Incorporating Knowledge about Interaction for Uniform Agent Design for Simulation and Operation

(Extended Abstract)

Jan D. Gehrke
jgehrke@tzi.de

Arne Schuldt
as@tzi.de

Centre for Computing Technologies (TZI)
University of Bremen, Am Fallturm 1, D-28359 Bremen

ABSTRACT

This paper deals with transferability of agent implementations from real-world operation to simulation. The objective is to minimise the additional effort for testing and evaluating agent behaviour, mainly arising from the need for synchronisation of simulation time. This can be achieved by incorporating knowledge about agent interaction. An implementation for the FIPA request agent interaction protocol demonstrates the feasibility of this approach.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems; I.6.8 [**Types of Simulation**]: Distributed

General Terms

Algorithms, Experimentation

Keywords

Agents, multiagent systems, multiagent-based simulation, synchronisation, communication, interaction protocols

1. INTRODUCTION

Multiagent systems (MAS) decrease problem complexity by decomposition. Thus, they ease developing software intended, e. g., to control distributed business processes. But thorough testing and evaluation is still important, since effects emergent from agent interaction cannot be predicted at design time [3]. Instead, simulation allows evaluating such systems at run-time. Multiagent-based simulation (MABS) should allow transferring agents easily from operation to simulation owing to its direct model mapping [4]. But in contrast to reality, simulation time does not progress implicitly. Unfortunately, this has a potentially high impact on agent implementation. Demanding a developer to explicitly consider handling of simulation time is contradictory to the seamless transfer intended from operation to simulation. Therefore, this paper contributes an approach towards uniform agent design for operation and simulation.

Cite as: Incorporating Knowledge about Interaction for Uniform Agent Design for Simulation and Operation (Short Paper), Gehrke and Schuldt, *Proc. of 8th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, Decker, Sichman, Sierra and Castelfranchi (eds.), May, 10–15, 2009, Budapest, Hungary, pp. XXX-XXX.

2. UNIFORM AGENT DESIGN

The overall goal is to disburden agent programmers from any thought on simulation-specific issues. In particular, simulated agents must be synchronised because they are executed in parallel. MABS systems should thus cope with implicit simulation time synchronisation to ease transferability of agent code. The particular focus of this paper is on interaction between software agents. Thus, physical environment design, physical simulation, and physical agents in general are not dealt with. Likewise, simulation of sensor processing is out of the scope of this paper.

Synchronisation can be conducted either in an optimistic or in a conservative way. Here, we employ a conservative approach due to the lower space complexity [2]. This property is particularly important if agents have extensive knowledge bases. In contrast to logical processes in classical parallel distributed simulation, agents are autonomous. Thus, additional quality criteria regarding synchronised message processing must be considered [5]. But despite of their importance, time model adequacy, causality, and reproducibility are not sufficient to implement uniform agent design. Besides, MABS systems should aim at

1. complying with MAS standards such as FIPA to ensure interoperability with agent on other platforms,
2. supporting and enhancing a well-established agent development environment developers are familiar with,
3. minimising simulation-specific issues for minimal effort of agent transfer from operation to simulation.

The envisioned optimal result is a simulation environment that is applied by just replacing the library of the agent environment for operation by an adapted library that contains a simulation middleware. The agent code implemented by users should not need to be changed.

3. IMPLICIT TIME PROGRESSION

Probably, the most frequent issue of programming agents for MABS concerns progression of time. When agents are applied in real-world context, physical time progresses automatically. Therefore, programmers expect the same from simulation time during execution of simulation model code. In discrete event simulation, the agent processes events, reasons about subsequent actions, and generates new events or messages with certain simulation timestamps. But processing these events does generally not take simulation time although it consumes computation time. Each progression of

simulation time may need synchronisation with other agents and has to be triggered explicitly or following some premodelled rules.

Without an implicit determination and handling of time progression, the simple code of sending a message to some agent and subsequently waiting for the answer in the next line of code is not applicable in simulation. This code mistakenly assumes that simulation time progresses because computation time does. Actually, this code may cause the simulation to stop unintentionally and permanently. The agent will wait for an answer that will not be received without simulation time progression. Hence, the general question for uniform agent design is when simulation time of agents should progress and how to implement this without burdening the programmer with specific thoughts on the time model or synchronisation. The MABS middleware must then resolve cases where it becomes a problem.

This can be achieved by identifying most important cases and introducing implicit time progression, i. e., unnoticed by user. This is, for instance, possible for message reception which always requires time progression if there are no messages available yet. A promising approach to handle message reception implicitly is to incorporate knowledge about agent interaction. Interaction protocols clearly define when agents wait for responses to messages sent. Hence, it is possible to implicitly derive when simulation time must progress.

4. IMPLEMENTATION

Examining the concept of uniform agent design requires implementations for both simulation and operation. In a case study, exemplary aspects have been implemented in the PlaSMA simulation middleware[1]. This middleware enhances the JADE [1] agent platform with means for simulation. It handles experiment initialisation, time management including message passing, as well as agent lifecycle management. Hitherto, the simulation middleware required explicit requests for time progression.

The FIPA request interaction protocol is the chosen subject of the initial proof of concept for explicit time progression. This choice can be motivated as follows. Firstly, the protocol is rather simple and thus traceable as opposed to, e. g., auctions. Secondly, it is nevertheless widely applied for versatile tasks. In JADE, interaction protocols are implemented by behaviours. Sending and receiving messages is encapsulated by the underlying implementation. Agent developers can simply implement callback methods to prepare messages and to handle responses respectively (Fig. 1). Behaviours are internally structured as finite state machines. Having sent a message, the behaviour switches into the next state to receive the responses which are then again delegated to the respective callback methods.

This allows transparently extending the implementation for simulation as follows. Consider that a request is sent. For time model adequacy each message transfer must consume simulation time [5], i. e., responses cannot be received without time progression. Hence, the simulation middleware can implicitly request time progression. Messages arriving can then be processed by the agent. If at one point in simulation time all messages from the past are processed, one can again implicitly request progression. For causality reasons no newer messages may be processed yet [5]. Generally

[1]http://plasma.informatik.uni-bremen.de/

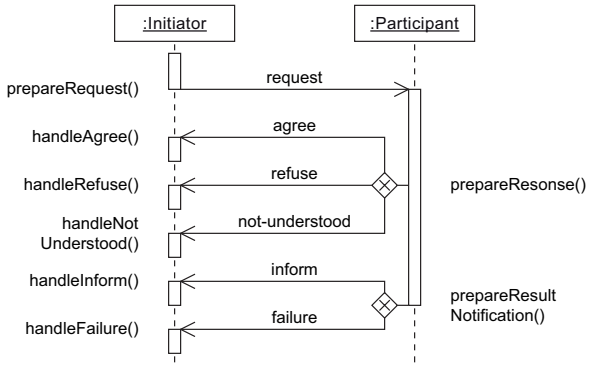

Figure 1: JADE callback methods handling the states of the FIPA request interaction protocol

speaking, time progression can always be requested implicitly whenever an agent awaits further messages that are not yet available in its inbox.

5. CONCLUSION AND OUTLOOK

The prototypic implementation for the request protocol indicates that uniform agent design based on knowledge about interaction is applicable in principle. The respective compatible libraries allow switching transparently from operation to simulation. Summing up, this approach eases evaluating agents in simulation to a great extent.

However, not all knowledge required for implicit time progression can be derived from agent interaction protocols. Additionally, time progression may also be necessary when agents act without interaction. Agents in real-world operation then sleep until some predefined event or some point in time. Likewise, the simulation middleware must implicitly map these time-stamps from physical time to simulation time. For further investigations, this aspect is currently implemented for arbitrary JADE behaviour classes.

Acknowledgement. This research is funded by the German Research Foundation (DFG) within the Collaborative Research Centre 637 “Autonomous Cooperating Logistic Processes: A Paradigm Shift and its Limitations” (SFB 637) at the University of Bremen, Germany.

6. REFERENCES

[1] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Chichester, UK, 2007.

[2] R. Fujimoto. *Parallel and Distributed Simulation Systems*. John Wiley & Sons, New York, NY, USA, 2000.

[3] N. R. Jennings. An Agent-Based Approach for Building Complex Software Systems. *CACM*, 44(4):35–41, 2001.

[4] H. V. D. Parunak, R. Savit, and R. L. Riolo. Agent-Based Modeling vs. Equation-Based Modeling: A Case Study and Users’ Guide. In *MABS 1998*, pages 10–25, Paris, France, 1998. Springer-Verlag.

[5] A. Schuldt, J. D. Gehrke, and S. Werner. Designing a Simulation Middleware for FIPA Multiagent Systems. In *WI-IAT 2008*, pages 109–113, Sydney, Australia, 2008. IEEE Computer Society Press.