Debugging Agents in Agent Factory

Rem Collier

School of Computer Science and Informatics, University College Dublin, Ireland
rem.collier@ucd.ie

Abstract. The ability to effectively debug agent-oriented applications is vital if agent technologies are to become adopted as a viable alternative for complex systems development. Recent advances in the area have focussed on the provision of support for debugging agent interaction where tools have been provided that allow developers to analyse and debug the messages that are passed between agents.

One potential approach for constructing agent-oriented applications is through the use of agent programming languages. Such languages employ mental notions such as beliefs, goals, commitments, and intentions to facilitate the construction of agent programs that specify the high-level behaviour of the agent. This paper describes how debugging has been supported for one such language, namely the Agent Factory Agent Programming Language (AFAPL).

1 Introduction

The provision of support for debugging MAS is increasingly seen as an important topic of research [10][17]. Much of the impetus behind this surge in interest is the ever increasing complexity of the problem domains to which agent technologies are being applied. Further, given the slow emergence of prefabricated agent systems that are open and extensible, developers are increasingly required to enhance and adapt these systems to construct new solutions. A side effect that arises from this trend is the need to integration prefabricated agent systems that have been developed using different agent toolkits.

While the issue of how to integrate diverse agent toolkits and architectures is well established through standards bodies such as the Foundation for Intelligent Phyisical Agents (FIPA) [12], the issue of how to debug synthesized multi-agent systems is not. Initial approaches to debugging have, by and large, come from the agent development toolkits research community [14][19][20]. While many of these approaches share common ground, for example agent state viewing tools and message monitoring tools, they are all intimately linked to a specific agent toolkit and do not interoperate.

More recently, a number of new debugging tools have begun to emerge [3][16]. The strength of this new wave of tools is twofold: (1) they have the potential to be independant of any specific agent toolkit or architecture, and (2) they present data at a level of abstraction that scales beyond the equivalent first wave solutions. One such tool is the ACLAnalyser tool, which provides analyses

R.H. Bordini et al. (Eds.): ProMAS 2006, LNAI 4411, pp. 229–248, 2007.
© Springer-Verlag Berlin Heidelberg 2007

of the interactions that occur during the execution of an agent system. While this system has been implemented and integrated with the JADE framework, its analysis is based on the passing of FIPA-ACL messages. As such, it should be a relatively trivial task to integrate the analyser tool with any FIPA-compliant agent platform, and subsequently provide support for the debugging of agent systems that are implement using multiple FIPA-compliant agent toolkits.

While implementation independence is an obvious strength when charged with the task of analysing large scale agent systems that have been implemented using multiple agent toolkits, it is not sufficient. Implementation independant tools, tend to focus on externally observable behaviours and are able to help developers to hone in on the particular agent or community of agents that are functioning incorrectly. As a result, implementation independant techniques often are not able to provide support for the inspection of the internal workings of those agents. In cases where such support is provided, for example [16], there is a reliance on the developer correctly annotating the internal behaviour of the agent to provide an observable trace of the agents behaviour.

This paper adopts the perspective that the provision of support for debugging of the behaviour of individual agents is best supported through the implementation of debugging tools that are tailored to those agents. Accordingly, this paper focuses on how one such agent development toolkit, known as Agent Factory [5] and its associated agent programming language, the Agent Factory Agent Programming Language (AFAPL), supports the debugging of agent systems. It tackles this issue in two parts: at compile time, and at run time. The former of these perspectives aims to reduce, as far as possible, the number of bugs that appear in the deployed agent system. The latter of these perspectives then focusses upon the support that is provided once the system is deployed.

2 Related Work

The Zeus Toolkit [19] comes packaged with a set of visualisation and debugging tools that provides support at both the agent level and the social level. At the agent level, Zeus includes an *micro tool*, which developers can use to monitor the internal state of the agent. Specifically, it provides a message view (messages sent and received), a summary of actions taken in response to incoming messages, a view of the coordination process for individual goals, a diary of tasks that the agent is committed to, a list of allocated resources, and finally, a summary of the tasks that are being executed or are scheduled for execution [19]. At the social level, the *society tool* provides developers with views of both the social structure of the implemented agent system and the messages that are passed between the agents in that system. Filtering mechanisms allow the developer to focus in on particualr types of interactions within the system.

A more recent tool is *ACL Analyzer* [3]. This tool provides social level support for developers of large multi-agent systems through the provision of visualisation and data mining tools for the analysis of agente interactions. Whilst the tool is implemented for the Jade toolkit [2], it has been designed in a way that it can