A Language for Exchanging Agent UML Protocol Diagrams

Marc-Philippe Huget

Agent ART Group
University of Liverpool
Liverpool L69 7ZF
United Kingdom
M.P.Huget@csc.liv.ac.uk

Abstract. For several years, interaction protocol designers have a new formalism which takes into account multiagent system features: autonomy, cooperation, etc. This formalism is called Agent UML [6]. For the moment, designers can describe protocols with the Agent UML protocol diagrams but they do not have a textual language in order to exchange protocols or to check properties on them. The aim of this paper is to provide such a language. This language is called AXF (Agent UML eXchange Format) and is structured as an XML file. This paper presents the syntax of this language and applies AXF to the example of the English Auction Protocol.

This paper is published as the technical report ULCS-02-009 from the department of computer science, University of Liverpool. Version 2002/03/01

1 Introduction

Multiagent system designers need formalisms or modeling languages so as to represent, among others, agents, interactions between agents, roles and collaborations. Several approaches exist: from logic approaches to software engineering modeling languages via formal description techniques used in distributed systems. For several years, a new approach is emerged which takes into account agents' characteristics. This approach is based on the modeling language UML which is authority on software engineering. The main modeling language of this new approach is Agent UML [2]. Readers can consult [5] for a description of several UML-based modeling languages. Since 2000, Agent UML supersedes the FIPA's proposal called UAML [4]. As Bauer et al. quoted it [3], Agent UML is based on UML for two reasons: (1) people who work in multiagent systems usually come from software engineering and UML is the most established language in software engineering. So, designers do not have to learn a new language. (2) several tools are available for UML so designers can use them for describing their diagrams.

In this paper, we are focusing on protocol diagrams which allow to represent interaction protocols. Several examples of such protocol diagrams are available in the following papers: [2] [7] [5].

Unlike some formal description techniques like SDL and Petri nets, Agent UML does not include both a graphical modeling language and a textual language. This drawback can slow down the adoption of Agent UML since it is difficult to exchange protocols. Moreover, designers cannot use different tools since it does not exist a common interchange format.

The aim of this paper is to present a textual language for storing Agent UML protocol diagrams. We propose the name of AXF (Agent UML eXchange Format) for this language. The structure of a file in AXF is based on XML tags. A similar approach exists for UML and is based on XML as well. This approach is called XMI (XML Metadata Interchange) [8]. We do not use it since sequence diagrams for UML and protocol diagrams for Agent UML do not share the same information.

The remainder of the paper is defined as follows. Section 2 gives the syntax of AXF. Section 3 presents the different features of protocol diagrams and describes how they are translated into AXF. The example of the English Auction Protocol is given in Section 4. Section 5 concludes the paper and describes future work.

2 The Agent UML eXchange Format (AXF) Syntax

The syntax of AXF (Agent UML eXchange Format) is given below through a BNF grammar. Further explanations on the content could be found in Section 3.

```
axf ::= preamble roles body
preamble ::=
             | name
             | name author
             | name author date
             | name date
name ::= "<name>" IDEN "</name>"
author ::= "<author>" IDEN "</author>"
date ::= "<date>" IDEN "</date>"
roles ::= "<roles>" role body_roles "</roles>"
body_roles ::=   /* empty */
               | role body_roles
role ::= "<role>" body_role "</role>"
body_role ::=   instance "/" name_role ":" class
              | instance "/" name_role
              | instance
              | name_role ":" class
              | name_role
              | class
```

```
20 instance ::= "<instance>" IDEN "</instance>"
   name_role ::= "<name-role>" IDEN "</name-role>"
   class ::= "<class>" IDEN "</class>"
   body ::=   /* empty */
            | protocol body
25 protocol ::=
       "<protocol-diagram>" body_protocol "</protocol_diagram>"
   body_protocol ::=    /* empty */
                      | state body_protocol
   state ::=   "<state>" NUMERIC body_state "</state>"
30           | "<state>" NUMERIC nested guards "</state>"
             | "<state>" NUMERIC interleaved "</state>"
   body_state ::=   message
                  | and_message
                  | xor_message
35                | or_message
   nested ::= "<nested>" IDEN "</nested>"
   guards ::=
              | "<guards>" STRING "</guards>"
   interleaved ::= "<interleaved>" IDEN "</interleaved>"
40 message ::= "<message>" body_message "</message>"
   body_message ::= from to ca card cond type comment rep next_state
   from ::= "<from>" role "</from>"
   to ::= "<to>" role "</to>"
   ca ::= "<CA>" STRING "</CA>"
45 card ::=   /* empty */
            | "<cardinality>" NUMERIC NUMERIC "</cardinality>"
   cond ::=   /* empty */
            | "<conditions>" STRING "</conditions>"
   type ::=   "<type>" "asynchronous" "</type>"
50          | "<type>" "synchronous" "</type>"
            | "<type>" "delay" "</type>"
   comment ::=   /* empty */
               | "<comment>" STRING "</comment>"
   rep ::= "<repetition>" NUMERIC NUMERIC "<repetition>"
55 next_state ::= "<next-state>" NUMERIC "</next-state>"
   and_message ::= "<and-message>" body "</and-message>"
   body ::= "<branch>" body_state "</branch>" card cond comment rep
   xor_message ::= "<xor-message>" body "</xor-message>"
   or_message ::= "<or-message>" body "</or-message>"
```

An AXF file is composed of three parts: (1) the preamble (line 2) which contains information about the file: the filename, the author and the date of creation, (2) the roles (line 10) and (3) the protocol diagrams (line 25).

3 A Description of Agent UML Diagrams

Several notions are encompassed in protocol diagrams: agents or agents' roles, messages, constraints on messages and the protocol unfolding. All these notions are explained in this section.

Agents and Agents' Roles

Agents on protocol diagrams can be represented by different manners: (1) they are represented by their identity called *instance* in Agent UML, (2) they are represented by their role in the protocol. For instance, in auction protocols, one has usually two roles: the *Seller* and the participants called *Auctioneer*. (3) finally, they can be represented by both their identity and their role.

The agent class can be given for completing the definition of the roles. Using roles reduce the size of the diagram. Actually, one does not need several identities but just one role if they all share the same role.

Agents are described on protocol diagram by a box and a text defined as follows: instance "/" role ":" class. For instance, for auction protocols, one could have Smith/Auctioneer. Figure 1 gives two roles: *Initiator* and *Participant*. The vertical bar specifies messages which are sent from or are sent to this role.

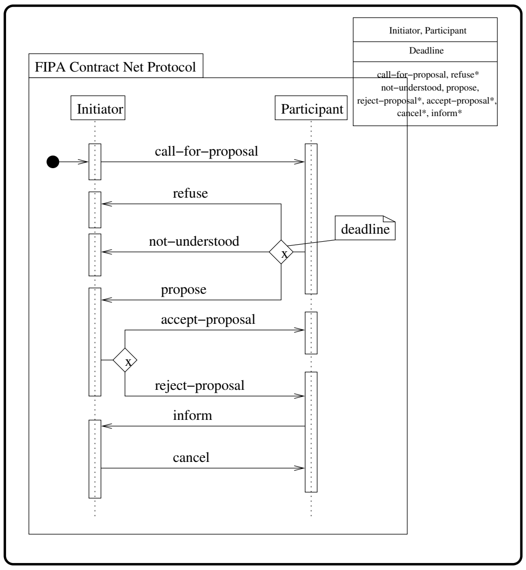

Fig. 1. Contract Net Protocol in Agent UML

Roles are always defined at the beginning of the file in AXF files (line 10 of the grammar). Roles are written between the couple of tags `<roles>` and

</roles>. A specific role is specified between the couple of tags <role> and </role>. In order to describe a role, one needs to provide the instance (between the couple of tags <instance> and </instance>), the role name (between the couple of tags <name-role> and </name-role>) and the class name (between the couple of tags <class> and </class>. For instance:

```
<roles>
   <role>
      <instance>UML-Airlines</instance>
      <name-role>Auctioneer</name-role>
      <class>Seller</class>
   </role>
</roles>
```

Lifelines

Agents are represented by a lifeline which corresponds to their participation in the protocol. When a lifeline is created for a role, this role becomes active for the protocol. This lifeline is present as long as the role has actions in the protocol. Lifelines are defined by a vertical bar drawn on the vertical bar of roles or linked to the vertical bar (see Figure 4 with the messages *not-understood*). Lifelines can be splitted or merged when several alternatives are available in the protocol or several paths are merged in one. For instance, it is the case on Figure 1 where several answers are available: *propose*, *refuse* or *not-understood*. Lifelines are not directly represented within AXF files.

Agent UML Connectors

Interaction protocol designers have three logical connectors for describing alternatives and choices (see Figure 2). The connector AND (see Figure 2a) means that messages have to be sent concurrently. The connectors OR (see Figure 2b) and XOR (see Figure 2c) mean that a choice between several messages has to be done. The connector OR means that zero or several messages can be chosen and the connector XOR means that one message has to be chosen. The term CA refers to communicative acts [1].

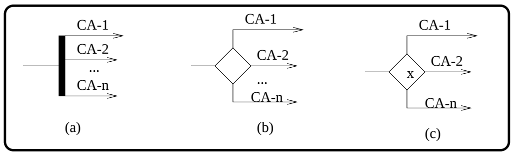

Fig. 2. Agent UML Connectors

These three connectors are described in AXF files by the couple of tags `<and-message>...</and-message>`, `<or-message>...</or-message>` and `<xor-message>...</xor-message>`. Messages for these connectors are given within the couple of tags `<branch>` and `</branch>`. Messages are either simple messages or several messages using the connectors AND, OR or XOR.

Various Notions

Several other notions are present in protocol diagrams:

Conditions: It is possible to put some conditions on sending messages. For instance, for the English auction protocol (see Figure 4), a condition is given for the messages *not-understood*. The variable m must have a value greater than 0 if one wants to use this message. Conditions are given inside curly brackets.
In AXF files, conditions are given between the tags `<conditions>` and the tag `</conditions>`.

Cardinality: Designers have to have the ability to send several copies of the same message (for advertising the opening of an auction for instance). In protocol diagrams, the multiple sending of messages has the meaning of multicast. Two numbers are defined: the first one is close to the vertical bar of the role which sends the message. The number is 1. The second number is close to the vertical bar of the role which receives the message. The value corresponds to the number of copies.
In AXF files, the cardinality is defined between the tags `<cardinality>` and `</cardinality>`. For instance:

```
<cardinality> 1 n </cardinality>
```

This cardinality means n messages are sending.

Type of message: Normally, messages are sent asynchronously (see Figure 3a). It is also possible to sent messages synchronously (see Figure 3b). The last possibility is when messages are not received instantaneously but after some delay (see Figure 3c).

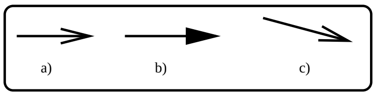

Fig. 3. Type of Messages in Agent UML

In AXF files, the type of message is described as the following:
- synchronous: `<type>synchronous</type>`
- asynchronous: `<type>asynchronous</type>`

– delay: `<type>delay</type>`

Comments: It is possible to insert some comments on diagrams, for instance *start cfp time* on the diagram of the Figure 4.
In AXF files, comments are given between the tags <*comment*> and `</comment>`.

Repetition: One example of repetition is given on Figure 4 for the message requesting the payment. The curly brackets contain two numbers which represent how many times the message has to be sent. Here, the values range from 0 to 1 since the current price might be less than the reserved price, thus no payment is requested.
In AXF files, repetitions are given between the tags <*repetition*> and `</repetition>`.

Nested and interleaved protocols: Since protocols are reusable, it is possible to link protocols together. Two solutions are available in Agent UML protocol diagram: nested protocols and interleaved protocols. Nested protocols correspond to protocols which are within another protocols. This case is used when one needs to repeat several times a protocol. Nested protocols continue while conditions hold.
Interleaved protocols correspond to protocols which are called during the execution of a second protocol. Bauer et al. give the example of an auction participant who requests some information about her/his bank account [3].
Nested and interleaved protocols are distinguished by the used tag (`<nested>`...`</nested>` and `<interleaved>`...`</interleaved>`). It is possible to insert guards for nested protocols by the couple of tags `<guards>` and `</guards>`. The guards are similar to conditions defined above.

4 The English Auction Protocol Example

We take the example of the English auction protocol in order to exemplify the use of the language AXF. The meaning of the English Auction Protocol is the following: one agent (the Auctioneer) wants to sell an item, s/he informs all the agents (or just a part of them who can be interested in it) (the Auction Participants) that a new auction begins. The Auctioneer informs the Auction Participants how much is the item. Then, the auction participants can inform the Auctioneer that they accept the bid. If the Auctioneer has more than two Auction Participants who accept the bid, it selects the one who sent first the agreement and informs it that it wins this round. It informs other Auction Participants who sent an offer that their bid is rejected. The auction is opened till it remains only one auction participant who accepts the bid.

When the Auctioneer closes the auction, it compares the current price for this item to the reserved price. If the hammer price is greater than the reserved price, the item is won by the last agent who accepts the bid. Auction Participants are informed that the auction is closed.

The protocol in Agent UML is shown on Figure 4.

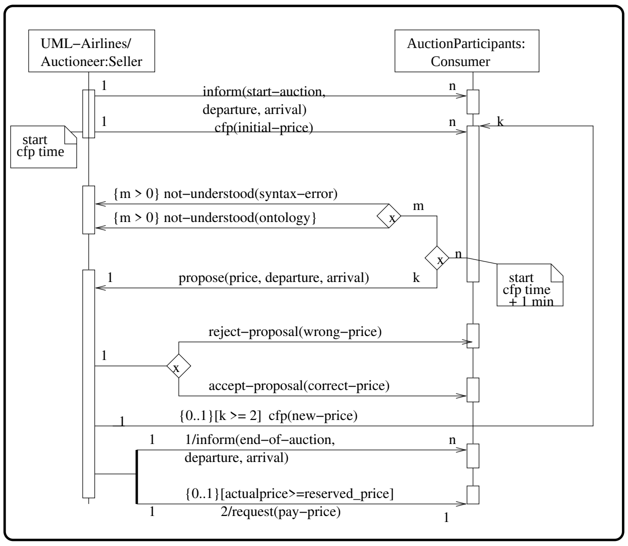

Fig. 4. The English Auction Protocol in Agent UML

The first message is sent to all agents by multicast. This situation is repeated once for the second message which details the item and the price. The deadline for the bidding is given as a comment. Then, auction participants can answer either by a *not understood*[1] message or by a *propose* message. If auction participants send a *propose* message, they inform the auctioneer they accept the bid. When the deadline is passed, the Auctioneer informs the Auction Participants if they have won this round (by a *accept proposal* message) or if they have lost this round (by a *reject proposal* message). The winner is the auction participant who sent first the acceptance of the bid. If the auctioneer has more than two agents which accept the price, it proposes once again the item at a higher price (message *cfp(new-price)*.

If the auctioneer has only one agent, it closes the auction and informs all the auction participants (message *inform(end-of-auction)*). If the current price is greater than the reserved price, it informs the last auction participant who accepts the bid that it wins the item (message *request(pay-price)*).

The AXF version of this protocol is given in appendix.

Messages on protocol diagrams are represented in AXF by a state (between the tags `<state>` and `</state>` and several fields. States allow to go from interaction states to interaction states and are only considered in the textual version of protocol diagrams.

For instance for this message:

[1] FIPA defines in its specifications that all FIPA agents have to be able to send the message *not understood* if they do not understand the content of the message or the message.

```
        <message>
040        <from>
              <role>
                 <instance>UML-Airlines</instance>
                 <name-role>Auctioneer</name-role>
                 <class>Seller</class>
              </role>
           </from>
           <to>
              <role>
                 <name-role>AuctionParticipants</name-role>
050              <class>Consumer</class>
              </role>
           </to>
           <CA>cfp(initial-price) </CA>
           <cardinality> 1 n </cardinality>
           <type> asynchronous </type>
           <comment> start-cfp-time </comment>
           <next-state> 3 </next-state>
        </message>
```

The sender and the receivers are specified between the couple of tags `<from>...</from>` and `<to>...</to>`. The communicative act is given between the tags `<CA>` and `</CA>`.

In this example, the message is sent from the instance *UML-Airlines* to the role *AuctionParticipants*. The communicative act is *cfp(initial-price)*. The message is sent asynchronously and a comment is associated to this one. The next state is the state 3.

A particular state is given lines 83 and 99: `end-state`. This state means that this message corresponds to the end of the interaction.

5 Conclusion

The modeling language Agent UML is currently being worked on as a possible standard. Agent UML is used in order to design several elements in multiagent systems [6] and particularly, interaction protocols. These interaction protocols are described by protocol diagrams. Until now, it does not exist a textual language corresponding to the protocol diagrams. The main advantage of such a language is allowing the reuse of protocols by another designers. Moreover, with such a language, tools have an interchange format.

The aim of this paper is to fill the gap by providing a textual language called AXF (Agent UML eXchange Format). The structure of the language is inspired from the tags used in XML. The paper describes the syntax of AXF and the different features present in this one. In order to exemplify, we apply this language to the example of the English auction protocol.

Several directions are possible for future work. Protocol diagrams are not the only one diagram in Agent UML. We have now to consider the others diagrams and integrate them in AXF. A second direction is about designing a tool for graphically design Agent UML diagrams and above all, validating properties on these diagrams.

References

1. J. L. Austin. *How to Do Things with Words.* Clarendon Press, 1962.
2. B. Bauer, J. Müller, and J. Odell. Agent UML: A formalism for specifying multiagent interaction. In P. Ciancarini and M. J. Wooldridge, editors, *Agent-Oriented Software Engineering (AOSE-00)*, 2000.
3. B. Bauer, J. P. Muller, and J. Odell. An extension of UML by protocols for multiagent interaction. In *International Conference on MultiAgent Systems (ICMAS'00)*, pages 207–214, Boston, Massachussetts, july, 10-12 2000.
4. FIPA. *Specification: Agent Communication Language.* Foundation for Intelligent Physical Agents, http://www.fipa.org/spec/fipa99spec.htm, September 1999. Draft-2.
5. J.-L. Koning, M.-P. Huget, J. Wei, and X. Wang. Extended modeling languages for interaction protocol design. In *Proceedings of Agent-Oriented Software Engineering (AOSE 01)*, Montreal, Canada, May 2001.
6. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proceedings of the Agent-Oriented Information Systems Workshop at the 17th National conference on Artificial Intelligence*, Austin, Texas, july, 30 2000. ICue Publishing.
7. J. Odell, H. V. D. Parunak, and B. Bauer. Representing agent interaction protocols in UML. In P. Ciancarini and M. Wooldridge, editors, *Proceedings of First International Workshop on Agent-Oriented Software Engineering*, Limerick, Ireland, june, 10 2000. Springer-Verlag.
8. OMG. XMI 1.1. Technical Report 01-03-10, OMG, 2001.

Appendix: English Auction Protocol in AXF

```
<name> English-Auction protocol </name>
<author> Marc-Philippe Huget </author>
<date> 02-02-20 </date>
<roles>
  <role>
     <instance>UML-Airlines</instance>
     <name-role>Auctioneer</name-role>
     <class>Seller</class>
  </role>
  <role>
     <instance>AuctionParticipants</instance>
     <class>Consumer</class>
  </role>
</roles>

<protocol-diagram>
  <state> 1
     <message>
        <from>
           <role>
              <instance>UML-Airlines</instance>
              <name-role>Auctioneer</name-role>
              <class>Seller</class>
           </role>
        </from>
        <to>
           <role>
              <instance>AuctionParticipants</instance>
              <class>Consumer</class>
           </role>
        </to>
        <CA>inform(start-auction, departure, arrival)</CA>
        <cardinality> 1 n </cardinality>
        <type> asynchronous </type>
        <next-state> 2 </next-state>
     </message>
  </state>
  <state> 2
     <message>
        <from>
           <role>
              <instance>UML-Airlines</instance>
              <name-role>Auctioneer</name-role>
              <class>Seller</class>
```

```
            </role>
         </from>
         <to>
            <role>
               <instance>AuctionParticipants</instance>
               <class>Consumer</class>
            </role>
         </to>
         <CA>cfp(initial-price) </CA>
         <cardinality> 1 n </cardinality>
         <type> asynchronous </type>
         <comment> start-cfp-time </comment>
         <next-state> 3 </next-state>
      </message>
   </state>
   <state> 3
      <xor-message>
         <branch>
            <xor-message>
               <branch>
                  <message>
                     <from>
                        <role>
                           <instance>AuctionParticipants</instance>
                           <class>Consumer</class>
                        </role>
                     </from>
                     <to>
                        <role>
                           <instance>UML-Airlines</instance>
                           <name-role>Auctioneer</name-role>
                           <class>Seller</class>
                        </role>
                     </to>
                     <CA>not-understood(syntax-error)</CA>
                     <conditions> m >= 0 </conditions>
                     <type> asynchronous </type>
                     <next-state> end-state </next-state>
                  </message>
               </branch>
               <branch>
                  <message>
                     <from>
                        <role>
                           <instance>AuctionParticipants</instance>
```

```
                        <class>Consumer</class>
                     </role>
                  </from>
                  <to>
                     <role>
                        <instance>UML-Airlines</instance>
                        <name-role>Auctioneer</name-role>
                        <class>Seller</class>
                     </role>
                  </to>
                  <CA>not-understood(ontology)</CA>
                  <conditions> m >= 0 </conditions>
                  <type> asynchronous </type>
                  <next-state> end-state </next-state>
               </message>
            </branch>
            <cardinality> m </cardinality>
         </xor-message>
      </branch>
      <branch>
         <message>
            <from>
               <role>
                  <instance>AuctionParticipants</instance>
                  <class>Consumer</class>
               </role>
            </from>
            <to>
               <role>
                  <instance>UML-Airlines</instance>
                  <name-role>Auctioneer</name-role>
                  <class>Seller</class>
               </role>
            </to>
            <CA>propose(price, departure, arrival)</CA>
            <cardinality> k 1 </cardinality>
            <type> asynchronous </type>
            <next-state> 4 </next-state>
         </message>
      </branch>
      <cardinality> n </cardinality>
      <comment> start-cfp-time + 1 min </comment>
   </xor-message>
</state>
<state> 4
```

```
      <xor-message>
         <branch>
            <message>
               <from>
                  <role>
                     <instance>UML-Airlines</instance>
                     <name-role>Auctioneer</name-role>
                     <class>Seller</class>
                  </role>
               </from>
               <to>
                  <role>
                     <instance>AuctionParticipants</instance>
                     <class>Consumer</class>
                  </role>
               </to>
               <CA>reject-proposal(wrong-price)</CA>
               <type> asynchronous </type>
               <next-state> 5 </next-state>
            </message>
         </branch>
         <branch>
            <message>
               <from>
                  <role>
                     <instance>UML-Airlines</instance>
                     <name-role>Auctioneer</name-role>
                     <class>Seller</class>
                  </role>
               </from>
               <to>
                  <role>
                     <instance>AuctionParticipants</instance>
                     <class>Consumer</class>
                  </role>
               </to>
               <CA>accept-proposal(correct-price)</CA>
               <type> asynchronous </type>
               <next-state> 5 </next-state>
           </message>
         </branch>
         <cardinality> k </cardinality>
       </xor-message>
   </state>
   <state> 5
```

```
   <message>
      <from>
         <role>
            <instance>UML-Airlines</instance>
            <name-role>Auctioneer</name-role>
            <class>Seller</class>
         </role>
      </from>
      <to>
         <role>
            <instance>AuctionParticipants</instance>
            <class>Consumer</class>
         </role>
      </to>
      <CA> cfp(new-price) </CA>
      <cardinality> 1 k </cardinality>
      <conditions> k >= 2 </conditions>
      <repetition> 0 1 </repetition>
      <type> asynchronous </type>
      <next-state> 3 </next-state>
   </message>
</state>
<state> 6
   <and-message>
      <branch>
         <message>
            <from>
               <role>
                  <instance>UML-Airlines</instance>
                  <name-role>Auctioneer</name-role>
                  <class>Seller</class>
               </role>
            </from>
            <to>
               <role>
                  <instance>AuctionParticipants</instance>
                  <class>Consumer</class>
               </role>
            </to>
            <CA> inform(end-of-auction, departure, arrival) </CA>
            <cardinality> 1 n </cardinality>
            <type> asynchronous </type>
            <next-state> end-state </next-state>
         </message>
      </branch>
```

```
        <branch>
           <message>
              <from>
                 <role>
                    <instance>UML-Airlines</instance>
                    <name-role>Auctioneer</name-role>
                    <class>Seller</class>
                 </role>
              </from>
              <to>
                 <role>
                    <instance>AuctionParticipants</instance>
                    <class>Consumer</class>
                 </role>
              </to>
              <CA> request(pay-price) </CA>
              <cardinality> 1 1 </cardinality>
              <conditions> actualprice >= reserved_price </conditions>
              <repetition> 0 1 </repetition>
              <type> asynchronous> </type>
              <next-state> end-state </next-state>
           </message>
        </branch>
     </and-message>
  </state>
</protocol-diagram>
```