Agent Modelling with Petri Nets

Dr Martin K. Purvis[1]
Dr Stephen J.S. Cranefield
Computer and Information Science
University of Otago

March 1996

Abstract

The use of intelligent software agents is a modelling paradigm that is gaining increasing attention in the applications of distributed systems. This paper identifies essential characteristics of agents and shows how they can be mapped into a coloured Petri net representation so that the coordination of activities both within agents and between interacting agents can be visualised and analysed. The detailed structure and behaviour of an individual agent in terms of coloured Petri nets is presented, as well as a description of how such agents interact. A key notion is that the essential functional components of an agent are explicitly represented by means of coloured Petri net constructs in this representation.

[1] Address correspondence to: Dr M.K. Purvis, Senior Lecturer, Department of Information Science, University of Otago, P.O. Box 56, Dunedin, New Zealand. Fax: +64 3 479 8311 Email: mpurvis@commerce.otago.ac.nz

1 Introduction

The increasingly complex processes of the industrial world often involve real-time responses to complicated sequences of events. In order to construct such systems effectively in the future, it will be necessary to employ advanced modelling approaches that can flexibly incorporate the roles of information servers, electronically controlled devices, and human actors so that the new systems can be embedded into existing processes. Examples of such systems are those in the area of computer-supported cooperative work, where key components to be modelled are the human participants in the cooperative processes. If a new modelling approach is to offer significant advantages, it must offer the capability of simulating the behaviour of these systems and that of their individual components.

Object-oriented systems and high-level Petri nets are two modelling approaches that individually offer particular advantages for the development of complex, concurrent systems, and there has been continued interest in advances along both fronts. Object-orientation offers attractive constructs for encapsulation and the partitioning of procedures and namespaces, while Petri nets offer an elegant graphical formalism for the examination and analysis of concurrent behaviour. Recently, there have been efforts to merge aspects of both object-oriented systems and Petri nets in order to realise the combined advantages of the two approaches [1-5].

Modelling and development of concurrent systems with intelligent software agents is another promising approach. Agents are autonomous elements endowed with intentions that are usually stored in a declarative manner and serve as a natural metaphor for modelling system components and their interactions. They are particularly appealing for the representation of group members communicating over a network, where the system is open and the number and nature of interacting players can change at any time.

Our approach seeks to combine the notion of an intelligent agent with that of coloured Petri nets [6]. The goal is to gain the benefits from both the natural capability of agents for modelling real distributed system applications and the capability of Petri nets for modelling synchronisation and concurrency. As will be discussed below, the approach differs from other efforts [7,8] by explicitly representing the functional make-up of an agent by means of coloured Petri nets. The following section describes the nature of our agent modelling approach, and the subsequent section discusses the manner in which Petri nets are incorporated into it.

2 Modelling With Agents

Consider how the modelling of any complex dynamic system is initially performed in the everyday world. We observe that

(a) it is natural to conceptualise the relevant features, *i.e.* the behavioural components to be modelled, in terms of simple or familiar elements, and
(b) the overall model structure and the number of individual elements must be kept simple enough so that the entire model can be easily understood, manipulated, and modified, if necessary.

If the system to be modelled is significantly complex, restriction *b* means that the individual elements must, themselves, encompass a fair degree of complexity. In that case it is appropriate to express these complex elements in terms of those dynamic entities from the real world with which we are most familiar: human agents; and in fact we intuitively construct mental models of this nature all the time. Software agents enable us to map these agent mental models directly into a computer representation, and consequently they facilitate the development of bigger and more complex systems [9,10].

There is considerable interest in agents and consequently a range of views concerning just what characterises an agent, but some of the essential properties are [11]

- *autonomy:* agents operate without direct (step-by-step) control of their actions from the outside;
- *goals or intentions:* agents are perceived to have goals that they attempt to achieve and may achieve these goals in various ways;
- *memory:* agents can remember past events (they retain state) and can possibly improve their behaviour as a result of this memory;
- *reactivity* and *proactivity*: agents perceive their environment and react to changes in it or direct actions upon them; they are also able to take the initiative and undertake actions in pursuit of their goals.

In our approach, we add an additional component of agents: *tools* (or *internal utilities*). Every modelling element in our system is essentially an agent, but there are two degenerate forms of agent to which we give alternative names: objects and utilities. Thus the modelling world consists of three types of entity: **agents, objects,** and **utilities** (Figure 1). Agents can operate on other agents, objects, and utilities can have new goals installed by receiving "commands" from other agents. Objects, on the other hand, are acted upon by agents and cannot initiate any action; they essentially store information. Utilities are similar to the internal utilities possessed by agents, except that they can be commanded to act on an object (and on other agents) by external agents. Figure 1 shows the lines of action available to the two agents P1 and P2, in the presence of an object and a utility. This figure could represent two engineering groups (P1 and P2), each using a common machine (Utility) for the development and modification of a specific product (Obj).

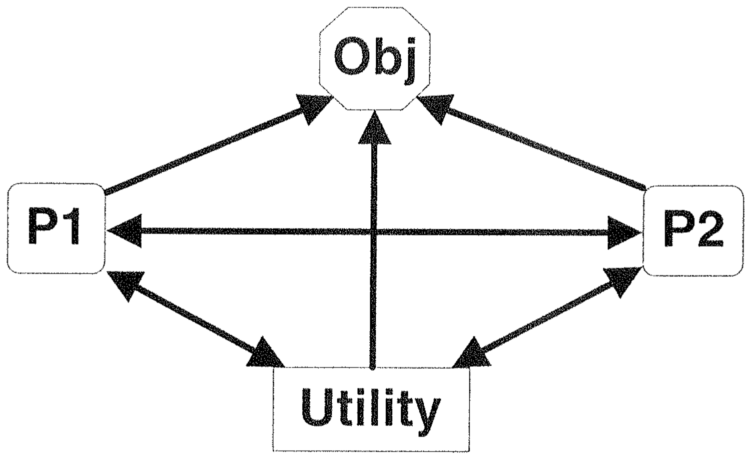

Figure 1: Lines of action available to agent modelling types

2.1 Examples

As examples, consider how (a) a video disk player and (b) a raster display system might be initially modelled in terms of these agents.

The video disk player is shown in Figure 2 [12]. There are several agents in the diagram. The lens has a servo-controller. It focuses light on the disk and receives the reflected light. The lens uses its servo-controller to keep this reflected light in focus for the light-sensitive diodes that convert reflected light signals to FM electronic signals for the signal processing electronics agent. The servo-controlled tracking mirrors detect and correct tracking errors to keep the light signal in the centre of the track and to correct for variations in rotation speed of the disk. The optical divider utility sends light received from the laser to the light-sensitive diodes to serve as a reference signal. It also sends light received from the disk by way of the tracking mirrors. The signal processing electronics agent compares the two light signals received and computes correction information that is sent to the tracking mirrors. Each of the agents has a limited set of goals and capabilities with which to carry out appropriate action. There are, of course, many different ways to model the same system at a high level, but this representation identifies some of the principal actors that must be designed in further detail.

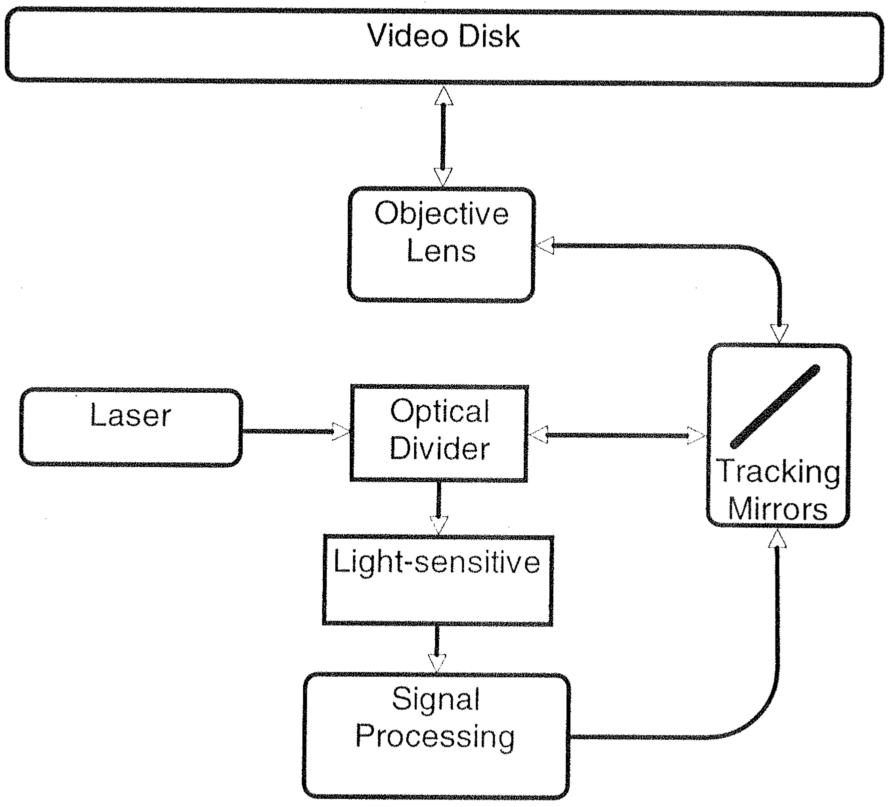

Figure 2: Agent model of video disk player

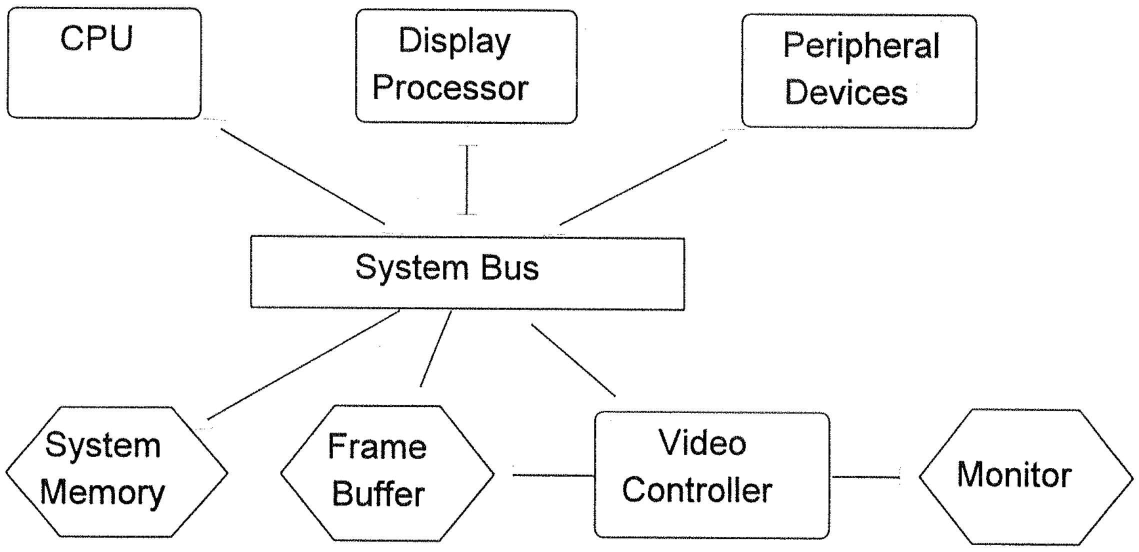

Figure 3: Agent model of raster display system

Figure 3 is a schematic representation of a single-address-space raster display system architecture [13]. Here the system bus is a utility that is used by the display processor, the CPU, and the video controller. By means of the bus, they access objects like the frame buffer and the video monitor.

3 Agents With Coloured Petri Nets

The efforts to combine the features of object-orientation and Petri nets can be grouped around two basic approaches [14]. In one approach, the overall control structure of the system is modelled by a single Petri net, while the tokens represent object instances of abstract data types. In the alternative approach, nets are used to provide a model of the internal behaviour of individual objects. Our approach is somewhat similar to this second line of development, except that we are using nets to model *agents*, rather than objects. As such, the basic functional components of an agent (the retention of memory, the possession of goals, the containment of internal utilities) are explicitly represented in terms of Petri nets. By means of this approach, a high-level representation in terms of agents, can be mapped down into a complete representation in terms of coloured Petri nets (CP-nets) and thereby take advantage of the features of CP-nets for modelling and analysis of concurrency. Such a representation can assist in the refinement of the agent model into a more mechanical or computer-oriented representation and can be used here to shed further light on the agent modelling representation.

Coloured Petri nets (CP-nets) have tokens which can carry a data value of potentially arbitrary complexity, and the data type of a token is referred to as its "colour". Coloured Petri nets defined by Jensen have three essential components:
— a net structure, which is just like that of ordinary Petri nets
— a set of data declarations
— a set of net inscriptions

The declarations define the colours and other components that can be used in the evaluation of expressions and are represented in terms of Standard ML (SML), which is the language employed in the commercial implementation of coloured Petri nets [15]. The inscriptions, also expressed in SML, are expressions which can be attached to a place, a transition, or an arc. When the input arc expressions of a transition are evaluated, the expression variables are bound to the appropriate colours, and the value of the expression must be equal to a multi-set of the colour that is attached to the input place. Transitions can have attached to them additional SML expressions, which are boolean guards. For a CP-net transition to be enabled and fire, it is necessary to examine the possible bindings of all the transition variables (in the guards and on the arc inscriptions); and for each such binding, the guard and all the input arc expressions are evaluated. If the resulting guard evaluates to 'true' and each place has at least as many tokens as are indicated by the evaluation of the input arc expressions, then the transition is enabled. If the enabled transition does fire (for a particular binding), then the output arc expressions are evaluated to determine the number of tokens to be added to the corresponding output places.

The combination of an inscription language and the distributed net representation of Petri nets offers a powerful high-level modelling facility. Hierarchical modelling can be performed by employing "substitution transitions". This mechanism allows a user to replace a single CP-net transition (and its adjacent arcs) with a more complex and detailed CP-net. In this way one can progressively refine a high-level coloured Petri net representation into more detailed models and develop a consistent modelling hierarchy. An agent scheme can then be placed at the top of this hierarchy by modelling it in terms of coloured Petri nets.

At the simplest level of Petri net modelling, the three object types shown in Figure 1 can each be represented as a single Petri net transition connected to an input place and output place, which can then be refined by means of a substitution transition. Figure 4 shows a more detailed CP-net representation of an agent. In the upper left of the figure is the input place, and in the lower right is the output place. The original single transition has now been replaced by the other nodes and arcs shown in the figure. The scope of the declarations for this CP-net is local to this particular agent.

As is true of most complex, dynamic systems, there are several separate process groups that operate concurrently in this model. In the upper left corner of Figure 4, is the process that collects input information and distributes it internally. This operates whenever new input information is available, irrespective of what may be occupying other components of the agent. On the right side of Figure 4, near the bottom, is a set of processes associated with the operation of the internal utilities of the agent. (Note that, with Standard ML, the expression 1\`u1+1\`u2+1\`u3 indicates a multiset comprising singles instances of u1, u2, and u3). For a human this might mean the operation of the arms and legs; for an operating system it might mean the operation of some line printers. The main set of processes of the agent run vertically down the centre of Figure 4.

Associated with each place in Figure 4 is a colour set, which is written in italic script in the figure. The colour set for the "Input" place is *Information*, which is as a record containing two fields, *info* and *mode*. These two fields each have associated colour sets, *info_data* and *info_mode*. The *info* field contains the actual information content, and the *mode* field identifies whether the information record is a routine message or a command. In principle, the *info_mode* colour set could be expanded to cover many different types of information records, such as alarms or questions. The purpose here is to demonstrate the basic possibilities while keeping the diagram as simple as possible.

When the "Input Information" place has a token present, the "Process Info" transition will be enabled. When it fires, it will pass a new token containing just the info part of the original token to either the "Goals" place or the "Info Data Available" place, depending on the mode field of the original Information token. This selection is accomplished by the arc inscriptions shown in the diagram. If the mode of the original token is a command (*mode*=c), the info data will be installed as a new token in the "Goals" place. On the other hand, if the mode of the original Information token is a message (*mode*=m), then the info data will be sent to the "Info Data Available" place by inserting it at the front of a list that was obtained from the "Info Data Available" place. The "Info Data Available" always contains a single token, which is a list of items of type *info_data*. Although this list could be the empty list, this place always has a list and so always has exactly one token in it.

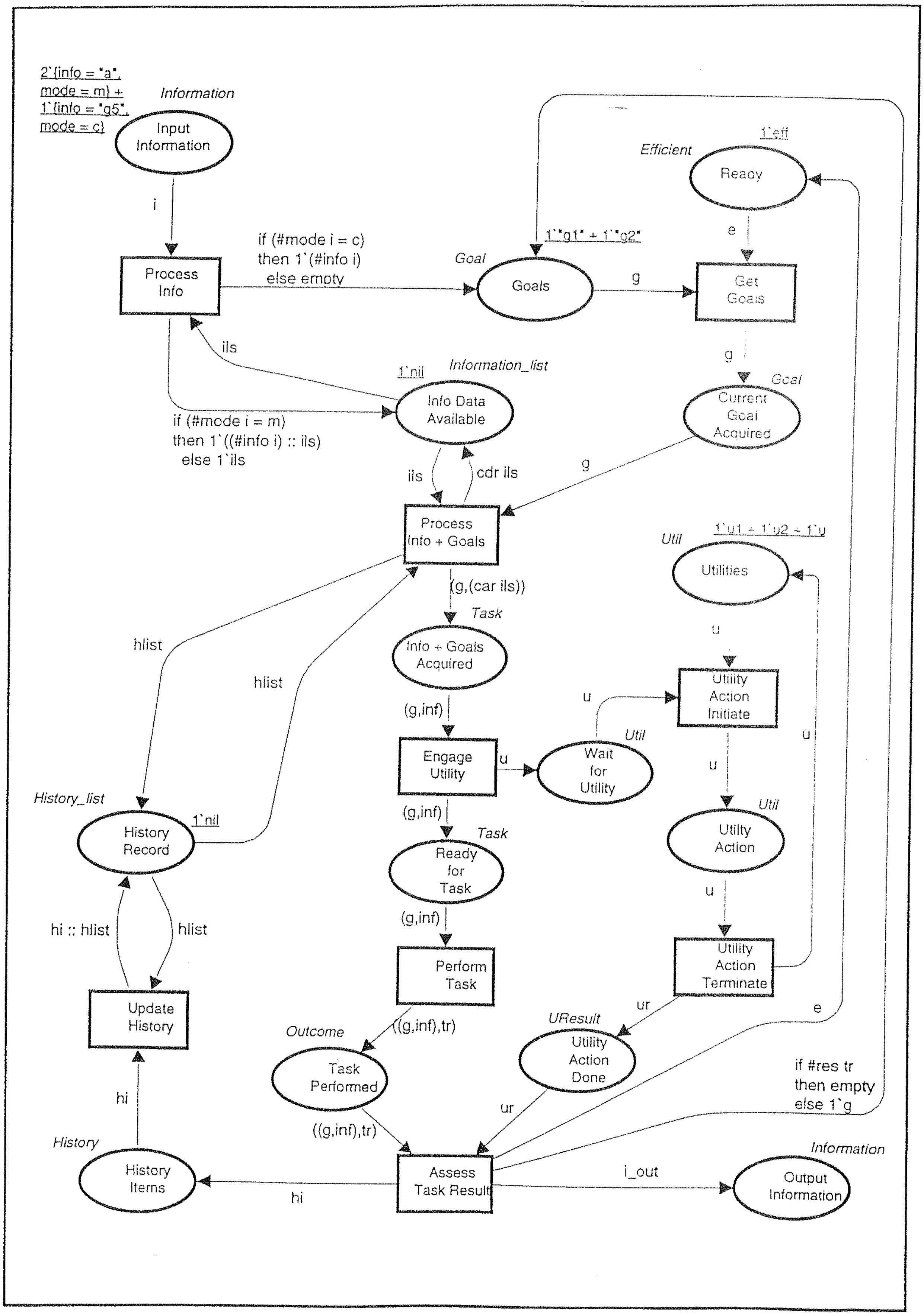

Figure 4: Coloured Petri net representation of an agent

The initial markings of the Petri net places in Figure 4 are indicated by underlined text. The initial marking of the Input Information place contains three tokens: two are of type message ("m") and one is of type command ("c"). The "Goals" place initially has two tokens that identify two goals, "g1" and "g2", where, again, for illustrative purposes, the structures of the goals is kept very simple.

The lower left corner of Figure 4 holds the record-keeping (memory) component of the agent. The process "Assess Task Result" sends a history item to the "History Items" place. The "Update History" transition places this item at the front of the token of type *History_list* that is held in the "History Record" place.

The main sequence of processes (transitions) in this agent model is initiated when a token is present in the "Ready" place in the upper right corner of Figure 4. In principle, there could be several tokens going concurrently through this sequence, but in this illustration we show only a single token in the initial marking. A goal is acquired from the "Goals" place and made available to the "Process Info+Goals" transition. This transition also has available to it the list of information present in the "Info Data Available" place and the list of history items present in the "History Record" place. Both of these lists could be empty, but since a list is always present in each of those two places, the "Process Info+Goals" transition is enabled as soon as there is a goal available in the "Current Goal Acquired" place. The firing of this transition sends a new token containing both the goal and the information to the "Info+Goals Acquired" place.

In light of the token containing the current goal and information that reaches it, the "Engage Utility" process may request a particular utility operation. It puts this request in the "Wait for Utility" place and transfers the goal and information by putting a token into the "Ready for Task" place. The "Perform Task" process (transition) then executes the operation that attempts to achieve the current goal. This key process is the one that will probably be most elaborately refined in subsequent design refinements. When the task is completed, the transition passes a single token containing the goal, the current information, and a task-result item (*tr*) to the "Task Performed" place.

The "Assess Task Result" process examines its input tokens to determine whether the goal was successfully achieved at this stage or not. If not, it sends the goal *g* back to the "Goals" place so that its achievement can be attempted during another sequence. This process also (1) sends information (operations, commands, or messages, to be applied to other objects) to the "Output Information" place, (2) sends a history information to be stored in the "History Items" place, and (3) returns a token to the "Ready" place.

There can be many variations on the basic structure of the agent shown here. For example, a separate set of nodes that model fuel consumption of a stored fuel supply could be included as a subsystem. When the fuel supply runs low, it could trigger a goal (with high Priority) to be installed in the "Goals" place to carry out tasks aimed at replenishing the fuel supply.

Note that the utilities subsystem runs in parallel with the "Perform Task" processes. With reference to the anthropomorphic inspiration of agents, one can imagine the internal utilities of an agent acting like its "arms and legs". One could extend this "mind/body" analogy by viewing the internal utilities subsystem as representing the electronic hardware of a hardware/software system under design: the utilities subsystem would then serve as the hardware "body" of the agent. Since research in hardware/software codesign has identified the early stages of hardware/software system design as the critical area where help is needed [16], the agent representation shown here could serve as an intuitive and flexible modelling platform for the high-level design of hardware/software systems.

The coordination among agents can also be represented in terms of a coloured Petri net. This is shown schematically in Figure 5 with three interacting agents, where a coordinator subnet is responsible for dispatching information appropriately so that an individual agent, say P1, can send a message to another agent, P2. This arrangement does not preclude the specification of synchronisation between interacting agents: that can be accomplished by appropriate inscriptions in the coordinator subnet.

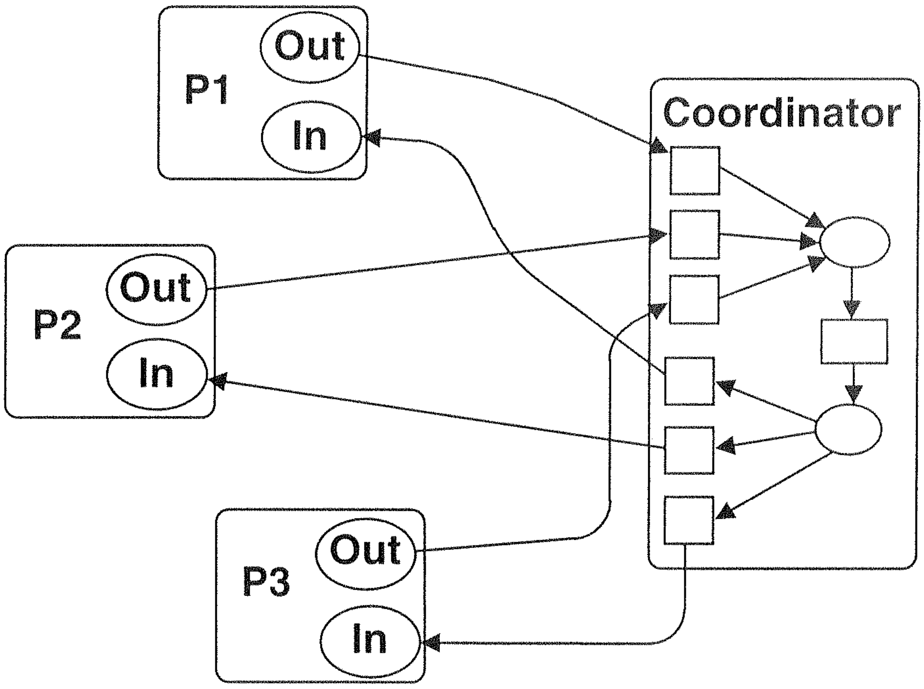

Figure 5: Coordination among three agents

As a result, the entire agent representation can be mapped into the coloured Petri net notation for the purposes of analysis.

4 Conclusions

At the present time the agent framework is implemented internally in terms of coloured Petri nets by means of Design/CPN [15]. Current work is now being devoted to implementing the entire approach using Common Lisp so that the dynamic creation of agents can be more easily facilitated. This will also facilitate the intended extension whereby behavioural nets of agents are encapsulated inside the agent objects. It will then be easier to represent agents in terms of composite-part objects, such as a memory maintenance entity, a goal management entity, etc. Each of these part objects could then have an encapsulated subnet for behavioural simulation and analysis as one of it constituent elements.

References

[1] O. Biberstein and D. Buchs, ``Structured Algebraic Nets with Object-Orientation," Workshop on Object-Oriented Programming and Models of Concurrency '95, June 1995, Turin.

[2] O. Biberstein and D. Buchs, ``Concurrency and Object-Orientation with Structured Algebraic Nets," IS-CORE Workshop, Evry, September 1995, pp. 73-75.

[3] Olivier Biberstein and Didier Buchs, ``An Object Oriented Specification Language based on Hierarchical Petri Nets", IS-CORE Workshop (ESPRIT), Amsterdam, September 1994.

[4] C. A. Lakos, "Object Petri Nets, Definition and Relationship to Coloured Nets", R94-3, Dept. Comp. Science, Networking Research Group, University of Tasmania, April 1994.

[5] R. Bastide and P. Palanque, "Cooperative Objects: a Concurrent Petri Net Based Object-Oriented Language", *IEEE System Man and Cybernetics* 93, Le Touquet, France, October 1993.

[7] K. Jensen, "Coloured Petri Nets: A High Level Language for System Design and Analysis", *Advances in Petri Nets 1990*, Springer-Verlag, Berlin, 1990.

[7] T. Holvoet and P. Verbaeten, "Using Agents for Simulating and Implementing Petri Nets", Dept. of Computer Science, K. U. Leuven, Belgium, http://idefix-45.cs.kuleuven.ac.be/~tom/PNTOX/pntox.html.

[8] M. Merz and W. Lamersdorf, "Agents, Services, and Electronic Markets: How Do They Integrate?", to appear in *IFIP/IEEE International Conference on Distributed Platforms*, Dresden, 1996.

[9] S. Cranefield, P. Gorman, and M. Purvis, "Communicating Agents: An Emerging Approach for Distributed Heterogeneous Systems", *New Zealand Journal of Computing*, 6:1B, August 1995, pp. 337-343.

[10] S. J. S. Cranefield and M. K. Purvis, "Agent-based integration of general-purpose tools", in *Proceedings of the Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management*, (December 1995).

[11] M. Wooldridge and N. R. Jennings, "Intelligent Agents: Theory and Practice", *The Knowledge Engineering Review*, vol. 10, 1995.

[12] B. Grob, *Basic Television and Video Systems*, McGraw-Hill, New Yori, 1984, p. 250.

[13] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer Grahics*, Addison-Wesley, 1990, p. 179.

[14] R. Bastide, "Approaches in Unifying Petri Nets and the Object-Oriented Approach" *Proceedings of the Application and Theory of Petri Nets 1995 Workshop on Object-Oriented Programming and Models of Concurrency*, Troino, Italy, June 1995.

[15] *Design/CPN Version 2.0*, Metasoft, Cambridge, MA, 1993.

[16] M. K. Purvis and D. W. Franke, An Overview of Hardware/Software Codesign", *International Symposium on Circuits and Systems,* San Diego, 1992.