Executing explicitly represented protocols

Joaquim Freire
Dept. Information Sciences and Technologies of ISCTE
Av. das Forças Armadas
Edifício ISCTE, 1600, Lisboa, Portugal
Joaquim.Freire@inst-informatica.pt

Luis Botelho
Dept. Information Sciences and Technologies of ISCTE
Av. das Forças Armadas
Edifício ISCTE, 1600 Lisboa, Portugal
Luis.Botelho@iscte.pt

ABSTRACT

This paper describes an approach that enables agents to execute any interaction protocol that can be expressed in the proposed XML representation. Therefore, agents will be capable of executing any protocol representation received in run-time. In our approach, explicitly represented protocols are converted in an agent internal structure more suitable for being processed than XML. The internal representation of the protocol is converted to a set of production rules that control the behaviour of the agent according to the protocol. The used XML representation, which closely mirrors AUML protocol diagrams, is defined by an XML Schema.

The proposal defines a protocol independent interface between the agent private decision processes and the protocol execution process through which the agent can inform the protocol execution process of its decisions regarding protocol alternative courses of action.

Workshop Topics and Area Keywords

Agent communication: protocols

Keywords

Interaction protocols; Protocol explicit representation; Protocol execution; Open multi-agent systems

1. INTRODUCTION

One key issue in open agent systems such as the Agentcities network [14] is the agent ability to participate in interactions following previously unpredicted conversation patterns. If the agents in an agent society have the capability to follow any protocol, the society becomes much more flexible and intelligent than if the agents could only follow a fixed number of previously known interaction protocols.

However, currently existing agents are capable of executing only a fixed predefined set of known protocols because interaction protocols are implicitly embedded in the agent code, as is the case in [3][11], and also in Jade [2] and FIPA-OS [10] agents. The way to circumvent this limitation is to build the agents on top of a generic mechanism capable of executing explicitly represented interaction protocols. This paper contributes to this endeavour by (i) presenting an approach to explicitly represent interaction protocols, (ii) providing a general-purpose mechanism that executes the represented protocols, and (iii) proposing an interface between the agent internal decision process and the protocol execution mechanism.

Given this mechanism, agent interaction goes as follows. The agent receives a message initiating a specific protocol. If the agent has already loaded the specified protocol, it may execute it. Otherwise, the agent retrieves the specified interaction protocol from its protocol database, loads it and may decide to execute it. If the specified protocol does not exist in the agent protocol database, the agent will ask a protocol server agent to send it the representation of the protocol and may decide to execute it. Then it stores the protocol for future use.

Section 2 describes the presented approach and section 3 presents conclusions and directions for future investigation.

2. PROTOCOL REPRESENTATION AND EXECUTION

Protocols are explicitly represented in XML [12] following a proposed XML Schema [13]. When the agent receives a message specifying a specific protocol, the agent converts the XML representation of the protocol into an internal Java object. Then, it converts the generated Java object into a set of production rules that can be used to control the agent behaviour according to the specified protocol. These rules representing the operational aspect of the protocol interact with the agent internal decision processes through a proposed interface.

2.1 Protocol representation

In our proposal, interaction protocols are represented in XML, following its graphical description in AUML (Agent Unified Modelling Language) [1], which has often been used for protocol description, for instance in the FIPA Specifications [6]. Since AUML is a graphical language, it is not practical for computational processing hence the use of XML.

We have chosen XML because there are tools that easily convert XML documents into programming language structures and assist editing XML documents, and because it is programming language-neutral.

Figure 1 depicts the AUML specification of a simple interaction protocol called FIPA-Request. FIPA-request has two roles: the Initiator, which initiates the protocol, and the Participant, which responds to the initial message. For each role, the protocol specifies the actions it may perform in each circumstance. For instance, in the initial state, when the Participant receives the *request* message, it may reply using the *not-understood* message, or the *refuse* message, or the *agree* message. If the Participant agrees, that is, if it chooses to send the *agree* message, the protocol state becomes the *agreed state*. In the agreed state, the Participant may send the *failure* message, the *inform-done* message, or the *inform-ref* message. The specification of the

protocol says nothing about the agent internal decision processes that lead the agent to choose one of the several possible alternatives in each protocol state.

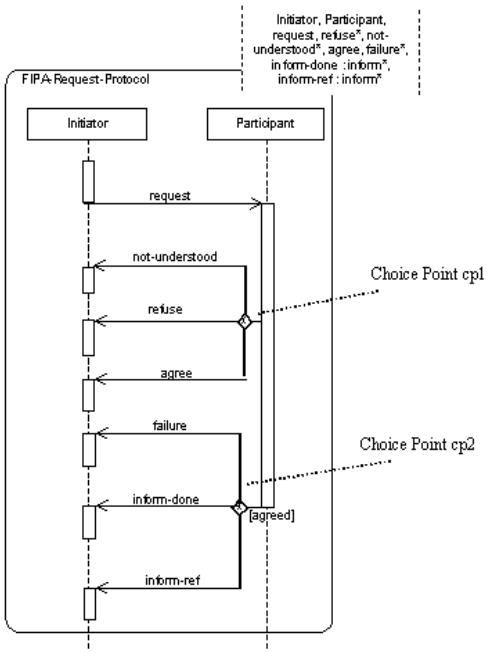

Figure 1. FIPA-Request Protocol

```
<schema>
  <complexType name = "Protocol">
    <all>
      <element name = "name" type = "string" />
      <element name = "role_spec" type = "ProtocolRoleSpec"
           minOccurs = "2" maxOccurs = "unbounded"/>
    </all>
  </complexType>
  <complexType name = "ProtocolRoleSpec">
    <all>
      <element name = "role" type = "string"/>
      <element name = "condition_action"
                type = "ConditionAction"
                minOccurs = "1" maxOccurs = "unbounded"/>
    </all>
  </complexType>
   <complexType name = "ConditionAction">
      <sequence>
        <element name = "condition" type = "Propositon"/>
        <element name = "action" type = "ActionTerm"/>
      </sequence>
   </complexType>
</schema>
```

Figure 2. Protocol XML Schema

In our proposal, the XML Schema used to define the XML protocol representation is composed by the protocol name and by a set of protocol specifications, one for each role. Each role protocol specification is a set of condition-action pairs (Figure 2).

Figure 3 shows the first condition-action pair of the specification pertaining to the participant role of the FIPA-request protocol.

```
<ConditionAction>
  <condition>
    <AtomicProposition>
      <received> request </received>
    </AtomicProposition>
  </condition>
  <action>
    <ActionAlternative>
      <SimpleAction>
        <ActionName> not-understood </ActionName>
      </SimpleAction>
      <SimpleAction>
        <ActionName> refuse </ActionName>
      </SimpleAction>
      <ActionSequence>
        <SimpleAction>
          <ActionName> agree </ActionName>
        </SimpleAction>
        <SimpleAction>
          <ActionName>assert</ActionName>
          <argument> agreed_state </argument>
        </SimpleAction>
      </ActionSequence>
    </ActionAlternative>
  </action>
</ConditionAction>
```

Figure 3. XML Representation of part of the FIPA-Request protocol

We used Castor [4] to create Java classes from XML Schemas. A possible alternative to using XML would be DAML-S [5]. However, in order to use the existing tools that convert XML to Java, we would need the XML Schema (or the DTD) for DAML-S, which is not available. Besides, the DAML project acknowledges the fact that it is not yet clear how to represent interaction protocols using DAML-S. Finally, the main point of the paper is not the concrete syntax for representing protocols. The main point is the process by which explicitly represented protocols are converted into executable code.

Another possible alternative would be XMI [9] because the AUML representation of interaction protocols is very similar to UML sequence diagrams. However, AUML is not exactly the same as UML, therefore we would have to adapt XMI.

2.2 From protocol representation to agent control

The idea is to dynamically generate executable code that causes the agent to comply with the desired protocol. Production rules are a suitable candidate because they can easily represent the part of the selected protocol pertaining to each involved agent, and because they can be created and executed in run-time.

The algorithm that translates the relevant part of the protocol to a set of rules is straightforward: the condition from the protocol condition-action pair is mapped into the rule condition, and the action from the protocol condition-action pair is mapped into the rule action. Even though the basic idea is simple, some details are worth noting.

In software engineering terms, interaction protocols are re-entrant control structures. This means that the same protocol may be governing different simultaneous agent interactions. The same agent may be involved in an instance of the FIPA-request protocol with agent A, and in another instance of the same protocol with agent B. However, the concrete conditions and actions pertaining to the first instance cannot be confused with those of the second instance. In order to circumvent this problem, the rules generated from the protocol explicit representation mention the protocol instance so that they may be used in different instances of the same conversation-pattern.

Another important aspect is that the rules generated from the protocol representation must ensure that the alternative courses of action pertaining to a given choice point in the protocol are not possibilities in a different choice point in the protocol. The alternatives available in choice point cp1 in the FIPA-Request protocol are *not-understood*, *refuse*, and *agree*, whereas the alternatives in choice point cp2 are *failure*, *inform-ref* and *inform-done*. Moreover, the alternatives in cp2 are available to the agent only if it chooses the *agree* alternative in choice point cp1. In order to deal with this problem, we had to use the notions of protocol state and protocol state transition in the protocol representation.

Each set of alternative courses of action is associated to a particular protocol state. Some alternative courses of action explicitly cause state transitions in the protocol.

In terms of production rules, the conditions of rules representing alternatives pertaining to a certain protocol state explicitly mention the protocol-state, which is represented by a proposition, using predicate *protocol_state/2*. The action-part of rules that cause protocol state transitions, explicitly remove the current protocol state and assert a new protocol state.

The following rule shows how the protocol-state is explicitly mentioned in the rule conditions and also how it is changed in the action part of the rule.

```
If   protocol_instance(fipa-request,
     ?ProtocolInstanceID) and
     protocol_state(?ProtocolInstanceID,
     ?PState) and
     selected_action(?ProtocolInstanceID,
     initial_state, agree(?Sender,
     ?Receiver, ?Content, ?ConverstionID))
Then agree(?Sender, ?Receiver, ?Content,
     ?ConversationID),
     retract(protocol_state(?ProtocolInstanc
     eID, ?PState))
     assert(protocol_state(?ProtocolInstance
     ID, agreed_state))
```

The above rule says that if an agent is executing the FIPA-request protocol in a given state, and it has decided to agree with the received request, then it sends the *agree* message and the state of the protocol changes to *agreed-state*. Quotation marks in the rule introduce variables.

The most difficult aspect of the translation algorithm is due to the need of generating specific rules from an abstract protocol, which does not represent the specific messages involved. In the case of the FIPA-request, the protocol specification (Figure 1) does not represent the complete content of the *request*, *not-understood*, *refuse*, *agree*, *failure*, *inform-ref* and *inform-done* messages involved. However the agent control rules deal with specific messages. For instance, the rules generated from the protocol representation must specify the reasons and the conditions that must be sent in the contents of the *not-understood*, *refuse*, *failure* and *agree* messages involved. Those reasons and conditions cannot be contained in the protocol specification just because they can only be determined by specific agents in the specific occasions in which the protocol is being executed.

The following rule shows how protocol abstract specifications of received and sent messages must be mapped into fully detailed messages.

```
If   protocol_instance(fipa-request,
     ?ProtocolInstanceID) and
     protocol_state(?ProtocolInstanceID,
     initial_state) and
     received(?ProtocolInstanceID,
     request(?Initr, ?Particip, ?Action,
     ?ConvID)) and not
     alternative_actions(?ProtocolInstanceID
     , initial_state, _)
Then
     assert(alternative_actions(?ProtocolIns
     tanceID, initial_state,
     [agree(?Particip, ?Initr, (?Action,
     ?Condition), ?ConvID),
     not-undertsood(?Particip, ?Initr,
     (?Action, ?Reason1), ?ConvID),
     refuse(?Particip, ?Initr, (?Action,
     ?Reason2), ?ConvID)]) )
```

The above rule says that if the FIPA-request protocol is being used, and its internal state is the initial state, and the received message is <initiator, request(participant, action)>, then the participant will have three available alternative courses of action: *agree*, *not-understood* and *refuse*. The contents of all these alternatives contain some unspecified slots. *agree* contains an unspecified condition, *not-understood* and *refuse* contain unspecified reasons.

When the Participant selects one of the alternative actions it fully instantiates the unfilled slots in the choice. For instance, if the Participant agent chooses to agree, it must instantiate the condition part of the message content.

Since our software is being developed in Java, we have chosen the JESS language (Java Expert System Shell) [7] to implement the production rules that represent the relevant part of the protocol.

2.3 Interface between the protocol and the agent decision processes

Protocols specify that, in given protocol states, the agent has a set of available alternative courses of action but it does not specify the agent internal decision process enabling it to choose amongst the possible alternatives. It is necessary to define a protocol independent interface between the protocol and the agent decision processes. This interface must allow agent designers to create decision processes capable of interacting with the protocol execution process. On one hand, the interface must allow the protocol execution process to inform the agent internal decision processes of the currently available alternative courses of action. On the other hand, the agent internal decision processes can

inform the protocol execution process of which alternative course of action was chosen.

The most important aspects of the interface between the protocol execution process and the agent internal decision processes are the predicate *alternative_actions/3* and the action *choose_alternative/3*. Through the predicate *alternative_actions/3*, the protocol execution process informs the agent internal decision processes of the available alternative courses of action. *choose_alternative/3* is an action used by the agent internal decision process to inform the protocol execution process of the decision regarding the selected alternative course of action.

`alternative_actions(ProtocolInstanceID, ProtocolState, AlternativeActions)`: in state *ProtocolState* of the protocol *ProtocolInstanceID*, the agent must choose amongst the alternative actions contained in the set *AlternativeActions*.

`choose_alternative(ProtocolInstanceID, ProtocolSate, Action)`: the agent internal decision process informs the protocol execution process that the action *Action* has been chosen in the state *ProtocolState* of the protocol *ProtocolInstanceID*. When this action is executed, the proposition `alternative_actions(ProtocolInstanceID, ProtocolState, AlternativeActions)` is removed and the proposition `chosen_alternative(ProtocolInstanceID, ProtocolState, Action)` is created. This way, the rules that control the protocol execution will know that the choice point has been overcome by an agent decision.

3. CONCLUSIONS AND FUTURE WORK

Although interaction protocols based agents are not the most sophisticated agents, they are an easy approach to agent engineering. The approach presented in this paper greatly expands the protocol approach because it enables the agent to understand and follow any protocol that can be written in the selected formalism. Of course this does not lead to fully autonomous agents such as those capable of planning their course of action.

Even though the protocol based approach to agent engineering is not the most sophisticated one, it is not less true that higher-level intelligence may emerge out of the social interaction of relatively simple agents [8]. Our approach contributes for broadening the range and complexity of interactions in an agent society.

In the proposed approach, the semantics of the protocol is implicitly represented by the production rules generated from the protocol representation. However, some doubts may arise with respect to the semantics of the messages in the protocol. In the current state of our work, it is assumed that the agent knows the semantics of those messages. In the case of FIPA ACL, the semantics associated with a specific message or communicative act are defined externally by FIPA. An interesting possibility would be to use a commonly accepted way of specifying the semantics of communicative acts, in order to decouple agents from specific communication languages.

Two other alleys of future investigation are the autonomous dynamic creation of new more complex protocols from simpler existing ones; and the creation of decision mechanisms to allow the agents to negotiate the protocol to be used.

4. ACKNOWLEDGEMENTS

The research described in this paper is partly supported by the EC project Agentcities.RTD, reference IST-2000-28385, and partly by UNIDE/ISCTE. The opinions expressed in this paper are those of the authors and are not necessarily those of the Agentcities.RTD partners.

5. REFERENCES

[1] Bauer, B.; Müller, J.P.; and Odell, J. Agent UML: A Formalism for Specifying Multiagent Interaction. In Agent-Oriented Software Engineering, Paolo Ciancarini and Michael Wooldridge eds., Springer, Berlin, 2001.

[2] Bellifemine, F.; Poggi, A.; and Rimassa, G. JADE - A FIPA-compliant agent framework". CSELT internal technical report, partially available in Proc. of the Fourth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology , 1999

[3] Botelho, L.M. A Control Structure for Agent Interaction". In Proceedings of the IEEE Intelligent Vehicle Symposium . 2000

[4] Castor Software. http://castor.exolab.org. 2002

[5] DARPA Agent Markup Language. DAML-S 0.6 Draft Release. http://www.daml.org/services/daml-s/2001

[6] Foundation for Intelligent Physical Agents. FIPA Interaction Protocol Library Specification. Document XC00025E. http://www.fipa.org. 2001

[7] Friedman-Hill, E.J. Jess, The Expert System Shell for the Java Platform. Distributed Computing Systems, Sandia National Laboratories, Livermore, CA. Technical Report SAND98-8206. http://herzberg.ca.sandia.gov/jess/. 2002

[8] Minsky, M, The Society of Mind. Simon and Shuster, N.Y. 1985

[9] Object Management Group. OMG XML Metadata Interchange (XMI) Specification. Version 1.2. Document formal/02-01-01.http://www.omg.org/cgi-bin/doc?formal/2002-01-01. 2002

[10] Poslad, S.; Buckle, P.; and Hadingham, R. The FIPA-OS agent platform: Open Source for Open Standards. In Proc. of the Fifth International Conference and Exhibition on The Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM2000). 2000

[11] Singh, M.P. Developing formal specifications to coordinate heterogeneous autonomous agents. In Proc. of the International Conference of MultiAgent Systems . 1998

[12] World Wide Web Consortium. Extensible Markup Language (XML) 1.0 (Second Edition http://www.w3.org/TR/2000/REC-xml-20001006). 2000.

[13] World Wide Web Consortium. XML Schema Part 0: Primer. http://www.w3.org/TR/xmlschema-0/ 2001.

[14] Willmott, S.; Dale, J.; Burg, B.; Charlton, P; and O'Brien, P. Agentcities: a worldwide open agent network". Agentlink News. 8:13-15, 2001