ACLAnalyser: a Tool for Debugging Multi-agent Systems

Juan A. Botía and Alberto López-Acosta and Antonio G. Skarmeta[1]

Abstract. Multi-agent systems are a special kind of distributed systems in which the main entities are autonomous, in a proactive sense. These systems are special because their unpredictability. Agents can spontaneously engage in complex interactions, guided by their own goals and intentions. When developing such kind of system, there are many problems the designer/programmer has to face. All these problems make virtually impossible to totally debug a complex enough multi-agent system. In this article we describe a debugging tool we have developed in our lab which pretends to alleviate the problems derived from distribution and unpredictability.

1 Introduction

In this paper we propose a tool devoted to assist on debugging both functional and coordination/cooperation related errors on Multi-agent systems. The ACLAnalyser tool has been designed to analyze runs on the JADE (Java Agents Development Environment) platform. Hence, it is useful to debug FIPA compliant MAS [1]. It is coded in Java and available as an add-on from the JADE web site[2].

2 The architecture of the ACLAnalyser

The architecture of the ACLAnalyser appears depicted at figure 1. In the figure, four main elements appear. These are the JADE execution environment, a logger, a relational database and the user module. The JADE environment represents the multi-agent system being debugged. The logger is actually another JADE agent (i.e. a spy agent), which actually is not part of the MAS being debugged. The JADE platform must be launched, along with the logger. This agent is similar to the Sniffer which comes with the JADE distribution. Its purpose is to intercept all the messages exchanged between agents. However, instead of simply reproducing it like the Sniffer does, this spy agent has two different tasks: (1) to keep track of all the interactions being performed in the run currently executing and (2) to store all messages in the third element of the architecture, a relational database. The last element of the architecture is the user module. This is in charge of presenting results to the user in order to help him keeping track of all events produced in the execution. This module is the most important part of the tool because it shows the user all the information needed to check if something went wrong in the execution of the MAS and, in that case, to make an appropriate diagnostic.

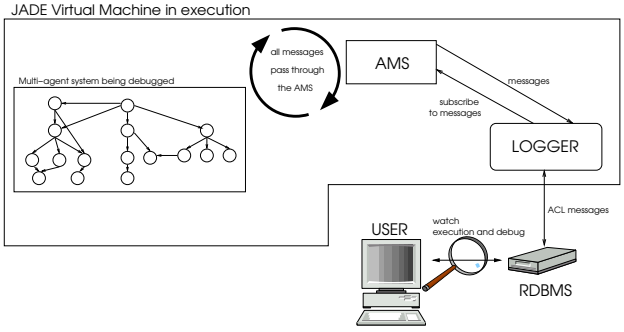

Figure 1. Architecture for the ACLAnalyser tool

3 Interaction protocols

In a run of the MAS being debugged, all the messages are organized into conversations, and all conversations belong to a session. A session is the set of messages exchanged in a run.

As the user module could operate with any conversation produced in a run, the tool needs a way of representing interaction protocols and a method to match a sequence of messages with a concrete interaction protocol.

In the ACLAnalyser, an interaction protocol has an internal representation as a finite state automaton. When a new conversation is initiated between agents, an automaton is created for all the participants as they become known for the tool. In this way, the state of the automaton represents the state of the conversation for the corresponding agent. When a new message is received by the spy agent, the roles for the sender and receiver of the message are obtained. Having these roles, state of each participant and the performative of the message, transitions are done for all automaton. All the interaction protocols accepted as standars in FIPA (see http://www.fipa.org) are defined in the ACLAnalyser. As well, the tool can be extended to recognize any ad hoc interaction by simply defining the corresponding automaton by means of a graphical interface.

[1] Departamento de Ingeniería de la Información y las Comunicaciones. Universidad de Murcia, Spain. email: juanbot, skarmeta@um.es

[2] http://jade.tilab.com

4 Zooming out

The ACLAnalyser tool can show a view consisting on a directed graph of all the agents implied in a run (i.e. a concrete session) or a graph of all the agents implied in a simple interaction (i.e. a conversation). In this graphical view, both nodes of the arc are agents and if, in the case that there is an arc from the agent i to the agent j, that means that agent i have sent one or more messages to agent j, and also that messages have been received by j. Also, each arc is labeled with the number of messages sent and the total number of bytes transferred. This basic view of the whole system has the following applications: detection of no communication, when expected, between two or more agents, detection of excessive number of bytes exchanged between two or more agents and detection of unbalanced execution configurations in which agents from a concrete group (or machine) show an amount of activity not compensated with the rest.

This kind of view is still useful when hundreds or even thousands of agents are implied, because it is possible to make a zoom out. This is performed by using a clustering process on the messages to detect groupings of agents and present them by means of that. It is based on the ROCK [2] clustering algorithm. This clustering algorithm works with boolean and categorical (i.e. symbols) data. ROCK uses the concept of link. This term is, in turn, based also in the concept of neighbor. Two items of data are considered as neighbors if they share some degree of similarity, depending on the problem. The number of neighbors between two entities corresponds to their number of links. Once the links are calculated, groupings are determined between entities which share a high number of links. The number of groups detected is a configuration parameter of the algorithm. Hence, the lower the number, the higher the abstraction level we are using to look at the agents society.

With respect to this concrete problem (i.e. the application of ROCK to exchanged messages between agents) two agents will be considered as neighbors if they exchanged some messages. In the one hand, the figure 2 shows a normal view the ACLAnalyser offers with a hundred agents. In the other hand,

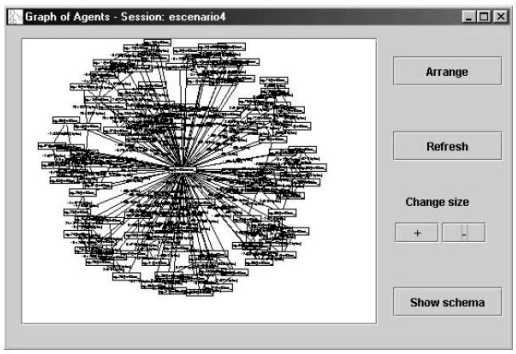

Figure 2. A normal view of a sample MAS being debugged

figure 3 represents the same MAS to which a zoom out has been applied. Notice that ten agent groups have been created. Now the arcs between two different groups represent the total size of the messages exchanged between the groups. It there no exists an arc between two groups it means that no message has been exchanged.

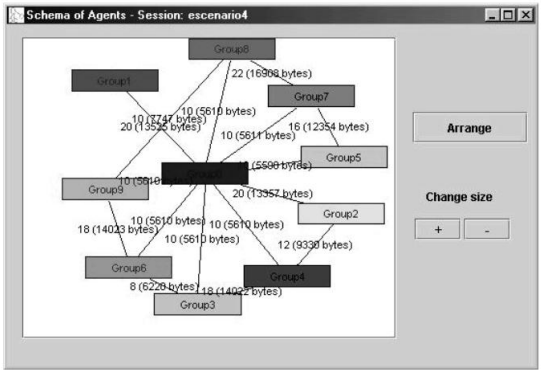

Figure 3. The MAS of figure 2 with a zoom out

5 Conclusions and future works

In this paper, some of the many functionalities of the ACLAnalyser has been shown. Its strengths are the use of a relational database which allows storage of any number of complex multi-agent interactions, views of the MAS at different abstractions levels by using categorical clustering, and a complete and versatile window system which allows accessing any information available for a run. This, as a whole, compound a powerful debugging system for complex multiagent systems. Future works include to investigate the application of OLAP (On-line Analytical Processing) techniques for visualizing the database and a more coupled integration with the execution system (i.e. JADE platform). The tool is also being extended to be distributed. In this scenario, a problem arises and this is that of how to order messages coming from different JADE platforms in the database. We are studying to apply logical clocks to solve the problem.

REFERENCES

[1] J Dale and E. Mamdani. Open standards for interoperating agent-based systems. *Software Focus*, 2(1), 2001.

[2] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000, (citeseer.nj.nec.com/guha00rock.html).