Abstract—As one of the most important artificial intelligence techniques, multi-agent systems are widely applied in many fields. In a multi-agent system, agent interactions are established through exchanging messages by following some interaction protocols. However, as the application domains of MASs are getting more and more complex, some limitations of current multi-agent interaction mechanisms are rising up. To cover some limitations of current agent interaction mechanisms, in this paper, we propose a Coloured Petri Net based approach to enable flexible agent interactions in open environments. This approach adopts Coloured Petri Net techniques to model agent interaction protocols, and separates interaction protocols and agents into two different layers. Through this way, agents can be more independent from a particular multi-agent system. Also, the flexibility and extensibility of multi-agent systems are enhanced.

I. INTRODUCTION

Nowadays, multi-agent system (MAS) is one of the most important concepts in both artificial intelligence (AI) and the mainstream of computer science (CS). A MAS can be considered as a society of agents that work together. In such a multi-agent society, interactions between agents are unavoidable. The interaction between agents occurs when an agent has some intentions and has decided to satisfy these through influencing other agents.

Agent interactions are established through exchanging messages that specify the desired performatives of other agents and declarative representations of the contents of messages. Generally, agents compose their messages in an agent communication language (ACL), then exchange messages with others by following some interaction protocols, which constrain the possible sequences of messages that may occur in agent interactions. In traditional interaction mechanisms, interaction protocols are governed by standards bodies such as FIPA and KQML, or written in immutable documents that will be distributed among agents.

As the application domains of MASs are getting more and more complex, it is increasingly needed for agents to be able to flexibly and robustly communicate each other in a changing and uncertain environment known as an open environment. Towards such environments, the limitations of current MAS interaction mechanisms is rising up. Firstly, most current interaction mechanisms require agents to be hard-coded with interaction protocols. This feature conflicts with the dynamic feature of open environments, and greatly reduces the flexibility of the agents. Secondly, in an open environment, interactions between agents may be influenced by some unexpected factors, such as unexpected message, loss of messages or deviation in the message order. However most current MASs lack error handling mechanisms to deal with these unexpected factors. It increases the interaction risk between in MASs. Finally, many current MASs lack a proper method to represent interaction protocols formally. This feature reduces the extensibility of MASs.

To capture dynamic features of open environments in our research, we hire Coloured Petri Net (CPN) techniques [1] into agent interaction modelings. The CPN is a high-level extension of the Petri Net (PN) [2]. It is a system modelling tool that can provide an appropriate mathematical formalism for the description, construction and analysis of distributed and concurrent systems. CPNs can express a great range of interactions in graphical representations and well-defined semantics, and allow formal analysis and transformations. CPN is considered as one of the best modelling tool for concurrent systems and interactions. By using CPNs, interaction protocols can be separated from agents, and the flexibility, extensibility and robustness of agent interactions can be improved.

In this paper, we propose a CPN based approach for agent interactions. This approach especially improve agent interactions from following aspects:

1) Develop a CPN based generic interaction models that can be adopted by heterogeneous agents from different origins;
2) Separate interaction protocols from agents and enable agents to select and execute CPN based protocols by their demands;
3) Use CPNs to visualise agent interactions for monitoring and debugging purposes.

In this approach, CPN models are used to represent agent interaction protocols. In addition, agents and interaction protocols are separated into two different layers. Through this way, we avoid hard-coding interaction protocols into agents, and allows agents to select interaction protocols according to their demands. This feature enables agents to interact more...
flexibly and make MASs more adaptable for open dynamic apportionments.

The rest of this paper is arranged as follows. We briefly present related concepts of CPNs in Section 2. Then, we introduce some related works that use CPN techniques in agent interactions in Section 3. In Section 4, we present how to use CPN models to represent agent interaction protocols. After that, the CPN based approach for flexible agent interactions is proposed in Section 5. Finally, the conclusion is given in Section 6.

II. PETRI NETS AND COLOURED PETRI NETS

A PN can be formally defined by the four-tuple \(PN = (P, T, A, \mu)\), where \(P = (p_1, p_2, ..., p_n)\) is a set of places, \(T = (t_1, t_2, ..., t_n)\) is a set of transitions, \(A\) is a set of directed arcs that link places and transitions together, and \(\mu\) is an assignment of tokens to the places of a PN. Places of a PN can contain tokens. Token allocation status in places is defined as marking of a PN. A transition can be fired if the token number of all input places is equal to or greater than their arcs’ weights [2]. After a transition is fired, the tokens of its input places will be moved to its output places, and this will cause marking to change.

Fig. 1 (a) shows a simple example of PNs. In this example, \(P = (p_1, p_2, p_3, p_4)\), \(T = (t_1, t_2, t_3)\). Since \(p_1\) and \(p_3\) have one token and other places have no tokens inside, the current marking of the PN is \(\mu = (1, 0, 1, 0)\). Currently, only \(t_3\) can be fired. After \(t_3\) is fired, the marking of the PN will be changed to \((1, 0, 0, 1)\), i.e. one token will be transferred from \(t_3\) to \(t_4\) (see Fig. 1 (b)).

![Fig. 1. An Example of Petri Nets](image)

Tokens of a PN can only have two values: true or false, which indicates if the token exists in the place or not, respectively. The One major difference between PNs and CPNs is that tokens of CPNs are not simple blank markers, but have data associated with them. CPN tokens can represent more meaningful information. Another difference is that arcs and transitions of CPNs can conduct arc functions and guard functions. These functions can control token transferring and transformations during transition firings. Hence, these advantages can be used to capture dynamic behaviours of concurrent systems.

A CPN can be defined by a 9-tuple [3], \(CPN=(\Sigma, P, T, A, F, E, C, G, \mu)\), where \(\Sigma\) is a set of non-empty data-types/color-sets (each colour set is a token data-type of a CPN); \(P\) is a set of places of the CPN; \(T\) is a set of transitions of the CPN; \(A\) is a set of arcs that link transitions and places of the CPN; \(F\) is a set of mapping functions that define from \(A\) into \(P \times T \cup T \times P\); \(C\) is a set of the colour functions that define \(P\) into \(\Sigma\); \(G\) is a set of guard functions of transitions that define token transferring conditions; \(E\) is a set of arc functions that describe the token transformation by passing arcs; \(\mu\) is the initialisation function that defines the initial marking of the CPN.

Tokens of a CPN can be defined in some colour sets, and can carry complex data such as schemes or specifications. These tokens are distinguished from each other according to tokens’ value (colour). Places of CPNs contain multi-sets of tokens. Arcs exiting and entering a place can have an associated constrain function to determine which multi-set elements (tokens) are to be removed or held. Transitions of CPNs are associated with some guard functions that enforce some constraints on tokens. Therefore, by using a CPN to model a concurrent system, the system constrains and data transition conditions can be defined in arc functions and guard functions. In addition, the data flow and transition of the status of a system can be represented by token allocations (markings).

Many CPN developing tools have been successfully developed and widely applied in various areas. In this research, we use CPN/Tools [4] as the CPN model developing tool. CPN/Tools is a tool for editing, simulating and analysing CPNs. It uses an extension of Standard Meta Language (SML), i.e. CPN ML, as the language for declarations and net inscriptions. By using CPN/Tools, the CPN models can have a XML format, which allows it to be interchanged among different platforms. Furthermore, CPN/Tools supports TCP/IP (Transmission Control Protocol/Internet Protocol) connections and makes it possible to link CPN models with other programs (agents). In this paper, all CPN models and diagrams are developed by using CPN/Tools.

III. RELATED WORK

The CPN is originally designed as modelling tools for concurrent systems. A difference between CPNs and some event-based process modeling notations is that CPNs can model both states and events of a system. This feature attracts MAS researchers’ attention to use CPNs to represent interaction states, and transitions between interaction states in a MAS.

Cost [5] proposed the use of CPNs as a model underlying a language for protocol specification by taking the advantages
of CPNs’ great expressive power to support for concurrency. In addition, Cost used CPN ML as the language to formalise interaction models and developed CPN interpreters to execute the CPN models. The use of CPN models in Cost’s work greatly facilitates the development of systems of interacting agents and proves the potential for interpreting CPN models from CPN ML expressions. However, Cost did not describe how to interpret CPN models and how to make agent interact by using the interpreted protocols in detail.

Freire [6] also investigated methods to represent interaction protocols. Differently, Freire used Agent Unified Modelling Language (AUML) to describe interaction protocols and hired XML as the interchange representation format. Comparing the work of Freire with the work Cost, it can be seen that CPNs have more advantages than AUML in two aspects: (1) AUML cannot be converted into a formal language directly, but CPN can be formally described in XML or CPN ML; (2) CPN can be directly used as a simulation tool for agent interactions, but direct simulations cannot be achieved by using AUML.

Nowostawski [7] described a layered approach based on CPNs that can be used for modelling complex, concurrent conversations among agents in a multi-agent system. In this research, agent interactions were described into three layers and represented by corresponding CPN models, respectively. Nowostawski also compared features of different interaction modelling tools, including UML, Deterministic State Automaton, Enhanced Dooley Graph and CPN. As the result, Nowostawski claimed that CPNs have more advantages in modelling complex concurrent interactions than other tools. Nowostawski’s work gave an outline of using CPNs in agent interaction modeling. Purvis et al [8] extended the work of Nowostawski and explored the potential application domains of CPN based interactions models, including environmental emergency systems, E-business, tourism systems and workflow management systems. However, both Nowostawski and Purvis did not describe how to dynamically construct interactions according to CPN models.

Some researchers used CPNs to monitor agent interactions. Poutakidis [9] used the PNs to monitor agent interactions, and generate precise and informative error messages when protocols were not correctly followed by the agents. Gutnik [10] introduced a scalable PN representation of interaction protocols for overhearing agent interactions. Poutakidis and Gutnik’s works demonstrated how to use CPNs to diagnose weaknesses in agent interactions. However, none of them included CPN analysis in the work.

There were also some works on the investigation of protocols’ flexibility, robustness and extensibility. Hutchison [11] presented an example protocol, known as Merchant-Customer protocol, to describe flexibility and robustness of agent interaction protocols in open systems. Ahn [12] suggested a handshaking mechanism for conversation policy agreements that enabled agents to exchange/agree to new conversation policies at runtime.

The current related works show the advantages of CPNs in developing more flexible and robust interaction mechanisms. Most of current research only focus on using CPNs to represent agent interactions. However, beside representations, CPNs can be applied to enhance agent interactions in many other aspects, including interaction simulation and interaction analysis. Also, CPNs can provide generic interaction models for handling activities in agent negotiation and cooperation. In this paper, we focus on discovering more advantages provided by CPNs on improving agent interactions.

IV. CPN Models for Interaction Protocols

To capture dynamic and concurrent features of agent interactions, in our research, we use CPN models to represent agent interaction protocols. Generally, it takes three steps to create CPN models for interaction protocols:

1) Define appropriate token colour sets, places and transitions according to the properties of the interaction protocol;
2) Use directed arcs to link places and transitions together according to the message flow that is defined in the interaction protocol;
3) Define guard and arc functions on related arcs and transitions according to the message transforming rules and the message transferring conditions, which are described in the interaction protocol.

In this section, we will introduce how to define CPN components and construct CPNs to satisfy the requirements of interaction protocol modelling.

A. Using CPN Tokens to Represent Messages

We use CPN tokens to represent messages that are exchanged between agents. To achieve this, a specific data-type (i.e. colour set) is needed for message tokens. Since different applications have different requirements for message structures, here, we give a generic colour set for message tokens, which is shown in Fig. 2. In this colour set, Sender and Receiver(s) indicate the IDs of agents that take part in the interaction. MessageContent contains the message that the agent wants to express. Flags are some boolean or integer parameters that facilitate transition firings.

```
colset MESSAGE =
  Sender x Receiver(s) x MessageContent x Flag(s);
```

Fig. 2. Generic colour set for Message Tokens

B. CPN Places for Interaction Protocols

The states of a interaction are represented by CPN places. For CPNs, each place has an associated type determining the kind of data that the place can contain. Here, the type of a place will determine which kind message can be exchanged in the interaction. In addition, the status of token allocation in places directly determines which transition can be executed at that moment (see Fig. 3).
C. CPN Transitions and Arcs for Interaction Protocols

The interaction policies are carried by transitions and arcs of the CPN. A transition can link with a number of input and output places by directed arcs. A transition is enabled if all of its input places have tokens, and the colors/values of these tokens satisfy constraints that are specified on the arc function and the guard function. A transition can be fired, which means the actions of this transition can occur, when this transition is enabled. When a transition occurs, it consumes all the input tokens as computing parameters and adds new tokens into all of its output places. After a transition occurs, the state (marking) of an interaction has been changed and an interaction will be in a terminal state when there is no enabled or fired transition. Hence, the directed arcs between transitions and places determine token transfer directions in the CPN model. Furthermore, guard functions on the transitions and the arc functions on the arcs will determine which tokens can be transferred through the transition (i.e. fire the transition) and how the token value be changed, respectively (see Fig. 4).

D. CPN Models for Interaction Protocols

After introducing the meanings of CPN components, in this subsection, we introduce how to use CPNs to model agent interaction protocols. Here we take the FIPA Inform protocol as an example to demonstrate how to use CPNs to model agent interaction protocols.

Fig. 5 shows the UML model for the FIPA Inform protocol. From this model, it can be seen that there are two executions in this protocol, which are inform and process. These two executions may lead to the interaction transform among three states, i.e. before inform, informed and processed.

![UML Model for FIPA Inform Protocol](image)

To describe the FIPA inform protocol in Fig. 5 into a CPN model, firstly, we can define four places corresponding to the three possible states of the interaction: a Start place for the start state, a Received place and a Terminated1 place for the informed state, and a Terminated2 place for the processed state. Secondly, two transitions, i.e. the Inform transition and the Process Inform transition, are defined to represent the Inform and Process execution, respectively. Finally, some related guard functions and arc functions are defined to check tokens’ format and modify related flags of tokens. The formal description of the CPN model is shown in Table IV-D. The graphical representation of the CPN model are shown in Fig. 6 to Fig. 8.

<p>| Table I |</p>
<table>
<thead>
<tr>
<th>Formal Description for the CPN Model of FIPA Inform</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPN</strong> = (Σ, P, T, A, N, C, G, E, I)</td>
</tr>
<tr>
<td><strong>P</strong> = {Start, Received, Terminated1, Terminated2}</td>
</tr>
<tr>
<td><strong>T</strong> = {Inform, Process Inform}</td>
</tr>
<tr>
<td><strong>Σ</strong> = {MESSAGE}</td>
</tr>
<tr>
<td><strong>G</strong> = {messageGood(message)}</td>
</tr>
<tr>
<td><strong>E</strong> = {send(message), (n=n+1)}</td>
</tr>
</tbody>
</table>

The changing of states of the interaction can be described as follows:

1) To start the interaction, at least one message token should be assigned to the Start place. If this token satisfies the constraints specified in the guard function MessageGood(), the Inform transition will be enabled. This means the agent is ready to Inform another agent (see Fig. 6).
2) After the Inform transition is fired, a message token will be removed from the Start place, and a new token will be generated in the Received and the Terminated1 place, respectively. At this moment, the system the Process Inform transition is enabled (see Fig. 7).
3) Finally, after the Process Inform transition is fired, the interaction will be terminated, i.e. both of Terminated1 and Terminated2 places have a token (see Fig. 8).

V. A CPN BASED APPROACH FOR AGENT INTERACTION

As introduced in the previous section, an agent interaction protocol can be represented in a CPN model. Through this way, agents can operate interactions through communicating with a CPN model of a particular interaction protocol. In this section, we introduce a CPN based approach that separate agents and interaction protocols in two different layers. In this approach, agents can select their preferred protocols first before they operate interactions.

A. A Layered Architecture for Agent Interactions

The CPN based approach adopts a layered architecture, which is shown in Fig. 9. This architecture has three layers: (1) the protocol layer, (2) the media layer and (3) the agent layer. The detailed descriptions of the three layers are as following:

1) The Protocol Layer: This layer is the top layer of the architecture. It conducts interaction protocols that are used in agent interactions. The interaction protocols are described in CPN models, and are independent from agents. The CPN models have some communication ports to accept and send data with outside programs. These ports allows the communication between agents and interaction protocols.

2) The Agent Layer: Agents are located in the bottom layer of the architecture. Since interaction protocols are conducted in another layer (i.e. the protocol layer), agents do not need to be hard-coded with interaction protocols. To achieve interactions, agents only need to compose their messages in appropriate formats and send messages to the protocol layer through communication ports.

3) The Media Layer: The media layer is between the protocol layer and the agent layer. It contains a number of facilitators that assist agent interactions. For different applications, the media layer can include various facilitators. However, two kinds of facilitators are necessary for most applications, i.e. the Protocol Register (PR) and the Port Manager (PM). The PR provides agents formal descriptions and constrains of interaction protocols in the Protocol Layer. The PM manages and allocates communication ports for agent interactions.

The layered architecture splits interaction protocols from agents to an individual layer. To interact with others, agents need to select a proper interaction protocol according to the information provided by the PR. Then, agents send a request to the PM to indicate their preferred protocol. Receiving the request from agents, the PM will allocate corresponding port(s) for agent interactions, and agents are able to use that/those
port(s) to send and receive message tokens. Thus, to operate an interaction, agents only need to create and send message tokens to the appropriate communication port that links with the CPN model. After that, the CPN model will be responsible to transfer messages in the direction and order that are defined in the interaction protocol.

B. Communication Interface between Protocols and Agents

By dividing agents and interaction protocols into two layers, agent interactions will be mediated by the protocol layer. As a result, an agent interaction becomes to a “token playing game”. Agents can achieve and control interactions through placing/removing tokens to/from corresponding CPN places in the protocol layer. Most CPN developing tools provide communication infrastructures to allow communications between CPN models and external programs/processes. In our work, we use Comms/CPN [13], which is the communication infrastructure provided by CPN/Tools, to establish the communication between agents and the protocol layer.

Comms/CPN can accept connections from various applications that are composed in different languages, such as Java, C++, applications, etc. These applications communicate with Comms/CPN through TCP/IP sockets. In this research, agents are developed by using Java. Hence we integrate a Java/CPN interface with agents to realise communications between agent interactions will be mediated by the protocol layer. As a result, an agent interaction becomes to a “token playing game”. Agents can achieve and control interactions through placing/removing tokens to/from corresponding CPN places in the protocol layer. Most CPN developing tools provide communication infrastructures to allow communications between CPN models and external programs/processes. In our work, we use Comms/CPN [13], which is the communication infrastructure provided by CPN/Tools, to establish the communication between agents and the protocol layer.

Comms/CPN can accept connections from various applications that are composed in different languages, such as Java, C++, applications, etc. These applications communicate with Comms/CPN through TCP/IP sockets. In this research, agents are developed by using Java. Hence we integrate a Java/CPN interface with agents to realise communications between agents and Comms/CPN. The Java/CPN interface is shown in Fig. 10. To communicate with the protocol layer, an agent uses the connection method to connect establish the connection with the Comms/CPN. The connection method takes a host name and port number as arguments. The host name and port number is corresponding to the CPN model (i.e. interaction protocol) that the agent want to adopt. Once the connection has been established (i.e. the socket is opened), input and output streams are extracted from the socket to allow the agent sends/receives messages to/from the CPN model by using the accept and send methods. Finally, the agent uses the disconnect method to close the input/output stream and the socket, so that the connection between the agent and the protocol layer is disconnected.

```java
public interface JavaCPNInterface {
    public void connect(String hostName, int port) throws IOException;
    public void accept(int port) throws IOException;
    public void send(Byte Array input, OutputStream stream) throws IOException;
    public byte[] receive() throws IOException;
    public void disconnect() throws IOException;
}
```

Fig. 10. Java/CPN Interface

VI. CONCLUSION

In conclusion, to cover some limitations of current agent interaction mechanisms, in this paper, we propose a CPN based approach for agent interactions. In this approach, we separate interaction protocols from agents by using CPN models to represent interaction protocols. In addition, the layered architecture of the approach allows agents to flexibly interact each other by selecting suitable protocols. Benefitted from advantages of CPN techniques, the approach enhance the flexibility and extensibility of MASs.

REFERENCES