

# Interaction Protocols for a Network of Environmental Problem Solvers

M. K. Purvis<sup>a</sup>, P. Hwang<sup>a</sup>, M. A. Purvis<sup>a</sup>, S. J. Cranefield<sup>a</sup>, and M. Schievink<sup>b</sup>

<sup>a</sup>*Information Science Department, University of Otago, Dunedin, New Zealand*

<sup>b</sup>*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 unforeseen 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 trappers 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.

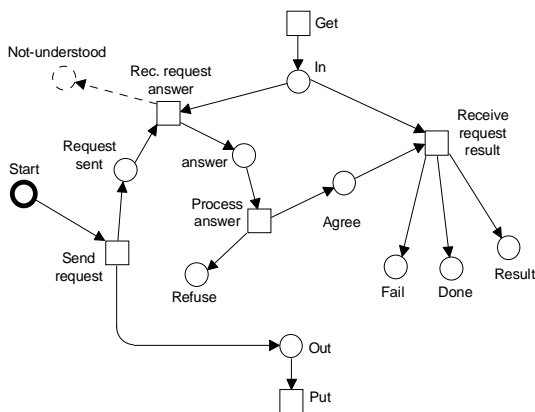
## 3. INTERACTION PROTOCOLS USING COLOURED PETRI NETS

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.

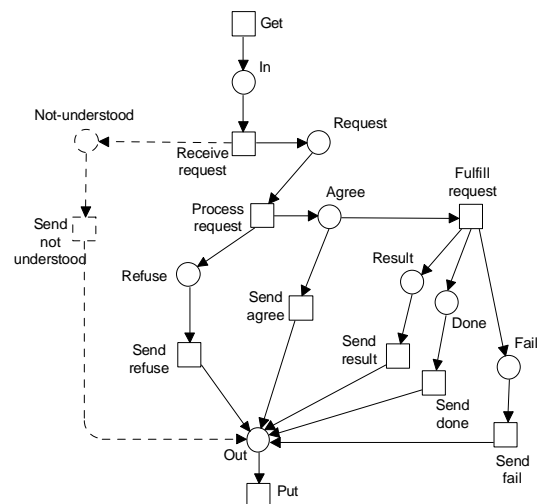


**Figure 1.** Request interaction protocol for the Initiator role.

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





