On the Application of Clustering Techniques to Support Debugging Large-Scale Multi-Agent Systems

Juan A. Botía, Juan M. Hernansáez, and Antonio F. Gómez-Skarmeta

Departamento de Ingeniería de la Información y las Comunicaciones
Universidad de Murcia

Abstract. This work analyses the problematic of debugging a multi-agent system. It starts from the fact that MAS are a particular type of distributed systems in which the active entities are autonomous in the sense that behavior and knowledge of the whole system is distributed among agents. It situates the problem by firstly studying the classical approaches for conventional code debugging and also the techniques used in distributed systems in general. From this initial perspective, it tries to situate agent and multi-agent systems debugging. It finally proposes the use of conventional data mining tasks like clustering to, by summarising, help in debugging huge MAS.

1 Introduction

Nowadays, there is almost a total lack of tools to assist in the task of debugging and monitoring agent based distributed information systems in where the typical scenario of execution involves hundreds of thousands agents. In such cases, strong and flexible tools are needed to log and recover all necessary data and to analyze it by giving enoughly abstract views. These views should maintain the appropriate abstraction level because, in scenarios involving such a high number of agents, there is a clear necessity of summarizing to gain sight into what is really happening in the system. This paper proposes the use of data mining over agent messages to support this difficult task. Techniques exposed in this paper are implemented in the ACLAnalyser. This tool is described elsewhere [4] and you can find it at the JADE agents platform web page, in the form of an add-on.

The rest of the article is organized as follows. Section 2 introduces the general problem of debugging pieces of software and delimitates the particular issue of debugging agents. Section 3 is devoted to define the general framework we propose here, i.e. to use data mining on agent communication language messages to assist the developer in debugging a MAS (Multi-Agent System). Finally, section 4 outlines initial conclusions and points out future research.

2 Debugging Software Artefacts

Debugging and testing software artefacts is not easy task [22]. Moreover, programming errors may lead to get an information system down virtually all the

R.H. Bordini et al. (Eds.): ProMAS 2006, LNAI 4411, pp. 217–227, 2007.

time, make services offered by a software company unavailable, make not dessired changes to valuable information and, in the worst case, produce wrong outputs.

In debugging software programs we find two different approaches. The first one consists in explicitly modifying the program being debugged to include checks about, for example, the values of critical variables. After that, the analysis process is done without having to execute the code, i.e. statically. This is the approach we may find, for example, in what is called *model checking* [5]. It consists in, given a code to check, to use a graph which represents the different states in which the code being checked may be found. After this graph is built, we use some search algorithm to explore all the possible states trying to find error states. For an example of a real system which systematically checks code on C and C++, please see [16]. A related approach is *program analysis*. It consists in analyzing the code, without having to execute it in order to detect, for example, deadlocks and data races. It detects problematic code regions in the source code, analyzing them and giving an accurate diagnosis about possible errors that may occur during runtime. Please, see [7] for an example of such debugging programs. The second approach consists in dynamically monitoring the program. This is what is called *dynamic checking* [24]. With a dynamic checker, the target code is modified in some way that it checks itself about invariants, and reports on possible violations of the invariants are used to follow the behavior and perform some diagnosis when necessary.

Considering this ideas, the following questions arise. Do model and dynamic checking have any applicability in the context of MAS programming? To what extent is conventional code, debugged by the above mentioned tools, related to that of a typical MAS? Starting from the considerations made in the last paragraph, we may deduce from the last paragraph that model and dynamic checking, may be applied to debug the source code of single agents, provided that they are coded in a language, let it be denoted with $\mathcal{L}$, and we have a checker for $\mathcal{L}$. In this case, we may detect data races and deadlocks in the internals of an agent. At the end of the day, this would be conventional debugging, i.e. it would be like debugging any other program written in $\mathcal{L}$. Notice that this simple analysis has been done on the basis that $\mathcal{L}$ is a general purpose language and not agent oriented like APRIL and JACK may be. An example of a general purpose language used to code agents is Java, as it is used in agent programming environments like JADE, for example. In this case, any existing debugger may be used to analyze the internals of any single agent. In the case that an agent oriented programming language is used, specific debugging techniques either pertaining to model or dynamic checking should be developed first. They should take into account typical agent programming elements like believes, goals, tasks, roles and so on. One good example of such approach is the Tracer tool [12]. It uses reverse engineering and a particular model ckecking to generate *relational graphs* which relate beliefs, intentions and actions. They are also used to generate explanations on actions.

But, would it be possible to apply model and dynamic checking at the inter-agent level? What are the particularities of MAS programming which makes it