Interaction Protocols for a Network of Environmental Problem Solvers

M. K. Purvis [a], P. Hwang[a], M. A. Purvis[a], S. J. Cranefield[a], and M. Schievink[b]

[a]*Information Science Department, University of Otago, Dunedin, New Zealand*

[b]*Department of Computer Science, University of Twente, Enschede, The Netherlands*

Abstract: Environmental management and emergency response often involves the joint cooperation of a network of distributed problem solvers, each of which may be specialised for a specific task or problem domain. Some of these problem solvers could be human, others could be 'intelligent' environmental monitoring and control systems. Environmental software systems are needed not only for the provision of basic environmental information but also to support the coordination of these problem solvers. An agent architecture can support the requirement associated with disparate problem solvers. The various stakeholders in the process are represented by software agents which can collaborate with each other toward achieving a particular goal. The communication between agents can be accomplished by using interaction protocols which are represented by coloured Petri nets (CPN). This paper describes an approach for providing this support by employing a software agent framework for the modelling and execution of environmental process tasks in a networked environment.

1. INTRODUCTION

The management of extended environmental areas when unforseen events take place 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. Consider, for example, what happens 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. In these cases 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 information systems (EIS) take on the ambitious task of providing decision and management support for operations in the context of large, complicated parts of the natural environment. To fulfill this task they require access to as much information as possible concerning how the environmental area is modelled and what operational capabilities are available for application in real time. With the increasing use of wireless communications, EIS components may come in and out of range as environmental professionals move around in the field. Thus, although many environmental information systems are closed systems (involving a fixed number of participants and data sources), there are situations where an open EIS is appropriate. For such an open EIS, dynamic distributed information system technology is needed so that the systems can adjust to changing conditions and provide satisfactory responses in a timely manner. In this paper, we describe some key features of such technology based on an open system of interacting software agents [Jennings, 1999]. A key component of this technology is both the specification and exchange of interaction protocols [Greaves and Bradshaw, 1999] for agents to use in a dynamic environment. This agent-based technology is described in the context of an example scenario that illustrates its operational aspects.

2. AGENT-BASED SYSTEMS

2.1 Software agents

Agent-based software engineering is based on the notions that (a) large interactive software systems should be built using modelling approaches that take advantage of abstraction, decomposition, and organisation [Booch, 1994] and (b) a collection of interacting agents offers the most intuitive, robust, and scalable modelling approach with those characteristics. With agent-based software a loosely-coupled collection of agents can cooperate to achieve a common goal. Each individual agent is presumed to be a specialist for a particular task, and the expectation is that, just as is 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 of 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. (Note that the agent system characteristic of being loosely coupled and open means that the communication is asynchronous: unlike a function or method call, there is no 'return' information available, and there is not even a guarantee of message delivery.) This model is particularly apt for the type of dynamic EIS under discussion, since many of the system components represent actual, physical human agents that will be brought to bear to handle an environmental problem.

For software agent systems to operate, the agents must be able to 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 connection with message 'content'. This shared understanding of the 'world' is referred to as an ontology, and considerable agent-based software engineering 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 [Gruber, 1993]. Understanding messages that refer to ontologies can require a considerable amount of reasoning, and consequently agents with such a capability must be highly 'intelligent' (i.e. equipped with advanced artificial intelligence technology).

2.2 Interaction protocols

However, there is a straightforward way of reducing the search space of possible responses to an agent message, and it is one that is also used by humans. This approach uses what are called "conversation policies", or sometimes "interaction protocols". Consider what happens when someone enters a restaurant and sits down at a table. Such a person doesn't have to worry about all the possible statements that might be made about food. Instead, he or she 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 or she be expected to pay the bill. This is 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.

With a software agent system, the same approach is used in connection with interaction protocols. If two agents share the same protocol, they can engage in a conversation and keep track of where they are in terms of the protocol.

For an agent-based EIS, information repositories or organisations that provide some service are interfaced to the system by agents. The agents representing these components will interact based on interaction protocols that they share. If the system is to be open, it must be possible for new agents to appear on the scene (a new service-provider, for example) and interact with the system. This means that newly appearing agents must be able to acquire the appropriate interaction protocol information. In the next section we describe how we achieve this.

3. 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) [FIPA, 2001] is an organisation devoted to the development of international standards for the interoperation of software agents. FIPA has developed some standard and general interaction protocols that can be adopted by agents, and these have been expressed as state machines [Greaves and Bradshaw, 1999]. Other representations for interaction protocols have been enhanced Dooley graphs [Parunak, 1996] and extended UML [Odell, *et al.*, 2000]. We use coloured Petri nets (CPNs)

[Jensen, 1992, Cost *et al.*, 2000], because their formal properties facilitate the modelling of concurrent conversations in an integrated fashion. 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. Additionally, coloured Petri nets 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. Moreover, coloured Petri nets facilitate the modelling of individual agent conversations within a larger behavioural modelling context associated with a particular problem domain.

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.

Whenever there is a 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 just 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.

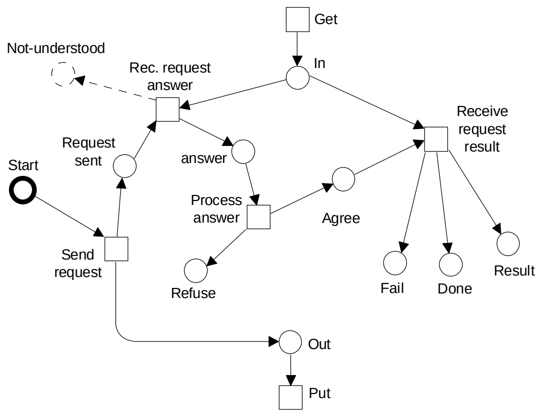

Figure 1. Request interaction protocol for the Initiator role.

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. In each model, there is a *Get* transition that has code associated with it (a net inscription) that obtains appropriate information from the agent's message receiving module (external to the Petri net model) and places it in the *In* place. The *In* place here is a *fusion node* (a place that is common to two nets): the very same *In* place may exist on other Petri nets that also represent conversations in which the agent may be engaged. 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.

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. This will, in turn, enable the *Put* transition, which has code associated with it that interacts with the agent's message sending module (external to the Petri net).

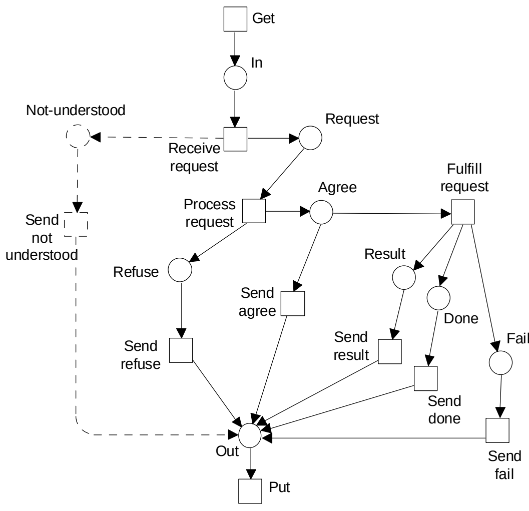

Figure 2. Request interaction protocol for the Participant role.

The Participant role-playing agent will get the incoming message (when the *Get* transition fires) and place it 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. 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 (a place common to two nets). 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.

If the message is understood by the Participant, it either agrees 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 the *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 and ultimately 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 (external to the Petri net). The appropriate transition will then ultimately be fired, and the response will be sent back to the Initiator.

When the Initiator gets the response, 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 like a waiter at a restaurant who might send several 'requests' to the kitchen and needs to keep track of the responses so that they 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.

4. INTERACTION PROTOCOLS FOR EIS PROBLEM SOLVERS

So with this technology available, let us see how it can be applied to an open, dynamic EIS involving a network of environmental problem-solving agents. As an example, we consider the following 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 (PDAs) 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 (such as four-wheel-drive trucks, airplanes, helicopters, and boats).

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 EISs, new rescue service providers can plug into the system as long as they are provided with the appropriate interaction protocols that are used by the given EIS. In section 5, we explain how this is accomplished.

In this scenario we assume that a skilled tramping guide or park ranger 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" EIS agent 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.

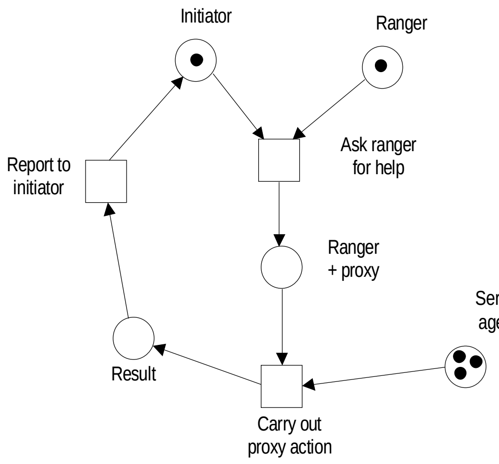

Figure 3. Top-level process model. Recruitment of service providers.

This over-simplifies the situation, because it shows a synchronous interaction between the Initiator and the Ranger. As a result of this interaction, the Ranger has the "proxy" information from the Initiator that is to be transmitted to the Service agents. In the real situation, however, a more complicated interaction can take place. The proxy 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 another party, which, following FIPA, we call the Destinator. This corresponds to the FIPA *Recruiting Interaction Protocol Specification* [FIPA, 2001].

Thus 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 interaction protocol modelling and monitoring can be essential for agent-based EIS.

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.

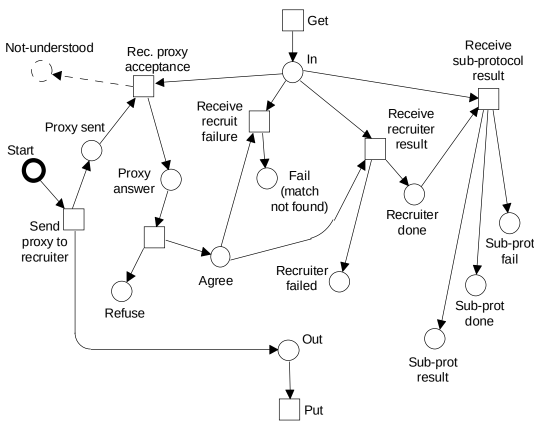

Figure 4. Interaction protocol for the Initiator role.

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.

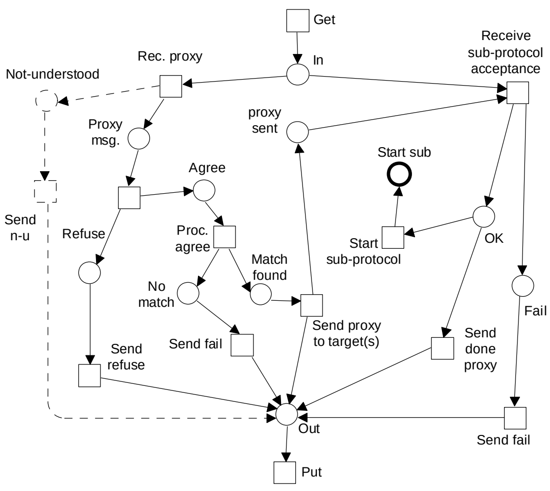

Figure 5. Recruitment interaction protocol for the recruiter role.

When the recruiter receives the proxy message from the Initiator (Figure 5), it either agrees or refuses 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 agents(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 an additional Destinator (not shown).

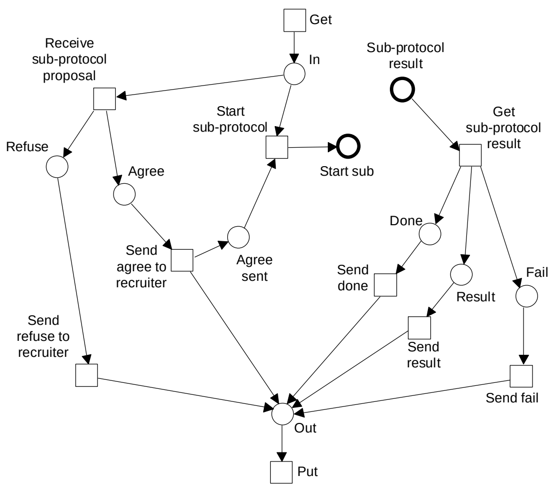

Figure 6. Recruitment interaction protocol for the target agents role.

5. IMPLEMENTATION

The agent-based system that we have developed is based on our Opal agent platform [Purvis *et al.*, 2002], and uses JFern [Nowostawski, 2002], a Java-based coloured Petri net simulator. When new agents appear as service providers and are to be incorporated into the network of available service providing agents, they are sent a *Propose* message (one of the FIPA-specified interactions) with a message content containing an XML-encoding of the interaction protocol that is used by the EIS network. 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 EIS.

The Opal system and the interaction protocol modules are implemented in Java and are being ported to the J2ME (http://java.sun.com/j2me/) environment for use on wireless platforms that can be applied in the EIS domain.

6. CONCLUSIONS

We have developed an approach for managing non-trivial interactions among a network of problem-solving service providers and have demonstrated how it can be used in connection with open, distributed environmental information systems. 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.

For the kinds of systems we have described, it is important not only to acquire information, but to specify actions to be taken, and we believe this is of fundamental importance for the future of environmental management systems.

7. ACKNOWLEDGEMENTS

The authors wish to thank Mariusz Nowostawski, the author of JFern and a principal developer of the OPAL system, for technical assistance.

8. REFERENCES

Booch, G., *Object Oriented Analysis and Design with Applications*. Addison Wesley, 1994.

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

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

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

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

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

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

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

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.

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).

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 (2002), http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2002-01.pdf.gz