6P_6 place. The I_5P_5 place represents a joint interaction state where *Participant* has sent a *failure* message and terminated (P_5), while *Initiator* has received a *failure* and terminated (I_5). The I_6P_6 place represents a joint interaction state in which *Participant* has sent an *inform* message and terminated (P_6), while *Initiator* has received an *inform* and terminated (I_6).

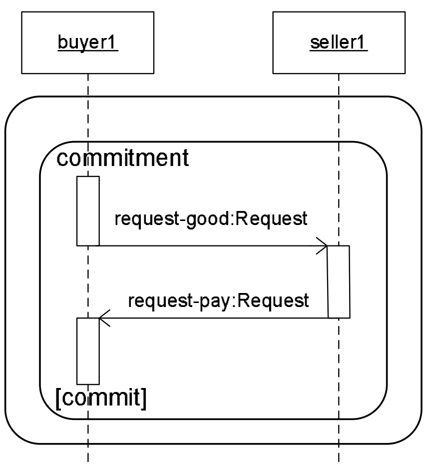
The implementation of the [*query-if*] and the [*query-ref*] message guard conditions requires a detailed discussion. These are not implemented in a usual manner in view of the fact that they depend on the original request communicative act. Thus, we create a special intermediate place that contains the original message type marked "*Original Message Type*" in the figure. In case an *inform* communicative act is sent, the transition guard verifies that the *inform* message is appropriate to the original *query* type. Thus, an *inform-t/f* communicative act can be sent only if the original query type has been *query-if* and an *inform-result* message can be sent only if the original query type has been *query-ref*.

5. Representing Nested & Interleaved Interactions

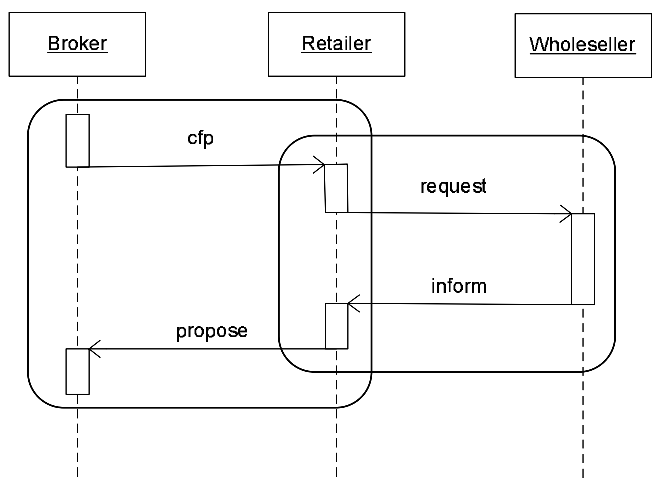
In this section, we extend the CP-net representation of previous sections to model nested and interleaved interaction protocols. We focus here on nested interaction protocols. Nevertheless, the discussion can also be addressed to interleaved interaction protocols in a similar fashion.

FIPA conversation standards (FIPA Specifications, 2003c) emphasize the importance of nested and interleaved protocols in modelling complex interactions. First, this allows re-use of interaction protocols in different nested interactions. Second, nesting increases the readability of interaction protocols.

The AUML notation annotates nested and interleaved protocols as round corner rectangles (Odell et al., 2001a; FIPA Specifications, 2003c). Figure 9-a shows an example of a nested protocol[5], while Figure 9-b illustrates an interleaved protocol. Nested protocols have one or more compartments. The first compartment is the name compartment. The name compartment holds the (optional) name of the nested protocol. The nested protocol name is written in the upper left-hand corner of the rectangle, i.e. *commitment* in Figure 9-a. The second compartment, the guard compartment, holds the (optional) nested protocol guard. The guard compartment is written in the lower left-hand corner of the rectangle, e.g. [*commit*] in Figure 9-a. Nested protocols without guards are equivalent to nested protocols with the [*true*] guard.



(a) Nested protocol



(b) Interleave protocol

Figure 9: AUML nested and interleaved protocols examples.

Figure 10 describes the implementation of the nested interaction protocol presented in Figure 9-a by extending the CP-net representation to using hierarchies, relying on standard CP-net methods (see Appendix A). The hierarchical CP-net representation contains three elements: a *superpage*, a *subpage* and a *page hierarchy* graph. The CP-net superpage represents the main interaction protocol containing a nested interaction, while the CP-net subpage models the corresponding nested interaction protocol, i.e. the *Commitment Inter-*

5. Figure 9-a appears in FIPA conversation standards (FIPA Specifications, 2003c). Nonetheless, note that the *request-good* and the *request-pay* communicative acts are not part of the FIPA-ACL standards.

action Protocol. The page hierarchy graph describes how the superpage is decomposed into subpages.

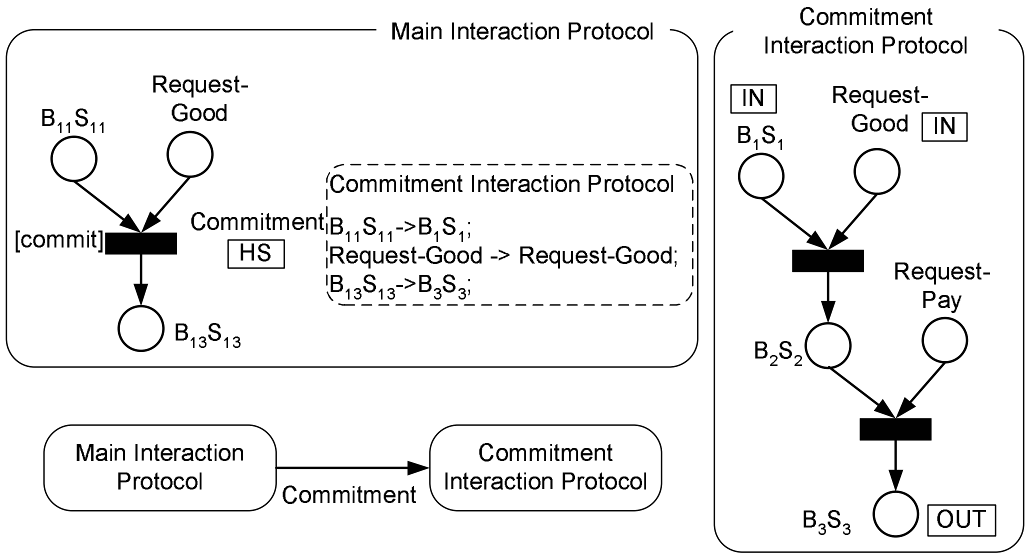


Figure 10: Nested protocol implementation using hierarchical CP-nets.

Let us consider in detail the process of modelling the nested interaction protocol in Figure 9-a using a hierarchical CP-net, resulting in the net described in Figure 10. First, we identify the starting and ending points of the nested interaction protocol. The starting point of the nested interaction protocol is where $Buyer_1$ sends a *Request-Good* communicative act to $Seller_1$. The ending point is where $Buyer_1$ receives a *Request-Pay* communicative act from $Seller_1$. We model these nested protocol end-points as CP-net *socket nodes* on the superpage, i.e. $Main\ Interaction\ Protocol$: $B_{11}S_{11}$ and *Request-Good* are input socket nodes and $B_{13}S_{13}$ is an output socket node.

The nested interaction protocol, the $Commitment\ Interaction\ Protocol$, is represented using a separate CP-net, following the principles outlined in Sections 3 and 4. This net is a subpage of the main interaction protocol superpage. The nested interaction protocol starting and ending points on the subpage correspond to the net *port nodes*. The B_1S_1 and *Request-Good* places are the subpage input port nodes, while the B_3S_3 place is an output port node. These nodes are tagged with the IN/OUT *port type tags* correspondingly.

Then, a *substitution transition*, which is denoted using HS (Hierarchy and Substitution), connects the corresponding socket places on the superpage. The substitution transition conceals the nested interaction protocol implementation from the net superpage, i.e. the $Main\ Interaction\ Protocol$. The nested protocol name and guard compartments are mapped directly to the substitution transition name and guard respectively. Consequently, in Figure 10 we define the substitution transition name as *Commitment* and the substitution guard is determined to be [*commit*].

The superpage and subpage interface is provided using the hierarchy inscription. The hierarchy inscription is indicated using the dashed box next to the substitution transition. The first line in the hierarchy inscription determines the subpage identity, i.e. the

Commitment Interaction Protocol in our example. Moreover, it indicates that the substitution transition replaces the corresponding subpage detailed implementation on the superpage. The remaining hierarchy inscription lines introduce the superpage and subpage port assignment. The port assignment relates a socket node on the superpage with a port node on the subpage. The substitution transition input socket nodes are related to the IN-tagged port nodes. Analogously, the substitution transition output socket nodes correspond to the OUT-tagged port nodes. Therefore, the port assignment in Figure 10 assigns the net socket and port nodes in the following fashion: $B_{11}S_{11}$ to B_1S_1, *Request-Good* to *Request-Good* and $B_{13}S_{13}$ to B_3S_3.

Finally, the page hierarchy graph describes the decomposition hierarchy (nesting) of the different protocols (pages). The CP-net pages, the *Main Interaction Protocol* and the *Commitment Interaction Protocol*, correspond to the page hierarchy graph nodes (Figure 10). The arc inscription indicates the substitution transition, i.e. *Commitment*.

6. Representing Temporal Aspects of Interactions

Two temporal interaction aspects are specified by FIPA (FIPA Specifications, 2003c). In this section, we show how *timed* CP-nets (see also Appendix A) can be applied for modelling agent interactions that involve temporal aspects, such as interaction duration, deadlines for message exchange, etc.

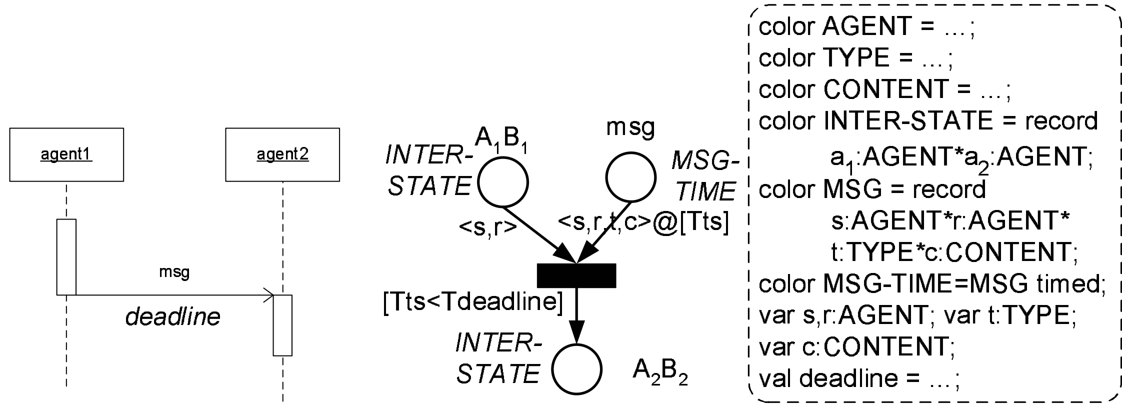
A first aspect, *duration*, is the interaction activity time period. Two periods can be distinguished: *transmission time* and *response time*. The transmission time indicates the time interval during which a communicative act, is sent by one agent and received by the designated receiver agent. The response time period denotes the time interval in which the corresponding receiver agent is performing some task as a response to the incoming communicative act.

The second temporal aspect is *deadlines*. Deadlines denote the time limit by which a communicative act must be sent. Otherwise, the corresponding communicative act is considered to be invalid. These issues have not been addressed in previous investigations related to agent interactions modelling using Petri nets.[6]

We propose to utilize timed CP-nets techniques to represent these temporal aspects of agent interactions. In doing so, we assume a global clock.[7] We begin with deadlines. Figure 11-a introduces the AUML representation of message deadlines. The *deadline* keyword is a variation of the communicative act sequence expressions described in Section 4. It sets a time constraint on the start of the transmission of the associated communicative act. In Figure 11-a, $agent_1$ must send the *msg* communicative act to $agent_2$ before the defined *deadline*. Once the *deadline* expires, the *msg* communicative act is considered to be invalid.

Figure 11-b shows a timed CP-net implementation of the deadline sequence expression. The timed CP-net in Figure 11-b defines an additional *MSG-TIME* color set associated with the net message places. The *MSG-TIME* color set extends the *MSG* color set, described in Section 4, by adding a time stamp attribute to the message token. Thus, the communicative

6. Cost et al. (1999, 2000) mention deadlines without presenting any implementation details.
7. Implementing it, we can use the private clock of an overhearing agent as the global clock for our Petri net representation. Thus, the time stamp of the message is the overhearer's time when the corresponding message was overheard.

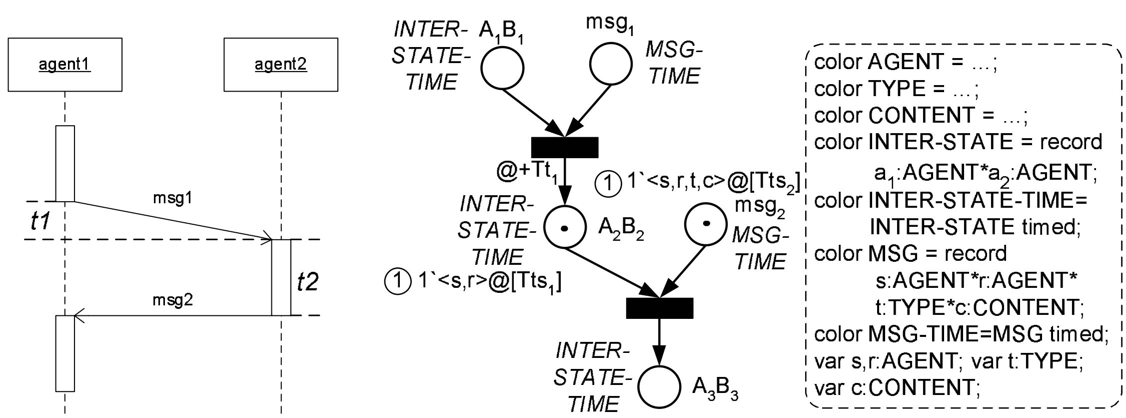


(a) AUML representation (b) CP-net representation

Figure 11: Deadline sequence expression.

act token is a record $\langle s, r, t, c\rangle@[Tts]$. The @[..] expression denotes the corresponding token time stamp, whereas the token time value is indicated starting with a capital 'T'. Accordingly, the described message token has a *ts* time stamp. The communicative act time limit is defined using the val *deadline* parameter. Therefore, the deadline sequence expression semantics is simulated using the following transition guard: $[Tts < Tdeadline]$. This transition guard, comparing the *msg* time stamp against the *deadline* parameter, guarantees that an expired *msg* communicative act can not be received.

We now turn to representing interaction duration. The AUML representation is shown in Figure 12-a. The AUML time intensive message notation is used to denote the communicative act transmission time. As a rule communicative act arrows are illustrated horizontally. This indicates that the message transmission time can be neglected. However, in case the message transmission time is significant, the communicative act is drawn slanted downwards. The vertical distance, between the arrowhead and the arrow tail, denotes the message transmission time. Thus, the communicative act msg_1, sent from $agent_1$ to $agent_2$, has a t_1 transmission time.



(a) AUML representation (b) CP-net representation

Figure 12: Interaction duration.

The response time in Figure 12-a is indicated through the interaction thread length. The incoming msg_1 communicative act causes $agent_2$ to perform some task before sending a response msg_2 message. The corresponding interaction thread duration is denoted through the t_2 time period. Thus, this time period specifies the $agent_2$ response time to the incoming msg_1 communicative act.

The CP-net implementation to the interaction duration time periods is shown in Figure 12-b. The communicative act transmission time is illustrated using the timed CP-nets @+ operator. The net transitions simulate the communicative act transmission between agents. Therefore, representing a transmission time of t_1, the CP-net transition adds a t_1 time period to the incoming message token time stamp. Accordingly, the transition $@+Tt_1$ output arc expression denotes a t_1 delay to the time stamp of the outgoing token. Thus, the corresponding transition takes t_1 time units and consequently so does the msg_1 communicative act transmission time.

In contrast to communicative act transmission time, the agent interaction response time is represented implicitly. Previously, we have defined a *MSG-TIME* color set that indicates message token time stamps. Analogously, in Figure 12-b we introduce an additional *INTER-STATE-TIME* color set. This color set is associated with the net agent places and it presents the possibility to attach time stamps to agent tokens as well. Now, let us assume that A_2B_2 and msg_2 places contain a single token each. The circled '1' next to the corresponding place, together with the multi-set inscription, indicates the place current marking. Thus, the agent and the message place tokens have a ts_1 and a ts_2 time stamps respectively. The ts_1 time stamp denotes the time by which $agent_2$ has received the msg_1 communicative act sent by $agent_1$. The ts_2 time stamp indicates the time by which $agent_2$ is ready to send msg_2 response message to $agent_1$. Thus, the $agent_2$ response time t_2 (Figure 12-a) is $ts_2 - ts_1$.

7. Algorithm and a Concluding Example

Our final contribution in this paper is a skeleton procedure for transforming an AUML conversation protocol diagram of two interacting agents to its CP-net representation. The procedure is semi-automated–it relies on the human to fill in some details–but also has automated aspects. We apply this procedure on a complex multi-agent conversation protocol that involves many of the interaction building blocks already discussed.

The procedure is shown in Algorithm 1. The algorithm input is an AUML protocol diagram and the algorithm creates, as an output, a corresponding CP-net representation. The CP-net is constructed in iterations using a queue. The algorithm essentially creates the conversation net by exploring the interaction protocol breadth-first while avoiding cycles.

Lines 1-2 create and initiate the algorithm queue, and the output CP-net, respectively. The queue, denoted by S, holds the initiating agent places of 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. The A_1B_1 place represents a joint initial interaction state for the two agents. Lines 7-23 contain the main loop.

We enter the main loop in line 8 and set the *curr* variable to the first initiating agent place in S queue. Lines 10-13 create the CP-net components corresponding to the current iteration as follows. First, in line 10, message places, associated with *curr* agent place, are

Algorithm 1 Create Conversation Net(**input**:$AUML$,**output**:CPN)

1: $S \leftarrow$ **new** queue
2: $CPN \leftarrow$ **new** CP $-$ net
3:
4: $A_1B_1 \leftarrow$ **new** agent place with color information
5: $S.enqueue(A_1B_1)$
6:
7: **while** S **not** *empty* **do**
8:     $curr \leftarrow S.dequeue()$
9:
10:     $MP \leftarrow CreateMessagePlaces(AUML, curr)$
11:     $RP \leftarrow CreateResultingAgentPlaces(AUML, curr, MP)$
12:     $(TR, AR) \leftarrow CreateTransitionsAndArcs(AUML, curr, MP, RP)$
13:     $FixColor(AUML, CPN, MP, RP, TR, AR)$
14:
15:     **for each** *place* p **in** RP **do**
16:         **if** p was not created in current iteration **then**
17:             **continue**
18:         **if** p **is not** terminating place **then**
19:             $S.enqueue(p)$
20:
21:     $CPN.places = CPN.places \bigcup MP \bigcup RP$
22:     $CP.transitions = CPN.transitions \bigcup TR$
23:     $CPN.arcs = CPN.arcs \bigcup AR$
24:
25: **return** CPN

created using the $CreateMessagePlaces$ procedure (which we do not detail here). This procedure extracts the communicative acts that are associated with a given interaction state, from the AUML diagram. These places correspond to communicative acts, which take agents from the joint interaction state *curr* to its successor(s). Then in line 11, the $CreateResultingAgentPlaces$ procedure creates agent places that correspond to interaction state changes as a result of the communicative acts associated with *curr* agent place (again based on the AUML diagram). Then, in $CreateTransitionsAndArcs$ procedure (line 12), these places are connected using the principles described in Sections 3–6. Thus, the CP-net structure (net places, transitions and arcs) is created. Finally, in line 13, the $FixColor$ procedure adds token color elements to the CP-net structure, to support deadlines, cardinality, and other communicative act attributes.

Lines 15-19 determine which resulting agent places are inserted into the S queue for further iteration. Only non-terminating agent places, i.e. places that do not correspond to interaction states that terminate the interaction, are inserted into the queue in lines 18-19. However, there is one exception (lines 16-17): a resulting agent place, which has already been handled by the algorithm, is not inserted back into the S queue since inserting it can cause an infinite loop. Thereafter, completing the current iteration, the output CP-net, denoted by CPN variable, is updated according to the current iteration CP-net components in lines 21-23. This main loop iterates as long as the S queue is not empty. The resulting CP-net is returned–line 25.

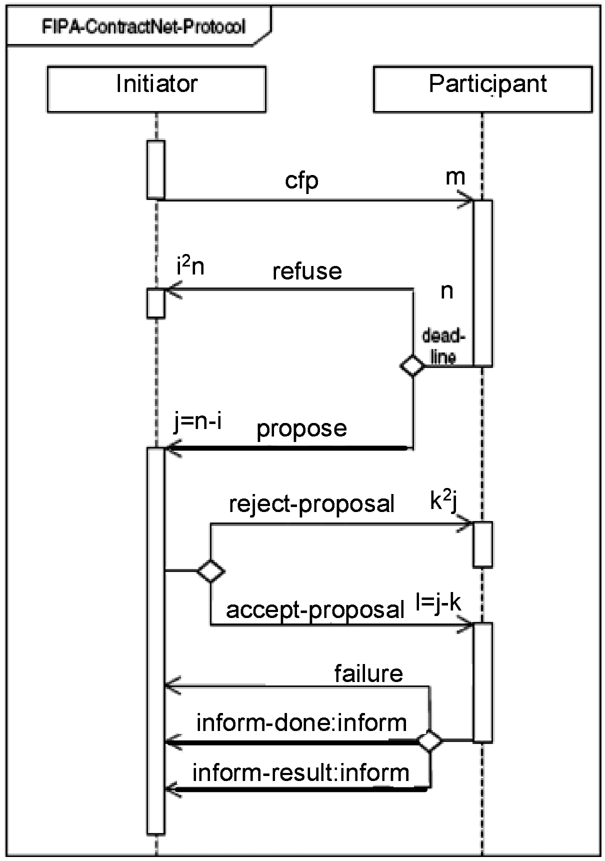


Figure 13: FIPA Contract Net Interaction Protocol using AUML.

To demonstrate this algorithm, we will now use it on the *FIPA Contract Net Interaction Protocol* (FIPA Specifications, 2003b) (Figure 13). This protocol allows interacting agents to negotiate. The $Initiator$ agent issues m calls for proposals using a cfp communicative act. Each of the m $Participants$ may refuse or counter-propose by a given *deadline* sending either a $refuse$ or a *propose* message respectively. A $refuse$ message terminates the interaction. In contrast, a *propose* message continues the corresponding interaction.

Once the *deadline* expires, the $Initiator$ does not accept any further $Participant$ response messages. It evaluates the received $Participant$ proposals and selects one, several, or no agents to perform the requested task. Accepted proposal result in the sending of *accept-proposal* messages, while the remaining proposals are rejected using *reject-proposal* message. *Reject-proposal* terminates the interaction with the corresponding $Participant$. On the other hand, the *accept-proposal* message commits a $Participant$ to perform the requested task. On successful completion, $Participant$ informs $Initiator$ sending either an *inform-done* or an *inform-result* communicative act. However, in case a $Participant$ has failed to accomplish the task, it communicates a $failure$ message.

We now use the algorithm introduced above to create a CP-net, which represents the *FIPA Contract Net Interaction Protocol*. The corresponding CP-net model is constructed in four iterations of the algorithm. Figure 14 shows the CP-net representation after the second iteration of the algorithm, while Figure 15 shows the CP-net representation after the fourth and final iteration.

The *Contract Net Interaction Protocol* starts from I_1P_1 place, which represents a joint interaction state where *Initiator* is ready to send a cfp communicative act (I_1) and *Participant* is waiting for the corresponding cfp message (P_1). The I_1P_1 place is created and inserted into the queue before the iterations through the main loop begin.

First iteration. The *curr* variable is set to the I_1P_1 place. 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. The I_2P_2 place denotes an interaction state in which *Initiator* has already sent a cfp communicative act to *Participant* and is now waiting for its response (I_2) and *Participant* has received the cfp message and is now deciding on an appropriate response (P_2). These are created using the *CreateMessagePlaces* and the *CreateResultingAgentPlaces* procedures, respectively.

Then, the *CreateTransitionsAndArcs* procedure in line 12, connects the three places using a simple asynchronous message building block as shown in Figure 1-b (Section 3). In line 13, as the color sets of the places are determined, the algorithm also handles the cardinality of the cfp communicative act, by putting an appropriate sequence expression on the transition, using the principles presented in Figure 6-b (Section 4). Accordingly, the color set, associated with I_1P_1 place, is changed to the *INTER-STATE-CARD* color set. Since the I_2P_2 place is not a terminating place, it is inserted into the S queue.

Second iteration. *curr* is set to the I_2P_2 place. The *Participant* agent can send, as a response, either a $refuse$ or a *propose* communicative act. $Refuse$ and *Propose* message places are created by *CreateMessagePlaces* (line 10), and resulting places I_3P_3 and I_4P_4, corresponding to the results of the $refuse$ and *propose* communicative acts, respectively, are created by *CreateResultingAgentPlaces* (line 11). The I_3P_3 place represents a joint interaction state where *Participant* has sent the $refuse$ message and terminated (P_3), while *Initiator* has received it, and terminated (I_3). The I_4P_4 place represents the joint state in which *Participant* has sent the *propose* message (P_4), while *Initiator* has received the message and is considering its response (I_4).

In line 12, the I_2P_2, $Refuse$, I_3P_3, *Propose* and I_4P_4 places are connected using the XOR-decision building block presented in Figure 3-b (Section 3). Then, the *FixColor* procedure (line 13), adds the appropriate token color attributes, to allow a deadline sequence expression (on both the $refuse$ and the *propose* messages) to be implemented as shown in Figure 11-b (Section 6). The I_3P_3 place denotes a terminating state, whereas the I_4P_4 place continues the interaction. Thus, in lines 18-19, only the I_4P_4 place is inserted into the queue, for the next iteration of the algorithm. The state of the net at the end of the second iteration of the algorithm is presented in Figure 14.

Third iteration. *curr* is set to I_4P_4. Here, the *Initiator* response to a *Participant* proposal can either be an *accept-proposal* or a *reject-proposal*. *CreateMessagePlaces* procedure in line 10 thus creates the corresponding *Accept-Proposal* and *Reject-Proposal* message places. The *accept-proposal* and *reject-proposal* messages cause the interacting agents to transition to I_5P_5 and I_6P_6 places, respectively. These agent places are created using the

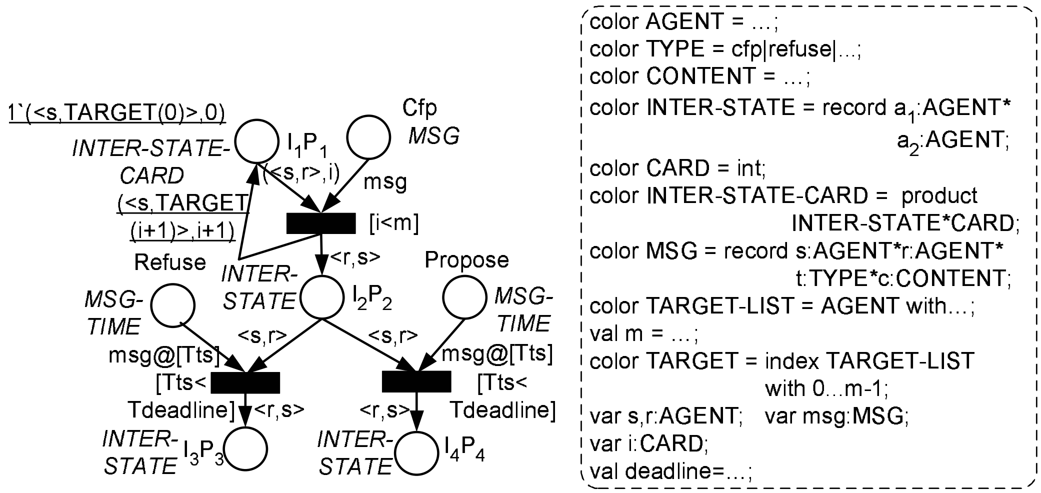


Figure 14: FIPA Contract Net Interaction Protocol using CP-net after the 2^{nd} iteration.

$CreateResultingAgentPlaces$ procedure (line 11). The I_5P_5 place denotes an interaction state in which $Initiator$ has sent a *reject-proposal* message and terminated the interaction (I_5), while the $Participant$ has received the message and terminated as well (P_5). In contrast, the I_6P_6 place represents an interaction state where $Initiator$ has sent an *accept-proposal* message and is waiting for a response (I_6), while $Participant$ has received the *accept-proposal* communicative act and is now performing the requested task before sending a response (P_6). The $Initiator$ agent sends exclusively either an *accept-proposal* or a *reject-proposal* message. Thus, the I_4P_4, *Reject-Proposal*, I_5P_5, *Accept-Proposal* and I_6P_6 places are connected using a XOR-decision block (in the $CreateTransitionsAndArcs$ procedure, line 12).

The $FixColor$ procedure in line 13 operates now as follows: According to the interaction protocol semantics, the $Initiator$ agent evaluates all the received $Participant$ proposals once the *deadline* passes. Only thereafter, the appropriate *reject-proposal* and *accept-proposal* communicative acts are sent. Thus, $FixColor$ assigns a *MSG-TIME* color set to the *Reject-Proposal* and the *Accept-Proposal* message places, and creates a [$Tts >= Tdeadline$] transition guard on the associated transitions. This transition guard guarantees that $Initiator$ cannot send any response until the *deadline* expires, and all valid $Participant$ responses have been received. The resulting I_5P_5 agent place denotes a terminating interaction state, whereas the I_6P_6 agent place continues the interaction. Thus, only I_6P_6 agent place is inserted into the S queue.

Fourth iteration. *curr* is set to I_6P_6. This place is associated with three communicative acts: *inform-done*, *inform-result* and $failure$. The *inform-done* and the *inform-result* messages are instances of the $inform$ communicative act class. Thus, *CreateMessagePlaces* (line 10) creates only two message places, $Inform$ and $Failure$. In line 11, $CreateResultingAgentPlaces$ creates the I_7P_7 and I_8P_8 agent places. The $failure$ communicative act causes interacting agents to transition to I_7P_7 agent place, while both $inform$ messages cause the agents to transition to I_8P_8 agent place. The I_7P_7 place represents a joint interaction state where $Participant$ has sent the $failure$ message and terminated (P_7), while $Initiator$ has received a $failure$ communicative act and terminated (I_7). On the other hand, the I_8P_8 place denotes an interaction state in which $Participant$ has sent the $inform$ message (either *inform-done* or *inform-result*) and terminated (P_8), while $Initiator$ has received an $inform$ communicative act and terminated (I_8). The $inform$ and $failure$ communicative acts are sent exclusively. Thus $CreateTransitionsAndArcs$ (line 12) connects the I_6P_6, $Failure$, I_7P_7, $Inform$ and I_8P_8 places using a XOR-decision building block. Then, $FixColor$ assigns a [$\#t$ *msg* = *inform-done* or $\#t$ *msg* = *inform-result*] transition guard on the transition associated with $Inform$ message place. Since both the I_7P_7 and the I_8P_8 agent places represent terminating interaction states, they are not inserted into the queue, which remains empty at the end of the current iteration. This signifies the end of the conversion. The complete conversation CP-net resulting after this iteration of the algorithm is shown in Figure 15.

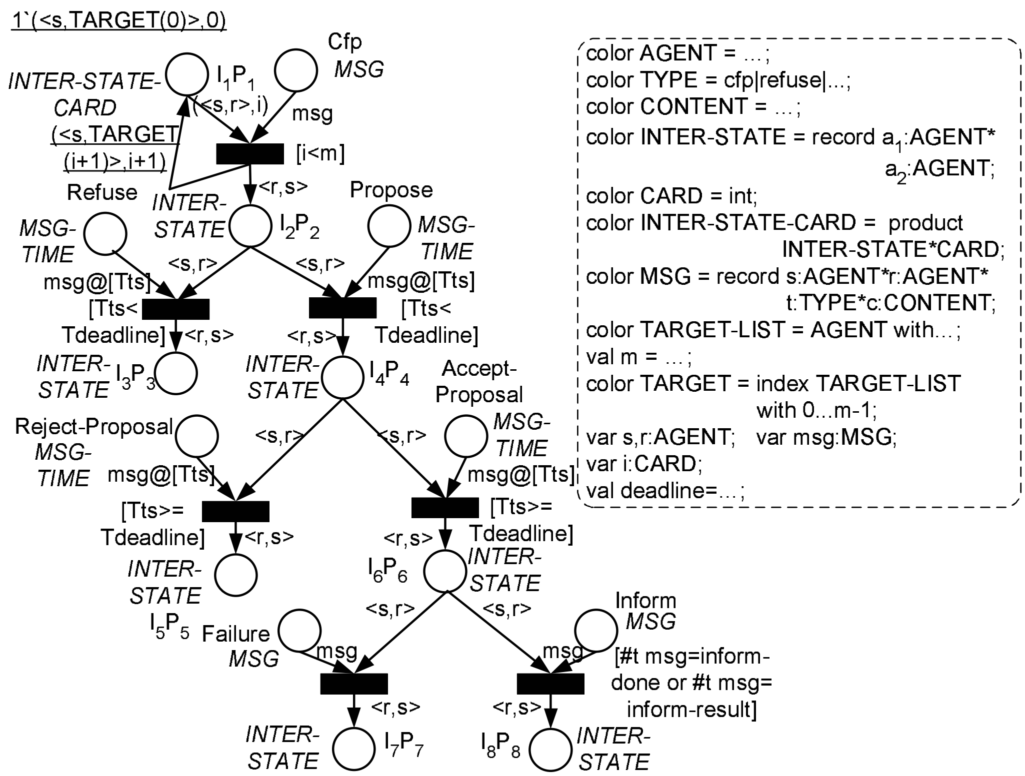


Figure 15: FIPA Contract Net Interaction Protocol using CP-net after the 4^{th} (and final) iteration.

The procedure we outline can guide the conversion of many *2-agent* conversation protocols in AUML to their CP-net equivalents. However, it is not sufficiently developed to address the general *n-agent* case. Appendix C presents a complex example of a *3-agent* conversation protocol, which was successfully converted manually, without the guidance of the algorithm. This example incorporates many advanced features of our CP-net representation technique and would have been beyond the scope of many previous investigations.

8. Summary and Conclusions

Over recent years, open distributed MAS applications have gained broad acceptance both in the multi-agent academic community and in real-world industry. As a result, increasing attention has been directed to multi-agent conversation representation techniques. In particular, Petri nets have recently been shown to provide a viable representation approach (Cost et al., 1999, 2000; Nowostawski et al., 2001; Mazouzi et al., 2002).

However, radically different approaches have been proposed to using Petri nets for modelling multi-agent conversations. Yet, the relative strengths and weaknesses of the proposed techniques have not been examined. Our work introduces a novel classification of previous investigations and then compares these investigations addressing their scalability and appropriateness for overhearing tasks.

Based on the insights gained from the analysis, we have developed a novel representation, that uses CP-nets in which places explicitly represent joint interaction states and messages. This representation technique offers significant improvements (compared to previous approaches) in terms of scalability, and is particularly suitable for monitoring via overhearing. We systematically show how this representation covers essentially all the features required to model complex multi-agent conversations, as defined by the FIPA conversation standards (FIPA Specifications, 2003c). These include simple & complex interaction building blocks (Section 3 & Appendix B), communicative act attributes and multiple concurrent conversations using the same CP-net (Section 4), nested & interleaved interactions using hierarchical CP-nets (Section 5) and temporal interaction attributes using timed CP-nets (Section 6). The developed techniques have been demonstrated, throughout the paper, on complex interaction protocols defined in the FIPA conversation standards (see in particular the example presented in Appendix C). Previous approaches could handle some of these examples (though with reduced scalability), but only a few were shown to cover all the required features.

Finally, the paper presented a skeleton procedure for semi-automatically converting an AUML protocol diagrams (the chosen FIPA representation standard) to an equivalent CP-net representation. We have demonstrated its use on a challenging FIPA conversation protocol, which was difficult to represent using previous approaches.

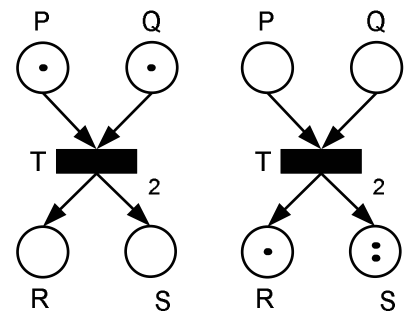
We believe that this work can assist and motivate continuing research on multi-agent conversations including such issues as performance analysis, validation and verification (Desel et al., 1997), agent conversation visualization, automated monitoring (Kaminka et al., 2002; Busetta et al., 2001, 2002), deadlock detection (Khomenco & Koutny, 2000), debugging (Poutakidis et al., 2002) and dynamic interpretation of interaction protocols (Cranefield et al., 2002; de Silva et al., 2003). Naturally, some issues remain open for future work. For example, the presented procedure addresses only AUML protocol diagrams representing two agent roles. We plan to investigate an *n-agent* version in the future.

Acknowledgments

The authors would like to thank the anonymous JAIR reviewers for many useful and informative comments. Minor subsets of this work were also published as LNAI book chapter (Gutnik & Kaminka, 2004b). K. Ushi deserves many thanks.

Appendix A. A Brief Introduction to Petri Nets

Petri nets (Petri Nets site, 2003) are a widespread, established methodology for representing and reasoning about distributed systems, combining a graphical representation with a comprehensive mathematical theory. One version of Petri nets is called Place/Transition nets (PT-nets) (Reisig, 1985). A PT-net is a bipartite directed graph where each node is either a place or a transition (Figure 16). The net places and transitions are indicated through circles and rectangles respectively. The PT-net arcs support only place $\rightarrow$ transition and transition $\rightarrow$ place connections, but never connections between two places or between two transitions. The arc direction determines the input/output characteristics of the place and the transition connected. Thus, given an arc, $P \rightarrow T$, connecting place P and transition T, we will say that place P is an input place of transition T and vice versa transition T is an output transition of place P. The $P \rightarrow T$ arc is considered to be an output arc of place P and an input arc of transition T.



(a) Before firing (b) After firing

Figure 16: A PT-net example.

A PT-net place may be *marked* by small black dots called *tokens*. The arc expression is an integer, which determines the number of tokens associated with the corresponding arc. By convention, an arc expression equal to 1 is omitted. A specific transition is *enabled* if and only if its input places *marking* satisfies the appropriate arc expressions. For example, consider arc $P \rightarrow T$ to be the only arc to connect place P and transition T. Thus, given that this arc has an arc expression 2, we will say that transition T is enabled if and only if place P is marked with two tokens. In case the transition is enabled, it may *fire/occur*. The transition occurrence removes tokens from the transition input places and puts tokens to the transition output places as specified by the arc expressions of the corresponding input/output arcs. Thus, in Figures 16-a and 16-b, we demonstrate PT-net marking before and after transition firing respectively.

Although computationally equivalent, a different version of Petri nets, called *Colored Petri nets* (CP-nets) (Jensen, 1997a, 1997b, 1997c), offers greater flexibility in compactly representing complex systems. Similarly to the PT-net model, CP-nets consist of net places, net transitions and arcs connecting them. However, in CP-nets, tokens are not just single bits, but can be complex, structured, information carriers. The type of additional information carried by the token, is called *token color*, and it may be simple (e.g., an integer or a string), or complex (e.g. a record or a tuple). Each place is declared by a *place color* set to only match tokens of particular colors. A CP-net place marking is a token *multi-set* (i.e., a set in which a member may appear more than once) corresponding to the appropriate place color set. CP-net arcs pass token multi-sets between the places and transitions. CP-net arc expressions can evaluate token multi-sets and may involve complex calculation procedures over token *variables* declared to be associated with the corresponding arcs.

The CP-net model introduces additional extensions to PT-nets. *Transition guards* are boolean expressions, which constrain transition firings. A transition guard associated with a transition tests tokens that pass through a transition, and will only enable the transition firings if the guard is successfully matched (i.e., the test evaluates to true). The CP-net transition guards, together with places color sets and arc expressions, appear as a part of net *inscriptions* in the CP-net.

In order to visualize and manage the complexity of large CP-nets, hierarchical CP-nets (Huber, Jensen, & Shapiro, 1991; Jensen, 1997a) allow hierarchical representations of CP-nets, in which sub-CP nets can be re-used in higher-level CP nets, or abstracted away from them. Hierarchical CP-nets are built from pages, which are themselves CP-nets. *Superpages* present a higher level of hierarchy, and are CP-nets that refer to *subpages*, in addition to transitions and places. A subpage may also function as a superpage to other subpages. This way, multiple hierarchy levels can be used in a hierarchical CP-net structure.

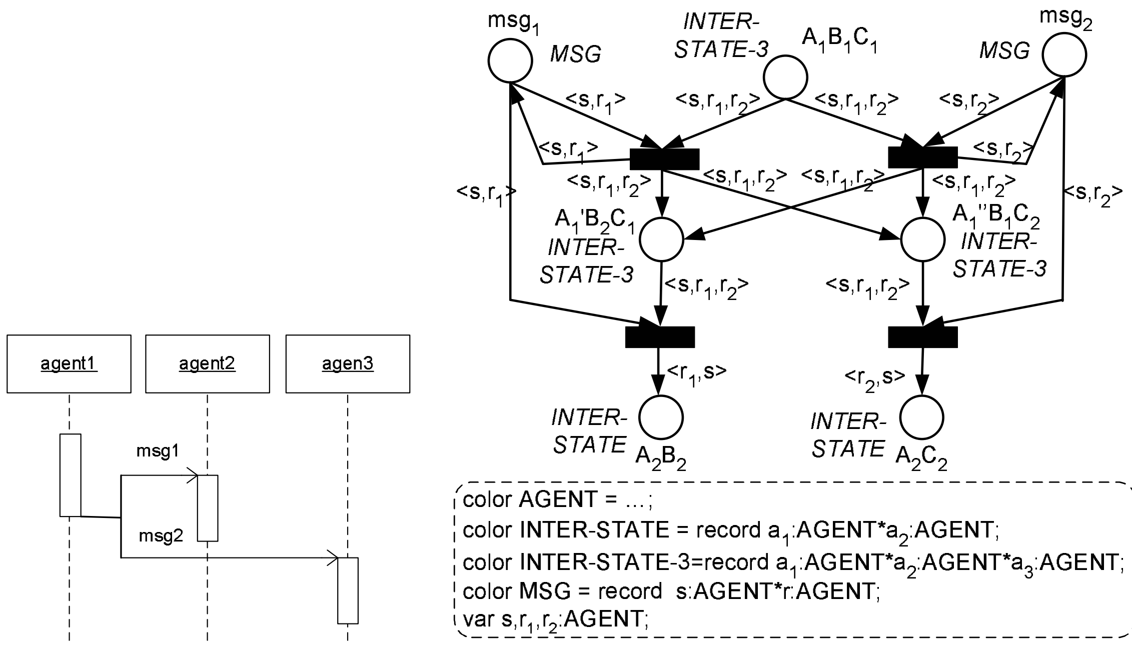
The relationship between a superpage and a subpage is defined by a *substitution transition*, which substitutes a corresponding *subpage instance* on the CP-net superpage structure as a transition in the superpage. The substitution transition *hierarchy inscription* supplies the exact mapping of the superpage places connected to the substitution transition (called *socket nodes*), to the subpage places (called *port nodes*). The *port types* determine the characteristics of the socket node to port node mappings. A complete CP-net hierarchical structure is presented using a *page hierarchy graph*, a directed graph where vertices correspond to pages, and directed edges correspond to direct superpage-subpage relationships.

Timed CP-nets (Jensen, 1997b) extend CP-nets to support the representation of temporal aspects using a *global clock*. Timed CP-net tokens have an additional color attribute called *time stamp*, which refers to the earliest time at which the token may be used. Time stamps can be used by arc expression and transition guards, to enable a timed-transition if and only if it satisfies two conditions: (i) the transition is *color enabled*, i.e. it satisfies the constraints defined by arc expression and transition guards; and (ii) the tokens are *ready*, i.e. the time of the global clock is equal to or greater than the tokens' time stamps. Only then can the transition fire.

Appendix B. Additional Examples of Conversation Representation Building Blocks

This appendix presents some additional interaction building blocks to those already described in Section 3. The first is the AND-parallel messages interaction (AUML representation shown in Figure 17-a). Here, the sender $agent_1$ sends both the msg_1 message to $agent_2$ and the msg_2 message to $agent_3$. However, the order of the two communicative acts is unconstrained.

The representation of AND-parallel in our CP-net representation is shown in Figure 17-b. The $A_1B_1C_1$, A_2B_2, A_2C_2, msg_1 and msg_2 places are defined similarly to Figures 3-b and



(a) AUML representation

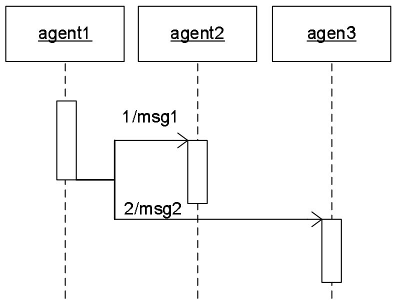
(b) CP-net representation

Figure 17: AND-parallel messages interaction.

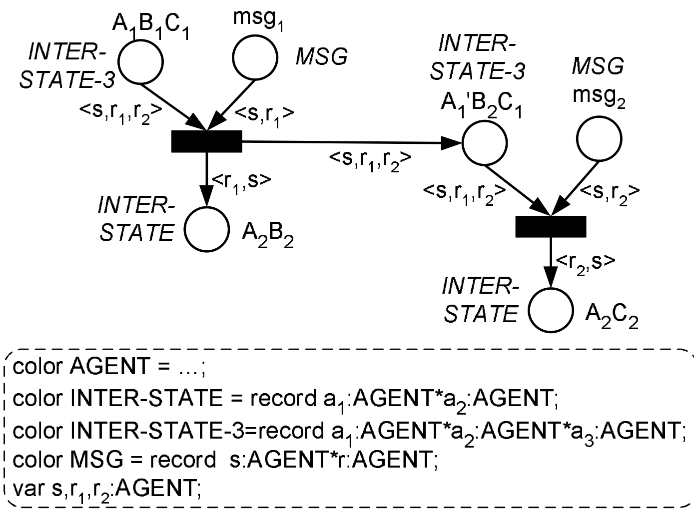
4-b in Section 3. However, we also define two additional intermediate agent places, $A_1'B_2C_1$ and $A_1''B_1C_2$. The $A_1'B_2C_1$ place represents a joint interaction state where $agent_1$ has sent the msg_1 message to $agent_2$ and is ready to send the msg_2 communicative act to $agent_3$ (A_1'), $agent_2$ has received the msg_1 message (B_2) and $agent_3$ is waiting to receive the msg_2 communicative act (C_1). The $A_1''B_1C_2$ place represents a joint interaction state in which $agent_1$ is ready to send the msg_1 message to $agent_2$ and has already sent the msg_2 communicative act to $agent_3$ (A_1''), $agent_2$ is waiting to receive the msg_1 message (B_1) and $agent_3$ has received the msg_2 communicative act (C_2). These places enable $agent_1$ to send both communicative acts concurrently. Four transitions connect the appropriate places respectively. The behavior of the transitions connecting $A_1'B_2C_1 \rightarrow A_2B_2$ and $A_1''B_1C_2 \rightarrow A_2C_2$ is similar to described above. The transitions $A_1B_1C_1 \rightarrow A_1'B_2C_1$ and $A_1B_1C_1 \rightarrow A_1''B_1C_2$ are triggered by receiving messages msg_1 and msg_2, respectively. However, these transitions should not consume the message token since it is used further for triggering transitions $A_1'B_2C_1 \rightarrow A_2B_2$ and $A_1''B_1C_2 \rightarrow A_2C_2$. This is achieved by adding an appropriate message place as an output place of the corresponding transition.

The second AUML interaction building block, shown in Figure 18-a, is the message sequence interaction, which is similar to AND-parallel. However, the message sequence interaction defines explicitly the order between the transmitted messages. Using the $1/msg_1$ and $2/msg_2$ notation, Figure 18-a specifies that the msg_1 message should be sent before sending msg_2.

Figure 18-b shows the corresponding CP-net representation. The $A_1B_1C_1$, A_2B_2, A_2C_2, msg_1 and msg_2 places are defined as before. However, the CP-net implementation presents an additional intermediate agent place–$A_1'B_2C1$–which is identical to the corresponding



(a) AUML representation

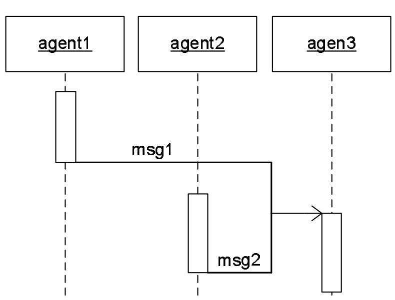


(b) CP-net representation

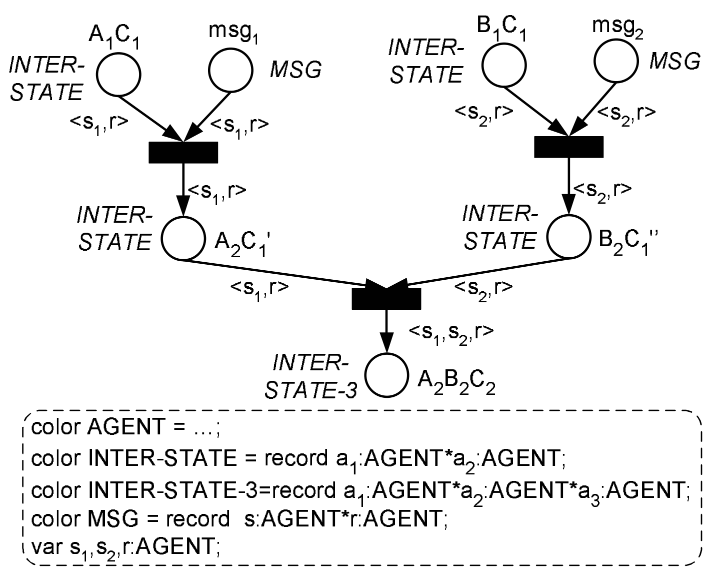
Figure 18: Sequence messages interaction.

intermediate agent place in Figure 17-b. $A_1'B_2C_1$ is defined as an output place of the $A_1B_1C_1 \rightarrow A_2B_2$ transition. It thus guarantees that the msg_2 communicative act can be sent (represented by the $A_1'B_2C_1 \rightarrow A_2C_2$ transition) only upon completion of the msg_1 transmission (the $A_1B_1C_1 \rightarrow A_2B_2$ transition).

The last interaction we present is the synchronized messages interaction, shown in Figure 19-a. Here, $agent_3$ simultaneously receives msg_1 from $agent_1$ and msg_2 from $agent_2$. In AUML, this constraint is annotated by merging the two communicative act arrows into a horizontal bar with a single output arrow.



(a) AUML representation



(b) CP-net representation

Figure 19: Synchronized messages interaction.

Figure 19-b illustrates the CP-net implementation of synchronized messages interaction. As in previous examples, we define the A_1C_1, B_1C_1, msg_1, msg_2 and $A_2B_2C_2$ places. We additionally define two intermediate agent places, A_2C_1' and B_2C_1''. The A_2C_1' place represents a joint interaction state where $agent_1$ has sent msg_1 to $agent_3$ (A_2), and $agent_3$ has received it, however $agent_3$ is also waiting to receive msg_2 (C_1'). The B_2C_1'' place represents a joint interaction state in which $agent_2$ has sent msg_2 to $agent_3$ (B_2), and $agent_3$ has received it, however $agent_3$ is also waiting to receive msg_1 (C_1''). These places guarantee that the interaction does not transition to the $A_2B_2C_2$ state until both msg_1 and msg_2 have been received by $agent_3$.

Appendix C. An Example of a Complex Interaction Protocol

We present an example of a complex *3-agent* conversation protocol, which was manually converted to a CP-net representation using the building blocks in this paper. The conversation protocol addressed here is the *FIPA Brokering Interaction Protocol* (FIPA Specifications, 2003a). This interaction protocol incorporates many advanced conversation features of our representation such as nesting, communicative act sequence expression, message guards and etc. Its AUML representation is shown in Figure 20.

The $Initiator$ agent begins the interaction by sending a *proxy* message to the $Broker$ agent. The *proxy* communicative act contains the requested *proxied-communicative-act* as part of its argument list. The $Broker$ agent processes the request and responds with either an *agree* or a $refuse$ message. Communication of a $refuse$ message terminates the interaction. If the $Broker$ agent has agreed to function as a proxy, it then locates the agents matching the $Initiator$ request. If no such agent can be found, the $Broker$ agent communicates a *failure-no-match* message and the interaction terminates. Otherwise, the $Broker$ agent begins m interactions with the matching agents. For each such agent, the $Broker$ informs the $Initiator$, sending either an *inform-done-proxy* or a *failure-proxy* communicative act. The *failure-proxy* communicative act terminates the *sub-protocol* interaction with the matching agent in question. The *inform-done-proxy* message continues the interaction. As the *sub-protocol* progresses, the $Broker$ forwards the received responses to the $Initiator$ agent using the *reply-message-sub-protocol* communicative acts. However, there can be other failures that are not explicitly returned from the *sub-protocol* interaction (e.g., if the agent executing the *sub-protocol* has failed). In case the $Broker$ agent detects such a failure, it communicates a *failure-brokering* message, which terminates the *sub-protocol* interaction.

A CP-net representation of the *FIPA Brokering Interaction Protocol* is shown in Figure 21. The *Brokering Interaction Protocol* starts from I_1B_1 place. The I_1B_1 place represents a joint interaction state where $Initiator$ is ready to send a *proxy* communicative act (I_1) and $Broker$ is waiting to receive it (B_1). The *proxy* communicative act causes the interacting agents to transition to I_2B_2. This place denotes an interaction state in which $Initiator$ has already sent a *proxy* message to $Broker$ (I_2) and $Broker$ has received it (B_2). The $Broker$ agent can send, as a response, either a $refuse$ or an *agree* communicative act. This CP-net component is implemented using the XOR-decision building block presented in Section 3. The $refuse$ message causes the agents to transition to I_3B_3 place and thus terminate the interaction. This place corresponds to $Broker$ sending a $refuse$ message and terminating (B_3), while $Initiator$ receiving the message and terminating (I_3). On the

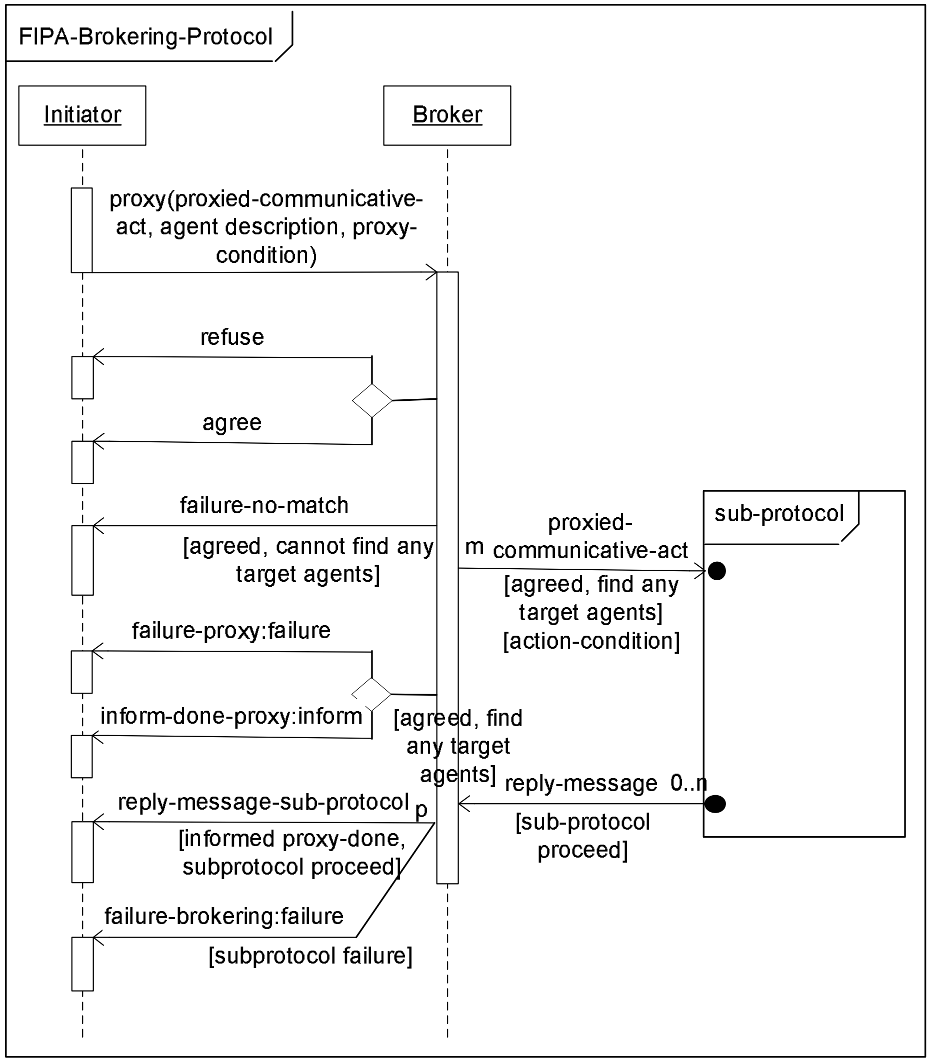


Figure 20: FIPA Brokering Interaction Protocol - AUML representation.

other hand, the *agree* communicative act causes the agents to transition to I_4B_4 place, which represents a joint interaction state in which the *Broker* has sent an *agree* message to *Initiator* (and is now trying to locate the receivers of the *proxied* message), while the *Initiator* received the *agree* message.

The *Broker* agent's search for suitable receivers may result in two alternatives. First, in case no matching agents are found, the interaction terminates in the I_5B_5 agent place. This joint interaction place corresponds to an interaction state where *Broker* has sent the *failure-no-match* communicative act (B_5), and *Initiator* has received the message and terminated (I_5). The second alternative is that suitable agents have been found. Then, *Broker* starts sending *proxied-communicative-act* messages to these agents on the established list of designated receivers, i.e. *TARGET-LIST*. The first such *proxied-communicative-act* message causes the interacting agents to transition to $I_4B_6P_1$ place. The $I_4B_6P_1$ place denotes a joint interaction state of three agents: *Initiator*, *Broker* and *Participant* (the receiver).

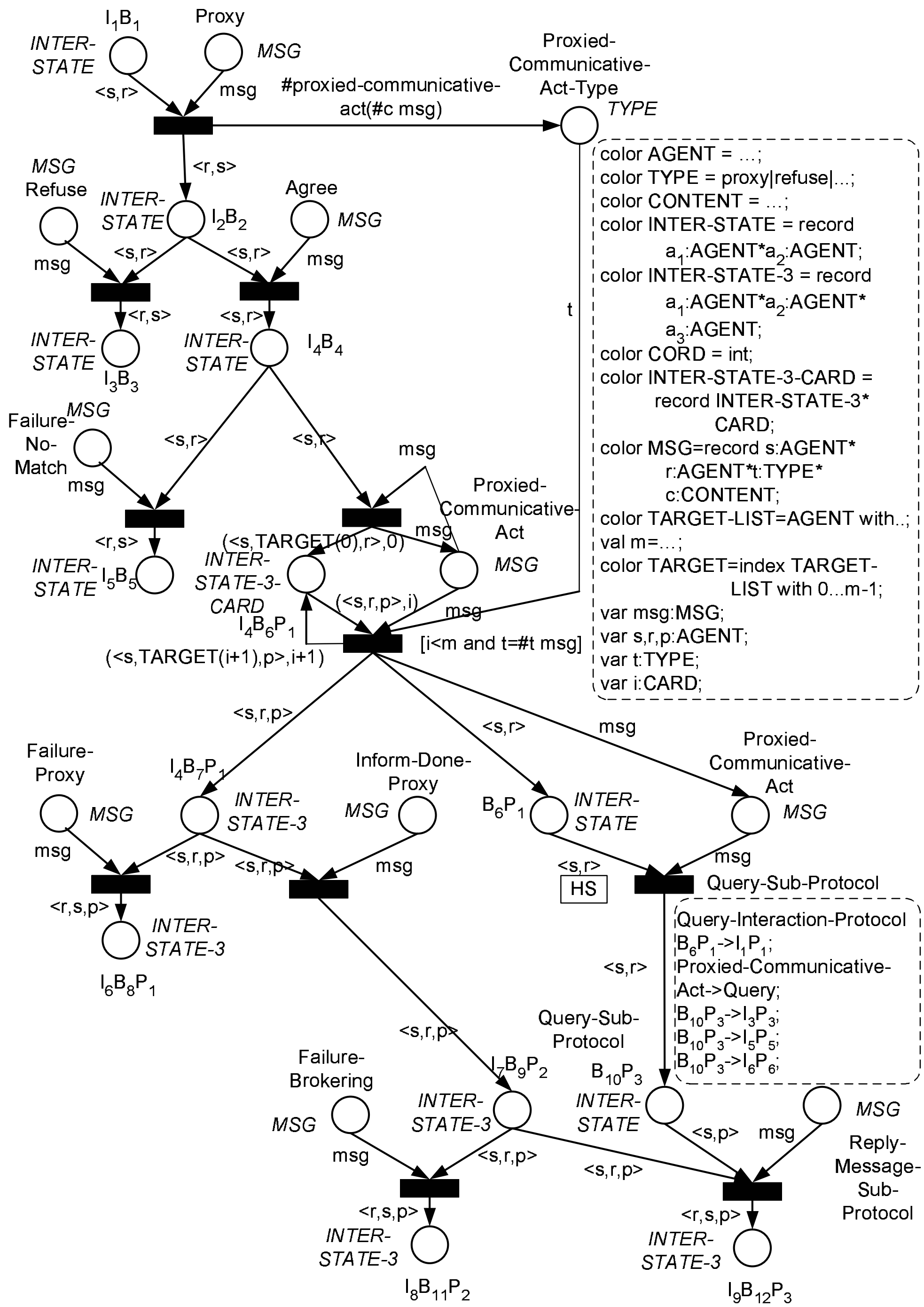


Figure 21: FIPA Brokering Interaction Protocol - CP-net representation.

The *Initiator* individual state remains unchanged (I_4) since the *proxied-communicative-act* message starts an interaction between *Broker* and *Participant*. The *Broker* individual state (B_6) denotes that designated agents have been found and the *proxied-communicative-*

act messages are ready to be sent, while *Participant* is waiting to receive the interaction initiating communicative act (P_1). The *proxied-communicative-act* message place is also connected as an output place of the transition. This message place is used as part of a CP-net XOR-decision structure, which enables the *Broker* agent to send either a *failure-no-match* or a *proxied-communicative-act*, respectively. Thus, the token denoting the *proxied-communicative-act* message, must not be consumed by the transition.

Thus, multiple *proxied-communicative-act* messages are sent to all *Participants*. This is implemented similarly to the broadcast sequence expression implementation (Section 4). Furthermore, the *proxied-communicative-act* type is verified against the type of the requested *proxied* communicative act, which is obtained from the original proxy message content. We use the *Proxied-Communicative-Act-Type* message type place to implement this CP-net component similarly to Figure 8. Each *proxied-communicative-act* message causes the interacting agents to transition to both the $I_4B_7P_1$ and the B_6P_1 places.

The B_6P_1 place corresponds to interaction between the *Broker* and the *Participant* agents. It represents a joint interaction state in which *Broker* is ready to send a *proxied-communicative-act* message to *Participant* (B_6), and *Participant* is waiting for the message (P_1). In fact, the B_6P_1 place initiates the nested interaction protocol that results in $B_{10}P_3$ place. The $B_{10}P_3$ place represents a joint interaction state where *Participant* has sent the *reply-message* communicative act and terminated (P_3), and *Broker* has received the message (B_{10}). In our example, we have chosen the *FIPA Query Interaction Protocol* (FIPA Specifications, 2003d) (Figures 7–8) as the interaction *sub-protocol*. The CP-net component, implementing the nested interaction *sub-protocol*, is modeled using the principles described in Section 5. Consequently, the interaction *sub-protocol* is concealed using the *Query-Sub-Protocol* substitution transition. The B_6P_1, *proxied-communicative-act* and $B_{10}P_3$ places determine substitution transition socket nodes. These socket nodes are assigned to the CP-net port nodes in Figure 8 as follows. The B_6P_1 and *proxied-communicative-act* places are assigned to the I_1P_1 and *query* input port nodes, while the $B_{10}P_3$ place is assigned to the I_3P_3, I_5P_5 and I_6P_6 output port nodes.

We now turn to the $I_4B_7P_1$ place. In contrast to the B_6P_1 place, this place corresponds to the main interaction protocol. The $I_4B_7P_1$ place represents a joint interaction state in which *Initiator* is waiting for *Broker* to respond (I_4), *Broker* is ready to send an appropriate response communicative act (B_7), and to the best of the *Initiator*'s knowledge the interaction with *Participant* has not yet begun (P_1). The *Broker* agent can send one of two messages, either a *failure-proxy* or an *inform-done-proxy*, depending on whether it has succeeded to send the *proxied-communicative-act* message to *Participant*. The *failure-proxy* message causes the agents to terminate the interaction with corresponding *Participant* agent and to transition to $I_6B_8P_1$ place. This place denotes a joint interaction state in which *Initiator* has received a *failure-proxy* communicative act and terminated (I_6), *Broker* has sent the *failure-proxy* message and terminated as well (B_8) and the interaction with the *Participant* agent has never started (P_1). On the other hand, the *inform-done-proxy* causes the agents to transition to $I_7B_9P_2$ place. The $I_7B_9P_2$ place represents an interaction state where *Broker* has sent the *inform-done-proxy* message (B_9), *Initiator* has received it (I_7), and *Participant* has begun the interaction with the *Broker* agent (P_2). Again, this is represented using the XOR-decision building block.

Finally, the *Broker* agent can either send a *reply-message-sub-protocol* or a *failure-brokering* communicative act. The *failure-brokering* message causes the interacting agents to transition to $I_8B_{11}P_2$ place. This place indicates that *Broker* has sent a *failure-brokering* message and terminated (B_{11}), *Initiator* has received the message and terminated (I_8), and *Participant* has terminated during the interaction with the *Broker* agent (P_2). The *reply-message-sub-protocol* communicative act causes the agents to transition to $I_9B_{12}P_3$ place. The $I_9B_{12}P_3$ place indicates that *Broker* has sent a *reply-message-sub-protocol* message and terminated (B_{12}), *Initiator* has received the message and terminated (I_9), and *Participant* has successfully completed the nested *sub-protocol* with the *Broker* agent and terminated as well (P_3). Thus, the $B_{10}P_3$ place, denoting a successful completion of the nested *sub-protocol*, is also the corresponding transition input place.

References

AUML site (2003). Agent unified modeling language, at *www.auml.org*..

Busetta, P., Dona, A., & Nori, M. (2002). Channelled multicast for group communications. In *Proceedings of AAMAS-02*.

Busetta, P., Serafini, L., Singh, D., & Zini, F. (2001). Extending multi-agent cooperation by overhearing. In *Proceedings of CoopIS-01*.

ChaibDraa, B. (2002). Trends in agent communication languages. *Computational Intelligence*, *18*(2), 89–101.

Cost, R. S. (1999). *A framework for developing conversational agents*. Ph.D. thesis, Department of Computer Science, University of Maryland.

Cost, R. S., Chen, Y., Finin, T., Labrou, Y., & Peng, Y. (1999). Modeling agent conversations with coloured Petri nets. In *Proceedings of the Workshop on Specifying and Implementing Conversation Policies, the Third International Conference on Autonomous Agents (Agents-99)*, Seattle, Washington.

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, Lecture notes in Computer Science*, pp. 178–192. Springer-Verlag.

Cranefield, S., Purvis, M., Nowostawski, M., & Hwang, P. (2002). Ontologies for interaction protocols. In *Proceedings of the Workshop on Ontologies in Agent Systems, the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-02)*, Bologna, Italy.

de Silva, L. P., Winikoff, M., & Liu, W. (2003). Extending agents by transmitting protocols in open systems. In *Proceedings of the Workshop on Challenges in Open Agent Systems, the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-03)*, Melbourne, Australia.

Desel, J., Oberweis, A., & Zimmer, T. (1997). Validation of information system models: Petri nets and test case generation. In *Proceedings of the 1997 IEEE International Conference on Systems, Man and Cybernetics: Computational Cybernetics and Simulation*, pp. 3401–3406, Orlando, Florida.

Finin, T., Labrou, Y., & Mayfield, J. (1997). KQML as an agent communication language. In Bradshaw, J. (Ed.), *Software Agents*. MIT Press.

FIPA site (2003). Fipa - the Foundation for Intelligent Physical Agents, at *www.fipa.org*..

FIPA Specifications (2003a). Fipa Brokering Interaction Protocol Specification, version H, at *www.fipa.org/specs/fipa0000033/*..

FIPA Specifications (2003b). Fipa Contract Net Interaction Protocol Specification, version H, at *www.fipa.org/specs/fipa0000029/*..

FIPA Specifications (2003c). Fipa Interaction Protocol Library Specification, version E, at *www.fipa.org/specs/fipa0000025/*..

FIPA Specifications (2003d). Fipa Query Interaction Protocol Specification, version H, at *www.fipa.org/specs/fipa0000027/*..

Gutnik, G., & Kaminka, G. (2004a). Towards a formal approach to overhearing: Algorithms for conversation identification. In *Proceedings of AAMAS-04*.

Gutnik, G., & Kaminka, K. A. (2004b). A scalable Petri net representation of interaction protocols for overhearing.. In van Eijk, R. M., Huget, M., & Dignum, F. (Eds.), *Agent Communication LNAI 3396: International Workshop on Agent Communication, AC 2004, New York, NY, USA*, pp. 50–64. Springer-Verlag.

Hameurlain, N. (2003). MIP-Nets: Refinement of open protocols for modeling and analysis of complex interactions in multi-agent systems. In *Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS-03)*, pp. 423–434, Prague, Czech Republic.

Huber, P., Jensen, K., & Shapiro, R. M. (1991). Hierarchies in Coloured Petri nets. In Jensen, K., & Rozenberg, G. (Eds.), *High-level Petri Nets: Theory and Application*, pp. 215–243. Springer-Verlag.

Jensen, K. (1997a). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Vol. 1. Springer-Verlag.

Jensen, K. (1997b). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Vol. 2. Springer-Verlag.

Jensen, K. (1997c). *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*, Vol. 3. Springer-Verlag.

Kaminka, G., Pynadath, D., & Tambe, M. (2002). Monitoring teams by overhearing: A multi-agent plan-recognition approach. *JAIR*, *17*, 83–135.

Khomenco, V., & Koutny, M. (2000). LP deadlock checking using partial order dependencies. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR-00)*, pp. 410–425, Pennsylvania State University, Pennsylvania.

Kone, M. T., Shimazu, A., & Nakajima, T. (2000). The state of the art in agent communication languages. *Knowledge and Information Systems*, *2*, 258–284.

Legras, F. (2002). Using overhearing for local group formation. In *Proceedings of AAMAS-02*.

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, Lecture notes in Computer Science*, pp. 193–204. Springer-Verlag.

Ling, S., & Loke, S. W. (2003). MIP-Nets: A compositional model of multi-agent interaction. In *Proceedings of the 3rd International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS-03)*, pp. 61–72, Prague, Czech Republic.

Mazouzi, H., Fallah-Seghrouchni, A. E., & Haddad, S. (2002). Open protocol design for complex interactions in multi-agent systems. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-02)*, pp. 517–526, Bologna, Italy.

Milner, R., Harper, R., & Tofte, M. (1990). *The Definition of Standard ML*. MIT Press.

Moldt, D., & Wienberg, F. (1997). Multi-agent systems based on Coloured Petri nets. In *Proceedings of the 18th International Conference on Application and Theory of Petri Nets (ICATPN-97)*, pp. 82–101, Toulouse, France.

Novick, D., & Ward, K. (1993). Mutual beliefs of multiple conversants: A computational model of collaboration in air traffic control. In *Proceedings of AAAI-93*, pp. 196–201.

Nowostawski, M., Purvis, M., & Cranefield, S. (2001). A layered approach for modeling agent conversations. In *Proceedings of the Second International Workshop on Infrastructure for Agents, MAS and Scalable MAS, the Fifth International Conference on Autonomous Agents*, pp. 163–170, Montreal, Canada.

Odell, J., Parunak, H. V. D., & Bauer, B. (2000). Extending UML in the design of multi-agent systems. In *Proceedings of the AAAI-2000 Workshop on Agent-Oriented Information Systems (AOIS-00)*.

Odell, J., Parunak, H. V. D., & Bauer, B. (2001a). 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.

Odell, J., Parunak, H. V. D., & Bauer, B. (2001b). Representing agent interaction protocols in UML. In Ciancarini, P., & Wooldridge, M. (Eds.), *Agent-Oriented Software Engineering*, pp. 121–140. Springer-Verlag, Berlin.

Parunak, H. V. D. (1996). Visualizing agent conversations: Using enhances Dooley graphs for agent design and analysis. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*.

Paurobally, S., & Cunningham, J. (2003). Achieving common interaction protocols in open agent environments. In *Proceedings of the Workshop on Challenges in Open Agent Systems, the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-03)*, Melbourne, Australia.

Paurobally, S., Cunningham, J., & Jennings, N. R. (2003). Ensuring consistency in the joint beliefs of interacting agents. In *Proceedings of the Second International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-03)*, Melbourne, Australia.

Petri Nets site (2003). Petri nets world: Online services for the international petri nets community, at *www.daimi.au.dk/petrinets*..

Poutakidis, D., Padgham, L., & Winikoff, M. (2002). Debugging multi-agent systems using design artifacts: The case of interaction protocols. In *Proceedings of the First International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS-02)*, pp. 960–967, Bologna, Italy.

Purvis, M. K., Hwang, P., Purvis, M. A., Cranefield, S. J., & Schievink, M. (2002). Interaction protocols for a network of environmental problem solvers. In *Proceedings of the 2002 iEMSs International Meeting:Integrated Assessment and Decision Support (iEMSs 2002)*, pp. 318–323, Lugano, Switzerland.

Ramos, F., Frausto, J., & Camargo, F. (2002). A methodology for modeling interactions in cooperative information systems using Coloured Petri nets. *International Journal of Software Engineering and Knowledge Engineering*, *12*(6), 619–636.

Reisig, W. (1985). *Petri Nets: An Introduction*. Springer-Verlag.

Rossi, S., & Busetta, P. (2004). Towards monitoring of group interactions and social roles via overhearing. In *Proceedings of CIA-04*, pp. 47–61, Erfurt, Germany.

Smith, I. A., & Cohen, P. R. (1996). Toward a semantics for an agent communications language based on speech-acts. In *Proceedings of AAAI-96*.

Wikstrom, A. (1987). *Functional Programming using Standard ML*. International Series in Computer Science. Prentice-Hall.

Xu, H., & Shatz, S. M. (2001). An agent-based Petri net model with application to seller/buyer design in electronic commerce. In *Proceedings of the 5th International Symposium on Autonomous Decentralized Systems (ISAD-01)*, pp. 11–18, Dallas, Texas, USA.