Multi-Agent System Interaction Protocols in a Dynamically Changing Environment

Martin Purvis
Information Science Department
University of Otago
Dunedin, New Zealand
+64-3-479-8318
mpurvis@infoscience.otago.ac.nz

Stephen Cranefield
Information Science Department
University of Otago
Dunedin, New Zealand
+64-3-479-8083
scranefield@infoscience.otago.ac.nz

Mariusz Nowostawski
Information Science Department
University of Otago
Dunedin, New Zealand
+64-3-479-8317
mnowostawski@infoscience.otago.ac.nz

Maryam Purvis
Information Science Department
University of Otago
Dunedin, New Zealand
+64-3-479-8423
tehrany@infoscience.otago.ac.nz

ABSTRACT

An area where multi-agent systems can be put to effective use is for the case of an open collection of autonomous problem solvers in a dynamically changing environment. One example of such a situation is that of environmental management and emergency response, which can require the joint cooperation of a distributed set of components, each one of which may be specialised for a specific task or problem domain. The various stakeholders in the process can all be represented and interfaced by software agents which collaborate with each other toward achieving a particular goal. For such situations new agents that arrive on the scene must be apprised of the group interaction protocols so that they can cooperate effectively with the existing agents. In this paper we show how this can be done by using coloured Petri net representations for each role in an interaction protocol and passing these nets dynamically to new agents that wish to participate in a group interaction. We argue that multi-agent systems are particularly suited for such dynamically changing environments, but their effectiveness depends on the their ability to use adaptive interaction protocols.

Keywords
Multi-agent systems, agent conversations, adaptive systems.

1. INTRODUCTION

There is a widely-held view that multi-agent systems provide a robust and scalable approach for the solution of complex problems [6]. Each individual agent is presumed to be a specialist for a particular task, and the expectation is that, just as in the sphere of human engineering, complex projects can be undertaken by a collection of agents, no one of which has the capability of performing all the required tasks for the project. In addition, if the system has an open agent architecture, then individual agents can be replaced by improved models, thereby enabling the system to improve gradually, grow in scope, and generally adapt to changing circumstances. For agent systems to operate effectively, they must exchange information in the form of messages, and the agents must have a common understanding of the possible message types and the terms (and possible relationships among the terms) that are used in the messages. This shared information can be represented by an ontology, and a considerable amount of research has been devoted to the development of techniques for representing ontologies and for reasoning about messages that have been expressed in terms of them [5]. However, understanding messages that refer to ontologies can require a considerable amount of reasoning, and this may place a computational burden on agents that could limit their overall responsiveness. Thus although the agents are individual specialists concerning their own problem domains, their need to cooperate with other agents in a larger contextual (*i.e.* ontological) environment can present performance problems in areas where agents are operating in real-time, real-world environments, such as environmental response systems or electronic business.

Of course a straightforward way of reducing some of the search space of possible responses to agent messages is by using conversation policies, or interaction protocols [4]. An interaction protocol specifies a limited range of responses that are appropriate to specific message types when a particular protocol is in operation. For example, when a person enters a restaurant, he (or she) doesn't have to worry about all the possible statements that might be made about

food. Instead, he expects to be given a menu and to place an order. Later the food will be brought, and only afterwards (for this particular restaurant, anyway) will he be expected to pay the bill. This may be called a "restaurant interaction protocol", and the existence of such a protocol greatly reduces the search space of possible responses required, which is limited to the responses appropriate to the particular point that one has reached in the protocol. The customer, the waiter, the cook, and the cashier all know this protocol and keep track of where they are in terms of it. (Note that the waiter and the cashier may be holding many simultaneous conversations with various customers, all using the same protocol.)

Everything should work reasonably smoothly if all the agents already know the interaction protocol prior to engaging in an interaction. But what happens in a new or changing environment, where the protocol is either unknown or may need to be changed to meet changing conditions? For example, when a traveler goes to a foreign country and enters an eating establishment, he may not be familiar with the specifics of the restaurant protocol for that locale. In that case he may need to revert to his traveler's dictionary (his ontological reference) and attempt to engage in some sort of complex negotiation in order to carry out even a simple transaction. This will not result in the kind of adaptive and responsive agent system that is needed for changing environments.

Thus for agent systems to operate effectively in highly dynamic environments, they need to have a mechanism for exchanging new or altered interaction protocols on the fly. In this paper we describe an approach to achieve this and discuss how this approach can make agent systems suited to certain types of problem areas.

2. INTERACTION PROTOCOLS USING COLOURED PETRI NETS

When an agent is involved in a conversation that uses an interaction protocol, it maintains a representation of the protocol that keeps track of the current state of the conversation. After a message is received or sent, it updates the state of the conversation in this representation. The Foundation for Intelligent Physical Agents (FIPA) [3] has developed some standard and general interaction protocols that can be adopted by agents, and these have been expressed as state machines [4]. Other representations for interaction protocols have been enhanced Dooley graphs [9] and extended UML [10]. We use coloured Petri nets (CPNs) [7,1], because their formal properties facilitate the modelling of concurrent conversations in an integrated fashion. We believe Coloured Petri nets are a more compact and intuitive representation for modelling concurrent processes than that offered by traditional finite-state machine techniques. Coloured Petri nets are similar to ordinary Petri nets in that they comprise a structure of places, transitions, and arcs connecting those two types of elements; but, in addition, CPNs also have structured tokens and a set of net inscriptions (arc expressions, guards, and place initialisations) which can be evaluated to yield new net markings when transitions are fired. The availability of net analysis tools means that it is possible to check the designed protocols and role interactions for undesired loops and deadlock conditions, and this can then help eliminate human errors introduced in the design process. Moreover, coloured Petri nets facilitate the modelling of individual agent conversations within a larger behavioural modelling context associated with a particular problem domain.

2.1 The FIPA request Interaction Protocol

In order to illustrate the Petri net modelling of agent conversations, we first consider one of the fundamental FIPA message types, the *request* message. When an agent sends a *request* to another agent, it expects a response message a little later, and so we can consider this message sequence a *request interaction protocol* that involves the relatively brief conversation.

We model all interaction protocols in terms of the roles in the interaction: for each role there is a separate Petri net (which differs somewhat from our earlier approach [9]). The collection of individual Petri nets associated with all the interaction protocol roles represents the entire interaction protocol. For every conversation, there are always at least two *roles*: that of the initiator of the conversation and the roles of the other participants in the conversation. In most cases, though, there are only two roles, the initiator and the single participant who receives the first message. In Figure 1, we show the Petri net representation of the initiator of the FIPA *request* interaction. Circles represent Petri net places, and boxes represent transitions. For diagrammatic simplicity, we omit the inscriptions from the diagram, but we will describe some of them below.

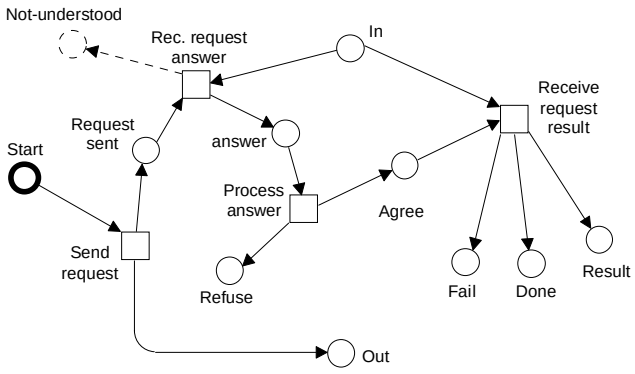

Figure 1. Request interaction protocol for the Initiator role.

The *Start* place will have a token placed there by the agent at the outset of the interaction, and it is highlighted with a thicker line than the other places. A token in this place initiates the interaction. The *In* place (in this and the following Petri net diagrams) will have tokens placed there when the agent receives messages from other agents. The *In* place here is a *fusion node* (a place common to two or more nets): the very same *In* place may exist on other Petri nets that also represent conversations in which the agent may be engaged. Every time the agent receives a message from another agent, a token with information associated with the message is placed in the *In* place that is shared by several Petri nets. The transitions connected to the *In* place have guards on them such that the transitions are only enabled by a token on the *In* place with the appropriate qualification.

Figure 2 depicts the Petri net scheme for the agent that plays the Participant (FIPA uses this term) role, who receives the initial request message.

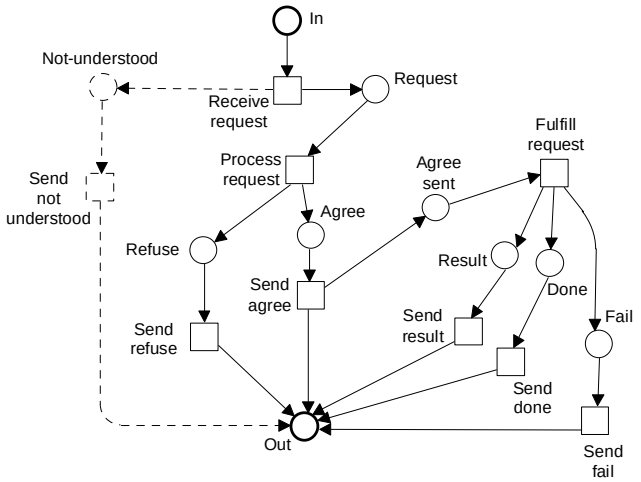

Figure 2. Request interaction protocol for the Participant role.

The Initiator of the request interaction will have a token placed in the *Start* place (Figure 1), and this will trigger the *Send request* transition to place a token in the *Out* place. Thus we are assuming that there is some communication transport machinery that causes tokens to disappear from a Petri net's *Out* place and (usually) a corresponding token to appear on the *In* place of another agent. We do not assume, however, that the transfer is instantaneous, or even guaranteed to occur. It is possible for agents messages to be lost in transit, and thus it is possible for a token to disappear from one role's *Out* place without a corresponding token appearing at another agent's *In* place.

The Participant role-playing agent will get incoming messages and place them in the *In* place. The *Receive request* transition will, if it doesn't understand the message, place a token in the *Not-understood* place. If it does understand the message, it will place a token in the *request* place. Note that the FIPA specifications often include the possibility of a "not-understood" response, but we regard such messages as similar to software exceptions and place them on another, parallel coloured Petri net (not shown) that deals with exceptional conditions and is connected to the primary net by a 'fusion' place. For this reason we show the *Not-understood* places in Figures 1 and 2, and the associated transition for sending a "not-understood" response by dashed lines, which indicate that these nodes are actually located on parallel nets that deal with exceptions. (We discuss parallel Petri nets in connection with policies in Section 6, below.)

If the original request message is understood by the Participant, it either agrees to or refuses the request and sends the appropriate response back to the Initiator (Figure 2). If the Initiator gets an *agree* message back from the Participant and subsequently the enabled *Process answer* transition is fired, a token with the appropriate information is put in the *Agree* place. Meanwhile the Participant attempts to fulfill the original request. In terms of the coloured Petri net, this activity is carried out as part of the coloured Petri net transition's *action* code (executable code that can be activated when a transition is fired) associated with the *Fulfill request* transition. Thus the *Fulfill request* transition is connected with the agent's actual carrying out of the request. Upon completion of the *action*, a token is placed in the *Done*, *Result*, or *Fail* place, depending upon the circumstances of the request action taking place at the Participant agent. The appropriate transition will then be enabled and ultimately fired, and the response will be sent back to the Initiator.

When the Initiator gets the response back from the Participant, the *Receive request result* transition will be enabled if the incoming message contains information that matches with the token that was already stored in the *Agree* place. Note that the Initiator could be involved in several concurrent request interaction conversations, and the placement of specific tokens in the *Agree* place enables this agent to keep track of which responses correspond to which conversations. This is how the coloured Petri net representation assists the agent in managing multiple, concurrent interactions involving the same protocol and can be compared to the manner in which a restaurant waiter keeps track of several 'orders sent to the kitchen so that resulting food preparations can be associated with the right customers.

Thus an interaction protocol has a specific Petri net associated with each role in the conversation, and the

participating agents can use these Petri nets to keep track of what stage they are in the conversation. The Petri nets representing all the roles of an interaction protocol can be encoded in XML and sent as the message content of a FIPA *inform* or *propose* message. This means that a new agent that appears on the scene can be sent the interaction protocol and can "load" the appropriate Petri net role into its conversation module and engage on-the-fly in a conversation using the new interaction protocol.

4. INTERACTION PROTOCOLS FOR ENVIRONMENTAL EMERGENCY SYSTEMS

Let us now consider a somewhat more involved situation, the management of extended environmental areas when unforseen events take place. This can require rapid responses on the part of many people or services with specialised skills. In such circumstances the resources and skills required to respond to the emergency may go well beyond the capabilities of the permanent staff who normally maintain the area. For example, when a massive forest fire breaks out in a national forest or when a blizzard threatens the lives of several scattered groups of trampers in a national park, the environmental managers may need to call on the services of a number of specialists who can provide crucial assistance in connection with specialised rescue operations and medical assistance. In today's economic and political climate, it is more likely that these specialist service providers are private operators who can be contracted by the government to respond to emergencies in critical situations, rather than people under the permanent employ of the government. Environmental management systems attempt to manage these dynamic situations.With the increasing use of wireless communications, system components may come in and out of range as environmental professionals move around in the field.

We examine a scenario in which a national park is managed by a collection of park rangers. The park has many tracks available for trampers, and tramping groups have guides equipped with personal digital assistants which can be used to contact park officers when necessary, such as in an emergency situation. Although park rangers can handle routine events themselves, there can be cases when they need outside assistance. If trampers are trapped in a remote location, they may need to be rescued quickly. This can require specialist medical personnel, firefighting professionals and equipment, professional mountain climbers and speleologists, and special types of transport (four-wheel-drive trucks, airplanes, helicopters, boats, etc.).

These specialist groups can be authorised to provide their services in an emergency and be compensated accordingly. As new people with special skills move into the national park region, they can be added to the network of potentially participating service agents that can be called upon to provide assistance in emergency situations. For open agent-based environmental management systems, new rescue service providers can plug into the system, as long as they are provided with the appropriate interaction protocols.

In this scenario we assume that a skilled tramping guide or park officer has assessed the situation and has an idea of the kind of assistance that is needed. This person has a personal software agent that we call the "Initiator". The Initiator communicates with the "Ranger" agent, who is a service recruiter, in order to find out what resources are available and see what can be organised. A simplified, top-level view of this activity can be shown for illustrative purposes in terms of the Petri net shown in Figure 3. Note that this Petri net does not represent an interaction protocol role, but, instead, represents a simplified view of the overall interaction. Here the Initiator sends a "proxied" message to the Ranger: within the content of the message to the Ranger is another message that is to be sent other service agents who will attend to the problem. The Ranger, here, is a recruiter of specialised agents who can deal with the particular emergency at hand. Thus the Initiator can send a message that (a) contains a request that the Ranger recruit service agents and (b) contains the interaction protocol to be used to interact with the target service agents.

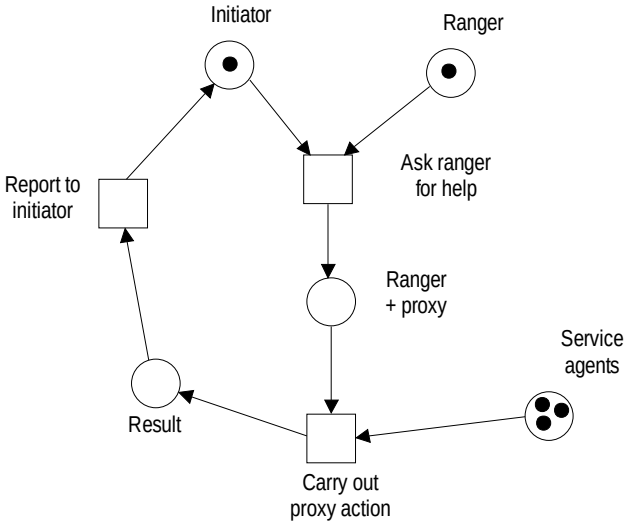

Figure 3. Top-level process model. Recruitment of service providers.

Figure 3 over-simplifies the situation, because it shows a synchronous interaction between the Initiator and the Ranger, which, as we have mentioned above, it not a true representation of asynchronous messaging. As a result of this interaction, the Ranger has received the proxied

information from the Initiator that is to be transmitted to the Service agents. The proxied information from the Initiator can actually involve a complex and asynchronous exchange of messages between the Ranger and the Service agents. That is, the Initiator asks the Ranger to carry out some sub-protocol interaction with the Service agents and then have the results from this sub-protocol sent back to the Initiator and possibly other parties, which, following FIPA, are designated as Destinators. This interaction corresponds to the FIPA *Recruiting Interaction Protocol Specification* [3].

There are four basic roles associated with this more complicated interaction protocol: Initiator, Ranger (the recruiter), Service-agent (the target agents of the original communication from the Initiator), and Destinator. It is here, in the context of such more complicated conversations, that dynamic interaction protocol modelling and exchange can help demonstrate how agent-based systems can respond effectively to changing conditions.

Figure 4 shows the coloured Petri net model for the Initiator role in the interaction protocol. Again, the conversation is begun when a token is placed in the *Start* place. The initial message sent contains the proxy message that specifies the additional interaction that is to take place between the ranger and some service agents.

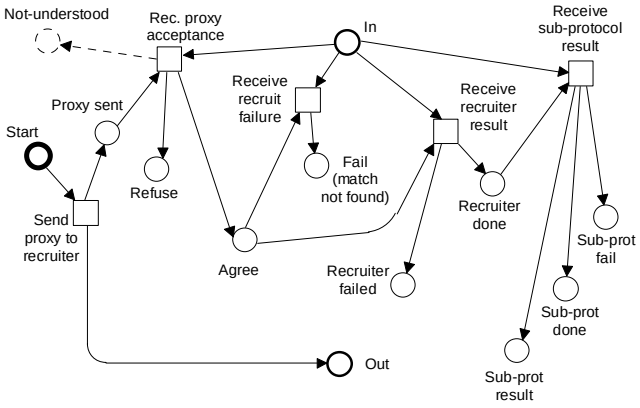

Figure 4. Recruiting interaction protocol for the Initiator role.

Figures 5 shows the interaction protocol role model for the Ranger (recruiter), and Figure 6 shows the interaction protocol role model for the target agents that execute the sub-protocol. The places identified with thick borders in Figures 5 and 6 (e.g. *Start sub* and *Sub-protocol result*) represent fusion nodes that are connected to other, parallel Petri nets (not shown), which carry out the sub-protocol interaction. One possible example of a sub-protocol interaction is the *Request protocol* shown in Figures 1 and 2.

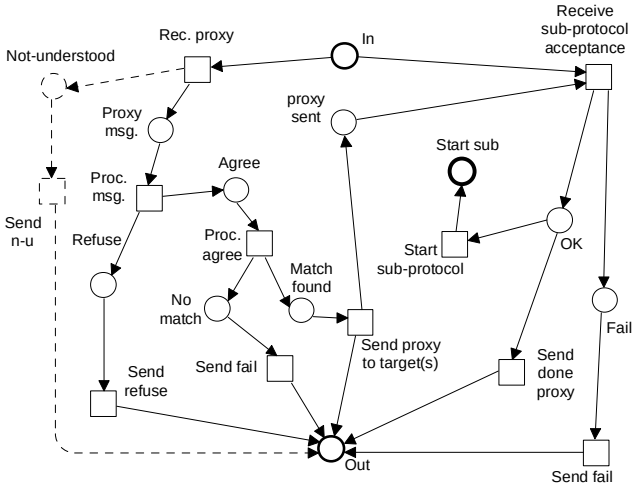

Figure 5. Recruiting interaction protocol for the Recruiter role.

When the recruiter receives the proxy message from the Initiator (Figure 5), it either agrees or refuses to carry it out and sends an answer back to the Initiator. If it agrees to the action, it checks if it knows of any target agents that can carry out the requested proxy action. If there are none ('no match'), it sends a failure message back to the Initiator. If, however, it does find a match, it sends the requested proxy action to the target agent(s). A target agent may agree or refuse to carry out the proxy action, and it reports this response to the Recruiter (Figure 6). If the target agent agrees, then a sub-protocol interaction is started between the Recruiter and the target agent. The steps associated with this sub-protocol interaction take place on another, parallel Petri net that is not shown here. When this proxy interaction is completed (it could result in failure), the results are reported back to the Initiator and/or possibly additional Destinators (Figure 7).

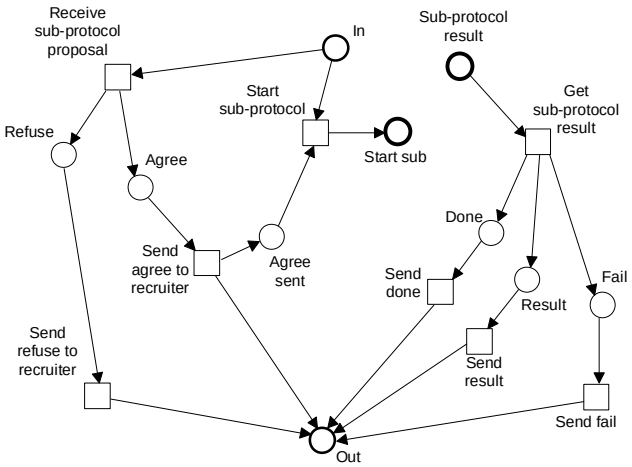

Figure 6. Recruiting interaction protocol for the Target agents role.

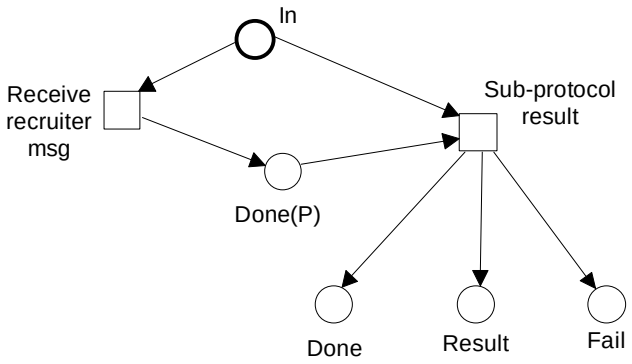

Figure 7. Recruiting interaction protocol for the Destinator role.

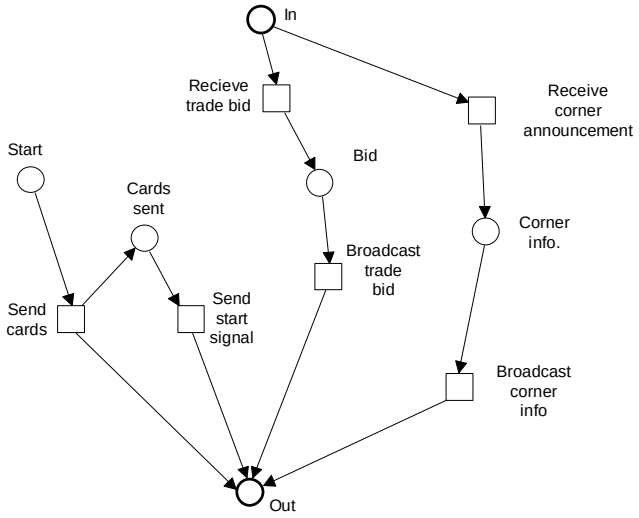

Figure 8. The Pit Game interaction protocol for the Dealer role.

5. E-BUSINESS APPLICATIONS

We show another example of agent interaction in the area of electronic commerce. For illustrative purposes, we look at the *Pit Game* [11], which is a card game that simulates commodity trading and exhibits some of its essential characteristics. In this game a dealer distributes nine cards to each player (three to seven players may play). There are exactly nine cards of each of the commodity types, corn, barley, wheat, rice, etc. The card deck is prepared so that the number of commodity types matches the number of players for the given game. When play begins, the players independently and asynchronously exchange cards with each other, attempting to "corner" the market by getting all nine cards of one type. They can only exchange cards that belong to the same commodity type. Thus if a player has six barley cards, two wheat cards and one rice card, he will typically initially attempt to trade away his two wheat cards, hoping to acquire one or two barley cards. Trading is carried out by a player annoucing, for example, that he has, say, two cards to trade. If another player also wishes to trade two cards, the two players may make an exchange. Whenever a player manages to get a "corner", he announces that fact to the dealer and the "hand" is finished (the implementation shown here is for a single "hand").

In our implementation of the game, the trading bids are sent to the dealer, who, in turn, broadcasts the bids to all the players. Figure 8 show shows the interaction protocol for the Dealer role. The Dealer deals out the cards and then sends the "start" signal to all the players (a broadcast message).

Whenever a player (Figure 9) has a hand of cards, he always checks to see if he has a "corner". If so, he announces this to the Dealer, who, in turn, announces it to the rest of the players, signalling the end of the hand. After players have received the start signal and assuming that they don't have a "corner", they may choose to make a bid. They do this by sending their bid to the Dealer, which in turn broadcasts the bid to all other players. At this point, the player's cards are also separated into the "cards offered" and the "cards remaining" places. When a bid is received, the player may choose to accept the bid. He does this by checking the bid against his own cards. If the bid is accepted, a message is sent to the player (not the Dealer) and a token is stored in a

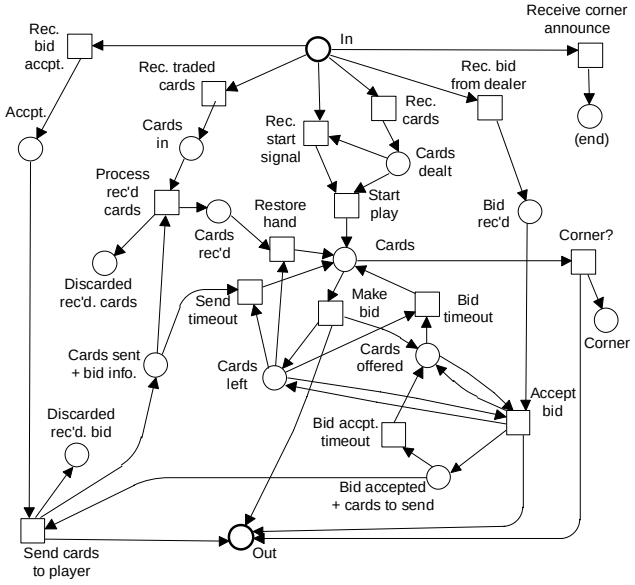

Figure 9. The Pit Game interaction protocol for the Player role.

place ("bid accepted + cards to send") for future reference. (If the player doesn't get a corresponding acceptance of his own bid before a given timeout period, then he gives up on this potential deal and restores his offered cards back to the "cards offered" place.)

A trade of cards can take place if two players have made bids, and they have both accepted each other's bid. Thus if player A has made a bid, has accepted player B's bid, and has received an acceptance message from B that his own bid has been accepted, then A will send his cards to B (and expect to receive a corresponding number of cards from B). When a player receives a message from another player that his bid has been accepted, it is stored in the "Accpt." place. The "Send cards to player" transition checks (by means of a guard) to make sure that the accepted bid matches information in the token located in the "bid accepted + cards to send" place. If so, it sends the cards to the other player and keeps a copy of the acceptance information in the "Cards sent" place. If the player receives a bid acceptance that is not applicable (such as a second acceptance that has come in after he has already decided to trade cards with someone who has sent in an earlier acceptance), then the bid acceptance is discarded. When traded cards are received, the "Process rec'd cards" transition checks to see that the received cards are associated with the bid acceptance information stored in the "Cards sent" place. If the cards do not match the bid acceptance, they are discarded. If nothing is received after some time, the "Send timeout" transition guard is enabled, and the cards are returned to the "Card" place. (Though the cards have been sent, the player still has a copy of what has been sent.) The "Bid timeout" transition is enabled if there have been no takers of a bid before a certain timeout period has elapsed. When this transition is fired, the cards are returned to the hand, and the player may chose to make another bid.

In real e-commerce trading situations new agent "players" could be sent the appropriate interaction protocols, similar but more elaborate to those shown in this example, and immediately begin participating in the trading arena.

6. IMPLEMENTATION

The agent-based system that we have developed is based on our Opal agent platform [13], and uses JFern [8], a Java-based coloured Petri net simulator. When new agents appear and are to be incorporated into the network of available agents, they are sent a FIPA *Propose* message by the group manager with a message content containing an XML-encoding of the interaction protocol that is used. The target agent parses the XML-encoded interaction protocol, and if it accepts the proposed mechanism for interaction, sends the *Accept* message back to the group manager.

We implement the agent conversation module as a layered Petri net. For each protocol, there is can be an additional layer that we have not shown, called a *policy* layer. This is the layer which would, with other approaches, be left to the agent application to coordinate and not be included explicitly in the conversation modelling process. However we feel that it is more appropriate to treat it as closely related to the conversation layer.

Policies may be implemented simply by set of rules, or, in more complex cases, they may have their own complex protocols that exist and change state in parallel with the immediate context of an ongoing conversation. Under these more complex circumstances, there might be a "policy-level interaction protocol" (another protocol, but at the policy level). It is under these conditions that we can benefit from having another modelling layer at the policy level, above that of the ordinary conversational modelling layer. The two layers can be joined together by representing them both as a coloured Petri Net.

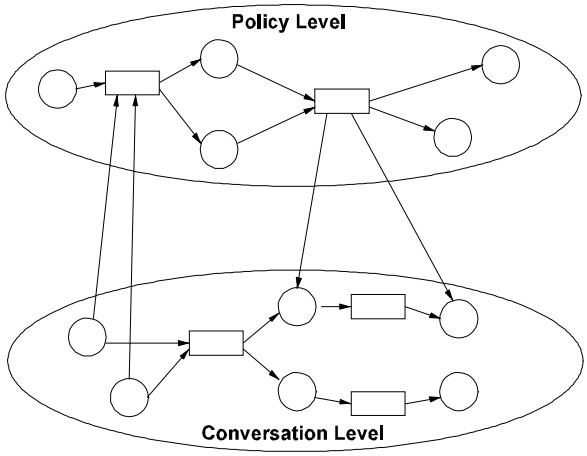

Figure 10. Petri net with conversation and policy levels.

In the Pit Game, for example, the possible rules for legal bids and legal card exchanges by the players are described by the basic interaction protocol. But existing above that level of abstraction is another level of discourse that can take place during the game. Suppose one of the players has a question concerning the official rules of the game and wants to have a ruling made by a referee. Or perhaps one of the players at some point wants to halt play so that he or she can attend to some urgent matter. These kinds of 'interrupts' or 'exceptions' are common to many kinds of interactions and can take place at almost any time. We already mentioned an example of this type of action in connection with the "not-understood" message in the FIPA interaction protocol specifications. The discourse involved in these interrupts are usually "off-topic" from the context of the

immediate conversation, and in fact they are often *about* the conversation that is taking place (such as an accusation of breaking the protocol rules associated with playing the game). Since they are likely to be "off-topic" and can occur at any moment, it can be tedious to include these kinds of conversational strands in the given (domain-specific) conversation protocol. To do so would clutter the visual simplicity of the original conversation protocol and would lessen the value in providing a easy-to-comprehend visual modelling representation of the interaction. On the other hand, to leave out the possibility of representing such events is to ignore the possibility of their occurrence and consequently means that there is a failure to model the world adequately so that its essentially contingent nature is recognised. Our solution is to model these kinds of interactions that can guide, interrupt, or redirect existing conversations by representing them as another, parallel modelling layer above that of the existing conversation layer. This idea was suggested in [2] for specific types of conversation, but we have generalised the notion and incorporated it into a Petri Net representation. Thus a *conversation* is a combination of *protocols* being instantiated and manipulated by a particular *policy*.

7. CONCLUSIONS

We have developed an approach for managing non-trivial interactions in a multi-agent system and have demonstrated how it can be used to make systems more responsive to changing protocols in a dynamic environment. In particular, we believe that this facilitates the operation of multi-agent systems in connection with applications where new service-providing agents are likely to appear and need to be given updated interaction protocols so that the group of agents can adapt to a changing environment.

Our approach is based on a network of interacting agents that follow the FIPA specifications, but we have introduced and implemented an interaction protocol mechanism based on coloured Petri nets that may include more complex interaction protocols than what FIPA has so far specified. When more complicated, multi-layered and concurrent conversations take place among groups of agents, the coloured Petri net approach that we use appears to offer advantages over the state-machine techniques that have been commonly used up until now.

8. REFERENCES

[1] Cost, S., Chen, Y., Finin, T., Labrou, Y., and Peng, Y., "Using
colored Petri nets for conversation modeling, *Issues in Agent Communication*, Lecture Notes in AI, Springer-Verlag, Berlin (2000).

[2] R. Elio and A. Haddadi. On abstract task models and conversation policies. In Working Notes of the Workshop on Specifying and Implementing Conversation Policies, pages 89-98, May 1999.

[3] FIPA. Foundation For Intelligent Physical Agents (FIPA). FIPA 2001 specifications, http://www.fipa.org/specifications/, 2001.

[4] Greaves, M, and Bradshaw, J. (eds.), *Specifying and Implementing Conversation Policies*, Autonomous Agents '99 Workshop, Seattle, WA, (May 1999).

[5] Gruber, T. R., "A Translation Approach to Portable Ontologies", *Knowledge Acquisition, (1993) 5(2):199-220.*

[6] Jennings, N. R., "Agent-oriented software engineering", *Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI*, (1999).

[7] Jensen, K., *Coloured Petri Nets – Basic Concepts, Analysis Methods and Practical Use*, Springer-Verlag, Berlin, 1992.

[8] Nowostawski, M., *JFern*, version 1.2.1, http://sourceforge.net/project/showfiles.php?group_id=16338, 2002.

[9] Nowostawski, M., Purvis, M., and Cranefield, S., "A Layered Approach for Modelling Agent Conversations", *Proceedings of the 2nd International Workshop on Infrastructure for Agents, MAS, and Scalable, MAS*, 5th International Conference on Autonomous Agents (2001) 163-170.

[10] Odell, J, Parunak, H. V. D., Bauer, B., "Extending UML for agents", *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, pp. 3-17, 2000.

[11] Parker Brothers, Inc., Salem, Mass., 1919. see http://www.centralconnector.com/GAMES/pit.html.

[12] Parunak. H. V. D., "Visualizing agent conversations: Using Enhanced Dooley graphs for agent design and analysis", *Proceedings of the Second International conference on Multi-Agent Systems ICMAS'96* (1996).

[13] Purvis, M., Cranefield, S., Nowostawski, M., and Carter, D., "Opal: A Multi-Level Infrastructure for Agent-Oriented Software Development", *Information Science Discussion Paper Series*, No. 2002/01, ISSN 1172-6024, University of Otago, Dunedin, New Zealand, http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2002-01.pdf.gz, (2002).