AUML Protocols: From Specification to Detailed Design (Extended Abstract)

Yoosef Abushark *and John Thangarajah
RMIT University, Melbourne, Australia
{yoosef.abushark,john.thangarajah}@rmit.edu.au

ABSTRACT

In this work, we show how AUML protocol specifications in the Prometheus methodology can be automatically propagated to the detailed design of the methodology by creating appropriate artefacts. The approach is general to all design methodologies that follow the BDI model of agents.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*methodologies*

Keywords

AOSE Methodology;Inter-Agent Interaction Protocols;

1. INTRODUCTION

In multi-agent systems inter-agent interaction plays a significant role. For example, in an agent-based trading system, the buyer and seller agents need to communicate with each other in order to complete a sale transaction. Current *BDI* agent design methodologies allow designers to capture these interactions in the form of *interaction protocols*. A common representation of interaction protocols is AUML (Agent Unified Modelling Language) sequence diagrams [2]. An AUML sequence diagram captures all the possible legal exchange of messages between agents including the temporal aspects.

Although most AOSE methodologies consider agent interaction protocols an essential part of the methodology [1], they provide little (if any) support for ensuring that the protocols are faithfully translated from specification to the detailed design artefacts. It is up to the designer to ensure that the protocols are indeed followed by the system, which can be a tedious and error-prone task that often result in a mismatch between the specification and implementation. In this work we present an approach to address the above. In most of the AOSE methodologies, the *detailed design* (the lowest level) is the closest to implementation and often can be auto-generated to skeleton code. In this work, we provide a mechanism for automatically creating these detailed design structures from the AUML protocol specification. We base our approach on the Prometheus methodology.

*Aknowledges King Abdulaziz University for scholarship

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

2. PROPAGATION MECHANISM

The three factors that influence the protocol propagation to the detailed design: (i) protocol participants, (ii) protocol trigger, and (iii) protocol sequence flow.

Protocol Participants: The messages in a protocol are between two participants (internal agents or actors that are external to the system) and there can be many participants in a single protocol. In a protocol an agent can play two roles: *Sender* or *Receiver* for a particular message. The sender agent needs to be able to send the message and the receiver agent needs to be able to receive and handle (act upon) the received message. The message therefore needs to be propagated into both the agents together with a plan in the sender agent that sends the message and a plan in the receiver agent to handle the message.

Protocol Trigger: The *protocol trigger* is the event (possibly external) that triggers the posting of the first message of the protocol, thus initiating the execution of the protocol. It is important to factor this into the protocol propagation. It is often the case that the protocol trigger is captured by the agent that sends the first message of the protocol. However, in some cases it may be captured by more than one agent.

For example, in Figure 1 the first message ('M1') is sent by 'Agent-A', hence, the protocol trigger would be captured only by 'Agent-A'. However, the first message of the protocol in Figure 2 might be 'M1' from 'Agent-A' or 'M2' from 'Agent-B' depending on the construct's guards. Hence, both agents need to capture the protocol's trigger with the guards propagated to the respective plans as *context conditions* that handles the trigger of each agent.

Protocol Sequence Flow: The sequence flow of an interaction protocol is the execution order of the communication between the participants. Thus, the propagation of the interaction protocols must ensure this sequence flow. When there are multiple messages, there are 3 distinct cases that influence the propagation mechanism: (1)Multiple messages sent in sequence from one agent, (2)Participants exchange messages and (3)Protocol contains constructs (e.g. ALT).

(1) The first case is where an agent sends multiple messages to other agents continuously, for example, 'M3' and 'M4' in Figure 1. The significant point here is to ensure that 'Agent-A' posts these messages in the same order specified, in other words, 'Agent-A' must not post 'M4' before posting 'M3'. Note that, even if the messages were posted by different plans, we show the ordering of messages via dashed-arrows between messages rather than between the plans, as the protocol only specifies ordering of messages, not plans.

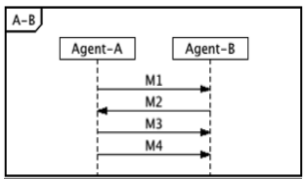

Figure 1: AUML Protocol With No Constructs

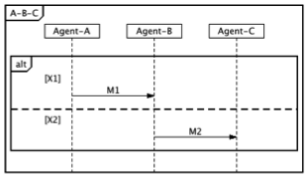

Figure 2: Protocol triggered by multiple agents.

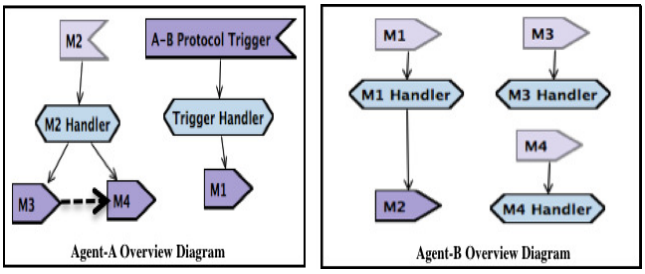

Figure 3: Agent Overview diagrams for the protocol in Figure 1

Further, ordering the plans is too strict, unnecessary and possibly undesirable as the plans may contain steps other than the posting of the message and are often executed concurrently.

(2) The second situation is when the protocol participants exchange messages between each other. For example, see the order of messages 'M1', 'M2' and 'M3' in Figure 1. In this case, it is important to ensure that 'Agent-A' sends 'M3' after receiving 'M2' and that 'Agent-B' sends 'M2' after receiving 'M1'. We enforce this ordering when messages are exchanged by having the plan that handles the incoming message post the outgoing message. For example, for the protocol in Figure 1, 'Agent-A' will have 'M1' posted by the protocol trigger handler plan as it is the first message of the protocol, and 'M3', 'M4' posted by the 'M2 Handler' plan. Similarly 'Agent-B' has a plan that handles 'M1' and posts 'M2', ensuring that ordering (see Figure 3).

(3) The third situation arises when an interaction protocol contains a construct. In addition to enforcing the control specified in the construct, having messages before and/or after a construct also affects the propagation mechanisms.
Before: If there are messages before the construct, then the last message before the construct is treated as the trigger for the construct which is created as an internal event.
After: In the cases where there are messages after a construct, the propagation mechanism needs to consider the fact that the construct in some instances may not occur. For example, in the protocol specified in Figure 4, if the guards 'X1' and 'X2' both evaluate to false, the ALT construct will not execute and the message flow should continue on past the construct.

Considering the direction of the first and last message of a construct's regions, the occurrence of messages before and after a construct and the direction of those messages, provides many combinations of unique cases to be considered when propagating protocols.

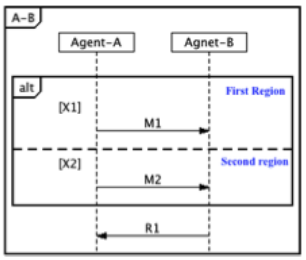

Figure 4: AUML Protocol (ALT Example)

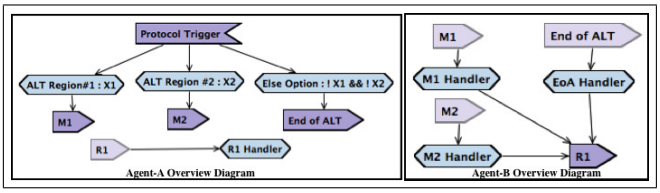

Figure 5: Agent Overview Diagrams (ALT Example)

In developing the propagation techniques we discovered nineteen unique cases for a protocol with just an ALT construct, and twelve unique cases for the OPT construct. For a full list of all the different cases including cases where a protocol contains multiple constructs we refer the reader to a detailed appendix which we have placed online (anonymously) at `http://tinyurl.com/propagation-cases`. Similarly, the algorithms in the form of flow-chart can be found at `http://tinyurl.com/propagation-algorithms`.

3. EVALUATION

We developed a prototype 'eTrading-System' as a multi-agent system with three agents; 'Seller Agent', 'Buyer Agent' and 'Bank Agent'. We specified one interaction protocol, *'Sale Transaction'* where the agents communicate with each other. The system was designed by using the PDT tool that follows the Prometheus methodology [3]. The evaluation that we conducted showed that the manual propagation of that simple protocol took even a well experienced developer 75 minutes. Testing the system, implemented by the same developer, revealed 6 errors with respect to the protocol whilst the proposed approach resulted in a system that followed the protocol with no errors. Thus, by automating this process, the development time is significantly reduced and the resulting system is more reliable than the manual process. Whilst the evaluation was not comprehensive in terms of the number of participants and variation of protocol specifications, it provided a good indication of the kind of benefits that our automated approach provides and the difficulty of the manual propagation.

4. REFERENCES

[1] S. DeLoach, L. Padgham, A. Perini, Susi, and J. Thangarajah. Using three AOSE toolkits to develop a sample design. *IJAOSE*, 3(4):416–476, 2009.

[2] J. Odell, H. Van Dyke Parunak, and B. Bauer. Representing agent interaction protocols in UML. In *Agent-Oriented Software Engineering*, pages 201–218. Springer, 2001.

[3] L. Padgham, J. Thangarajah, and M. Winikoff. Prometheus design tool. In *Proceedings of the 23rd AAAI Conference on AI*, pages 1882–1883, 2008.