Debugging Mobile Agent Systems

Masayuki Higashino
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
s032047@ike.tottori-u.ac.jp

Shin Osaki
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
s092018@ike.tottori-u.ac.jp

Shinya Otagaki
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
s082009@ike.tottori-u.ac.jp

Kenichi Takahashi
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
takahashi@ike.tottori-u.ac.jp

Takao Kawamura
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
kawamura@ike.tottori-u.ac.jp

Kazunori Sugahara
Department of Information and Electronics
Graduate School of Engineering, Tottori University
Tottori 680-8552, Japan
sugahara@ike.tottori-u.ac.jp

ABSTRACT

A mobile agent is an autonomous software module that can migrate between different computers. A mobile agent is designed and implemented like a human, and mobile agents work together by interactions among them like a human community. Thus, a mobile agent technology is helpful when we develop distributed systems with an easy-to-understand design and implementations for humans. Many researchers have proposed various applications through mobile agent technologies. However, mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. This paper discusses problems of mobile agents for debugging, and proposes a remote debugger in order to solve these problems. Our proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system.

Categories and Subject Descriptors

I.2.11 [**Distributed Artificial Intelligence**]: Multiagent systems—*mobile agent systems*; D.2.5 [**Testing and Debugging**]: Distributed debugging

General Terms

Design

Keywords

mobile agent systems, debugging, platforms

1. INTRODUCTION

A mobile agent is an autonomous software module that can work on different computers and migrate between these computers. These computers install an agent runtime environment (ARE) which is connected by a network, and a mobile agent can work on these AREs. A mobile agent is designed and implemented like a human, and these mobile agents work together by interactions among them like a human community. Thus, mobile agent technologies are helpful when we develop distributed systems with easy-to-understand design and implementations for humans.

Many researchers have proposed various design, implementations, applications, and platforms of mobile agent systems [10, 19]. However, mobile agent technologies are not used much as compared to other networking or programming technologies in the real world because migrations of mobile agents make it difficult to debug the system. Since many mobile agents work concurrently and migrate to many computers in a mobile agent system, when a problem occurs in a mobile agent, a programmer must find this mobile agent with difficulty from many mobile agents and many AREs. Since a mobile agent migrates from a computer to another computer, in order to debug this mobile agent, it requires connections to these computers and functions of remotely debugging. Since a mobile agent migrates even when a programmer is debugging this mobile agent, the remote debugger must change a destination of a connection to a computer which the agent migrates to. Additionally, since behavior of mobile agent is affected from interactions with other mobile agents, even if a mobile agent executes a same program of a task, then a result of it is not always same. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced.

A debugger for ordinary software other than a mobile agent system does not support functions for characteristics described above of a mobile agent system, and it is hard to debug mobile agent systems. Therefore, this paper discusses problems on debugging of mobile agent systems, and proposes a remote debugger for mobile agent systems in order to solve these problems. Proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system.

The rest of this paper is organized as follows. Section 2 points out related works. Section 3 discusses problems and approaches in debugging of mobile agent systems. Section 4 describes design and a implementation of our approaches. Section 5 shows the evaluations and their results. Finally, Section 6 draws the conclusions.

2. RELATED WORKS

In the research area of multi-agent systems, there have been proposed several technologies for developers. IOM/T [23], *Q* [24], and SDLMAS [25] is a description language to define interactions and scenarios among agents for multi-agent systems. However, the definitions supported by these languages are based on only remote messaging among agents who basically stay on a computer and do not basically migrate. Thus, it is difficult to apply interactions proposed on the area of multi-agent systems to interactions including mobility.

Also, in the research area of mobile agent systems, there have been proposed approaches like mentioned above. Mobile Object-Z (MobiOZ) [21, 22] can specify mobile agent applications with a specification notation extended the Z formal specification notation [11] and verify the model of the applications by the SPIN (Simple Process meta language INterpreter) model checker [3]. LAM (Logical Agent Mobility) [27, 7] also can verify models of applications by the SPIN. π-ADL [17] is an architecture description language based on the higher-order typed π-calculus, which is one of a model for a process calculus, for specifying dynamic and mobile software architectures, and models described by π-ADL can be verified by CADP (Construction and Analysis of Distributed Processes) toolbox [9], and it can be applied to mobile agent systems.

If definitions with these languages (notations) are valid logically, generating native source codes as implementations from these definitions, bugs by implementing do not occur. However, in order to apply these definitions to existing applications, developers must design and implement additional definitions and generators for applications. Therefore, these approaches make the applications robust, but choosing these approaches becomes difficult in practice, because the costs increase with increasing a scale and complexity of the applications. Additionally, there are applications that cannot be applied to approaches mentioned above. Therefore, a debugger for mobile agents is required.

Several researchers have proposed a tool to visualize and debug interactions among agents [16, 13, 20, 26, 5], but these tools do not focus on mobility of agents. JADE [2] and Agent Factrory [6, 4] is a platform for multi-agent systems that includes debuggers. These platforms support mobility of agents, but their debuggers do not support and focus on mobility of agents, because these platforms mainly focus on multi-agent systems and subsequently introduced mobility. Also [15] suggested requirements for IDEs (Integrated Development Environments) to support construction of multi-agent systems, but they touched only on mobility just a little. MiLog [8] is a framework (platform) including an IDE for mobile agent systems that can visualize locations of agents in a network and dump logs of each agent. However, because its features are simple, in massive mobile agent systems, debugging processes with these features is laborious.

Therefore, this paper discusses problems of mobile agents for debugging, and proposes an effective remote debugger in order to solve these problems.

3. PROBLEMS AND APPROACHES ON DEBUGGING MOBILE AGENT SYSTEMS

Mobile agent systems have characteristics that multiple mobile agents interact with other agents, work on multiple nodes, and migrate to these nodes. However, these characteristics make it difficult to debug the system for the following reasons:

1. *Multiple targets to debug* A debugger for general software targets one application (called a process in OS). However, in a mobile agent system, many multiple mobile agents work concurrently and migrate concurrently to many nodes. Thus, at first, a debugger for mobile agent systems must search targets from a network with search queries including various condition of mobile agents.

2. *Working on remote nodes:* In mobile agent systems, a mobile agent works on distant nodes. Thus, in order to check behavior of an agent, a debugger must connect to a node on which this agent is working. A remote debugger is required in order to debug mobile agents.

3. *Migrating on debugging:* A mobile agent migrates among nodes even when a programmer is debugging them. However, a debugger for general software or multi-agent systems does not consider frequent changing a destination according as migrations of agents. Thus, in order to debug this mobile agent, the remote debugger must change a destination of a connection to a node to which this agent migrates. Additionally, the debugger must continuously support features for debugging such as single-stepping, breaking, and tracking the values of variables, even when the agent migrates.

4. *Side effects by interactions other agents and nodes:* Behavior of a mobile agent is affected from interactions with other mobile agents and AREs. Even if a mobile agent executes a same program of a task, then a result of it is not always same by these side effects. Therefore, when a programmer reproduces a bug in order to debug it, the environments of not only mobile agents but also AREs must be reproduced.

4. DESIGN AND IMPLEMENTATION

This paper considers a design of a debugger for mobile agent systems on the basis of approaches described above.

4.1 Searching Function

4.1.1 Parameters of Search Query

A programmer cannot grasp all mobile agents in the system; because there are multiple targets to debug in mobile agent systems and the number of mobile agents and nodes is large. Therefore, a searching function is required which finds agents from a network of mobile agent systems by various parameters. In order to search mobile agents, a name, a state, and a location of a mobile agent is a useful information. A name gives a classification to mobile agents such as a class name of instances, a role, etc. A state represents a runtime state of a mobile agents like a state of a thread in Java, such as *running*, *waiting*, and *terminated*. A location is an identifier of node on which a mobile agent is working currently. By using these parameters as a query for searching, a programmer can filter mobile agents which have suspicious behavior from many mobile agents in a system.

4.1.2 Identifiers for Mobile Agents

Because some platforms (frameworks) of mobile agents have no identifiers to give uniqueness for mobile agents, a debugger for mobile agent systems must give a new identifier to searched mobile agents. Thus, a debugger gives UUID (Universally Unique Identifier) [14] to mobile agents as unique identifiers. Also, these identifiers are also used by other functions described below.

4.1.3 Method to Search

Since mobile agents are used on various network topologies such as a P2P (peer-to-peer) and a client-server, a suitable method to search depends on a network topology used by an application. Thus, our approach does not limit the kind of method to search mobile agents. In our implementation, we use a query flooding simply because this paper does not focus on a performance of searching.

4.2 Monitoring Function

Because it is not always find mobile agents which have suspicious behavior when searching, the notifications when mobile agents cause problems are required. Therefore, if a parameter of search query matches a mobile agent when the mobile agent causes problems, the node on which this agent stay sends a notification to a node of developers.

4.3 Debugging Function

4.3.1 Breaking of Mobile Agent

In order to debug a mobile agent, a programmer finds a mobile agent of a target by using searching or monitoring function and sets the mobile agent to a debug mode. Because there is a case that multiple instances of mobile agents are generated from one class or cloned, it is possible that multiple mobile agents which are generated from same erroneous source codes cause problems. Thus, breakpoints must be specified in source codes, and these mobile agents must have source codes on runtime.

4.3.2 Single-Stepping of Mobile Agent

Basically, a function of single-stepping can be realized by applying breaking but there are problems when a migration occurs between steps. Generally, nodes are connected by different sessions of TCP (transmission control protocol). Since TCP maintains order of packets in a session but TCP does not maintain order between different sessions, a remote debugger must maintain order by other method. Thus, a mobile agent in a debug mode has a counter of the number of migrations. When the mobile agent migrates, this agent sends a notification of a migration with the counter. Additionally, mobile agent migrates with an own runtime state such as a function call stack, a program counter, and information of breakpoint. Thus, a programmer can continuously execute single-stepping of a mobile agent with migrations.

4.4 Logging Function

In order to reproduces a bug, the environments of not only mobile agents but also AREs must be reproduced. Thus, in order to support reproducing, each node stores logs of interactions between mobile agents and nodes.

5. EXPERIMENT

In order to evaluate our debugger for mobile agent system, we have experimented to debug a test application which includes a bug.

Our debugger is implemented to the mobile agent framework called Maglog (Mobile AGent system based on proLOG) [12]. The agent of Maglog is implemented by extending PrologCafé [1]. PrologCafé is a 100% pure Java implementation of the Prolog programing language, which contains a Prolog-to-Java source-to-source translator and a Prolog interpreter.

Therefore, Java programmers are able to completely access to a Prolog interpreter's internal execution states such as choice point stack, trail stack, a set of variable bindings, and etc. When an agent migration, agent execution state, application data, program codes, and identifiers are converted to byte array by Java Object Serialization [18], and transferred among AREs through HTTP/1.1 protocol. Thus, the information of breakpoint is realized as an instance of a prolog interpreter.

In this application, there is an agent that migrates to nodes at random. The application is implemented by a third party, and a programmer has tried to debug the application without knowing the bug.

In the experiment, the agent has stopped with an exception on somewhere in a network. The programmer has found the location of the agent by using search function and tried to clear cause of this by using single-stepping function. As a result, the programmer could find out cause that the agent mistakes the increment for the decrement of a list contains random destination locations.

6. CONCLUSION

This paper has discussed problems of mobile agents for debugging, and proposed a remote debugger in order to solve these problems. Our proposed remote debugger supports functions of searching, single stepping execution, breaking, and viewing variables for a mobile agent who behaves with migrations on a running system. The experiment shows that our debugger for mobile agent systems finds effectively bugs.

7. REFERENCES

[1] M. Banbara, N. Tamura, and K. Inoue. Prolog cafe: A prolog to java translator system. In *Proceedings of the 16th International Conference on Applications of*

Declarative Programming and Knowledge Management, pages 1–11, 2005.

[2] F. Bellifemine, G. Caire, T. Trucco, and G. Rimassa. *JADE PROGRAMMER'S GUIDE*. Telecom Italia S.p.A., 2010.

[3] M. Ben-Ari. *Principles of the Spin Model Checker*. Springer London, 2008.

[4] F. M. T. Brazier, B. J. Overeinder, M. van Steen, and N. J. E. Wijngaards. Agent factory: Generative migration of mobile agents in heterogeneous environments. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, SAC '02, pages 101–106, New York, NY, USA, 2002. ACM.

[5] L. Cabac, T. Dörges, M. Duvigneau, and D. Moldt. Requirements and tools for the debugging of multi-agent systems. In *Proceedings of the 7th German Conference on Multiagent System Technologies*, MATES '09, pages 238–247, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] R. Collier. Debugging agents in Agent Factory. In *Proceedings of the 4th International Conference on Programming Multi-Agent Systems*, ProMAS '06, pages 229–248, Berlin, Heidelberg, 2007. Springer-Verlag.

[7] J. Ding, D. Xu, X. He, and Y. Deng. Modeling and analyzing a mobile agent-based clinical information system. *The International Journal of Intelligent Control and Systems*, 10(2):143–151, 2005.

[8] N. Fukuta, T. Ito, and T. Shintani. Milog: A mobile agent framework for implementing intelligent information agents with logic programming. In *Pacific Rim International Workshop on Intelligent Information Agents*, 2000.

[9] H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2010: a toolbox for the construction and analysis of distributed processes. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS '11/ETAPS '11, pages 372–387, Berlin, Heidelberg, 2011. Springer-Verlag.

[10] A. R. Hurson, E. Jean, M. Ongtang, X. Gao, Y. Jiao, and T. E. Potok. Recent advances in mobile agent-oriented applications. In A. Y. Zomaya, editor, *Mobile Intelligence: Mobile Computing and Computational Intelligence*, Wiley Series on Parallel and Distributed Computing. John Wiley & Sons, Inc., 1st edition, 2010.

[11] ISO/IEC. *Information technology — Z formal specification notation —Syntax, type system and semantics*, 1 edition, 2002. ISO/IEC 13568:2002(E).

[12] T. Kawamura, S. Motomura, and K. Sugahara. Implementation of a logic-based multi agent framework on java environment. In *Proceedings of IEEE International Conference on Integration of Knowledge Intensive Multi-Agent Systems*, pages 486–491, 2005.

[13] D. N. Lam and K. S. Barber. Debugging agent behavior in an implemented agent system. In *Proceedings of the 2nd International Conference on Programming Multi-Agent Systems*, ProMAS '04, pages 104–125, Berlin, Heidelberg, 2005. Springer-Verlag.

[14] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. Request for Comments (RFC) 4122, 2005.

[15] S. Lynch and K. Rajendran. Breaking into industry: Tool support for multiagent systems. In *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, pages 136:1–136:3, New York, NY, USA, 2007. ACM.

[16] D. T. Ndumu, H. S. Nwana, L. C. Lee, and J. C. Collis. Visualising and debugging distributed multi-agent systems. In *Proceedings of the 3rd Annual Conference on Autonomous Agents*, AGENTS '99, pages 326–333, New York, NY, USA, 1999. ACM.

[17] F. Oquendo. π-ADL: an architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. *ACM Software Engineering Notes (SEN)*, 29(3):1–14, 2004.

[18] Oracle Corporation. Object Serialization. Web, 2013. `http://docs.oracle.com/javase/6/docs/technotes/guides/serialization/`.

[19] A. Outtagarts. Mobile agent-based applications: a survey. *International Journal of Computer Science and Network Security (IJCSNS)*, 9(11):331–339, 2009.

[20] L. Padgham, M. Winikoff, and D. Poutakidis. Adding debugging support to the prometheus methodology. *Engineering Applications of Artificial Intelligence*, 18(2):173–190, 2005.

[21] K. Taguchi and J. Song Dong. An overview of Mobile Object-Z. In *Proceedings of the 4th International Conference on Formal Engineering Methods: Formal Methods and Software Engineering (ICFEM 2002)*, pages 144–155, 2002.

[22] K. Taguchi and J. Song Dong. Formally specifying and verifying mobile agents — model checking mobility: the MobiOZ approach. *International Journal of Agent-Oriented Software Engineering*, 2(4):449–474, 2008.

[23] D. Takuo, T. Yasuyuki, and H. Shinichi. IOM/T: An interaction description language for multi-agent systems. In *Proceedings of the 4th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2005)*, pages 778–785, 2005.

[24] I. Toru. Q: A scenario description language for interactive agents. *IEEE Computer*, 35(11):42–47, 2002.

[25] I. Čavrak, A. Stranjak, and M. Žagar. SDLMAS: A scenario modeling framework for multi-agent systems. *Journal of Universal Computer Science*, 15(4):898–925, 2009.

[26] G. Vigueras and J. A. Botia. Tracking causality by visualization of multi-agent interactions using causality graphs. In *Proceedings of the 5th International Conference on Programming Multi-Agent Systems*, ProMAS '07, pages 190–204, Berlin, Heidelberg, 2008. Springer-Verlag.

[27] D. Xu, J. Yin, Y. Deng, and J. Ding. A formal architectural model for Logical Agent Mobility. *IEEE Transactions on Software Engineering*, 29(1):31–45, 2003.