PROCEEDINGS OF THE WORKSHOP ON APPLICATIONS OF SOFTWARE AGENTS
ISBN 978-86-7031-188-6, pp. 52 - 58, 2011

An overview of agent mobility in heterogeneous environments

Dejan Mitrović[1], Mirjana Ivanović[1], Zoran Budimac[1], and Milan Vidaković[2]

[1] Department of Mathematics and Informatics, Faculty of Sciences,
University of Novi Sad, Serbia
{dejan, mira, zjb}@dmi.uns.ac.rs
[2] Faculty of Technical Sciences, University of Novi Sad, Serbia
minja@uns.ac.rs

Abstract. Mobility is an important feature of some software agents, allowing them to leave their host and continue task execution on another machine in the network. For the mobility to work, however, the network needs to be comprised of interoperable multi-agent systems. Currently, there exists a large number of multi-agent system solutions which, unfortunately, do not satisfy this requirement, due to standards incompliance, different implementation technologies, and so on. This paper provides an overview of existing approaches to solving this problem, and enabling seamless agent mobility in heterogeneous environments.

1 Introduction

Agent technology represents one of the most consistent approaches to distributed software development. And although there is no generally agreed upon definition of the term, *software agents* can be described as executable software entities with various degrees of intelligence, that act autonomously in order to reach their design objectives.

Agents live in an environment which enables them to execute their tasks. These environments, called *multi-agent systems* (*MAS*), control the agent lifecycle, incorporate security mechanisms that protect both the agent and the environment itself, provide the inter-agent communication infrastructure, and so on.

An important feature of some software agents is *mobility*. Mobility enables an agent to (physically) leave its current host machine and continue the execution on another machine in the network. The mobility feature can be used to provide efficient and elegant solutions for a large variety of problems. From the past experience of the authors, it is worth noting the application of mobile agent in workflow [23, 6] and document management [7] systems.

In general, the process of agent migrating to another location involves several steps [10]:

1. Suspending the agent's execution flow
2. Saving the runtime state

3. Transferring the code and state to the target machine
4. Restoring the runtime state at the target machine and resuming the execution flow

Two basic types of mobility can be distinguished, based on what is assumed to be the agent's "runtime state": *weak* and *strong*. The systems supporting weak mobility save and restore only the runtime values of (all or some of) the agent's properties. The agent's execution is resumed on the target machine by, for example, automatically sending a pre-defined message to the agent. Strong mobility, on the other hand, includes saving and restoring the entire runtime state of the agent, including the execution stack, the program counter, and so on. Strong mobility is more transparent than weak, in the sense that agent is completely unaware of it. However, it is technically more difficult to implement, requiring support in the underlying implementation platform (e.g. modified Java Virtual Machine [20]).

Currently, there exists a large number of multi-agent systems supporting either weak or strong agent mobility. The most notable examples include *JADE* [13] and *Aglets* [2], supporting weak, as well as *D'Agents* [8] and *NOMADS* [15], supporting strong mobility.

Over the time, there have been several attempts to standardize the architecture and design of multi-agent systems. One of the most important goals was to assure interoperability between systems developed by different vendors. The two notable standardization attempts are the *FIPA* specification set [9] and the *OMG MASIF* specification [16]. The key issue among existing *MAS* solution, however, is exactly the lack of interoperability. According to [21]: "Two mobile agent systems are interoperable if a mobile agent of one system can migrate to the second system, the agent can interact and communicate with other agents (local or even remote agents), the agent can leave this system, and it can resume its execution on the next interoperable system". The lack of interoperability arises as a consequence of standards incompliance, usage of different implementation technologies, different sets of APIs offered to hosted agents, etc. This is a severely limiting factor in agent development, and in the wide-spread use of agent technology.

Several approaches have been developed in order to solve problems arising from incompatibility between existing *MAS* solutions. These approaches are discussed in the following section.

2 Approaches to agent mobility

According to an analysis published in [3], based on the types of multi-agent systems that appear on the agent's migration path, several types of mobility can be distinguished:

– *Homogeneous*. This assumes that all *MAS* nodes on an agent's path are based on the same implementation technology (e.g. Java), offer the same set of APIs, etc. No changes to the agent's code are needed during the migration. Obviously, this type of migration is the easiest to handle.

– *Cross-platform*. The agent migrates between different multi-agent systems, but all of which are implemented using the same platform. This means that what needs to adapted is the way in which the agent makes API calls to its hosting *MAS*; its executable code remains the same.
– *Agent-regeneration*. The agent migrates between instances of the same *MAS*, but running on different virtual machines or platforms. As a consequence, the agent's executable code needs to be regenerated at each host.
– *Heterogeneous*. Nodes in the agent's migration path are instances of different types of multi-agent systems, and are running on different virtual machines or platforms. This scenario requires regeneration of the agent's executable code as well as modification of the API calls it makes to the hosting *MAS*.

In practice, although there is a large number of available *MAS* implementations, the majority is developed using Java and running on Java Virtual Machine. The key problem to solve would, therefore, be defined by the cross-platform mobility pattern. Much more flexibility, however, is offered by agent-regeneration and heterogeneous mobility types.

2.1 Cross-platform mobility

The most common approach to solving the problem of interoperability in the cross-platform mobility pattern is through a form of *software layering*. The core idea of this approach is to build abstraction layers on top of existing multi-agent systems. The abstraction layer serves as an intermediary between an agent and its host *MAS* – it offers a unique set of API to the agent, and translates agent's calls to these functions into "native" calls to the underlying *MAS*.

Grid Mobile-Agent System (*GMAS*) [1] is a system that introduces three software layers in order to support cross-platform *MAS* interoperability:

– *GMAS* API: the common API, placed between an agent and a *MAS*
– *Foreign2GMAS*: a translation layer, with the task of translating agent's native calls into calls to the *GMAS* API
– *GMAS2Native*: a translation layer which transforms agent's calls to *GMAS* API into calls to the native platform

A *MAS* that wants to be able to dispatch its agents to other systems needs to offer an implementation of the *Foreign2GMAS* layer. Similarly, a *MAS* that wants to be able to host agents from other systems needs to include a *GMAS2Native* module. In this setting, an agent leaves its home *MAS* through a *Gateway* component. The target *MAS* uses a *Launcher* component to restore the agent once it arrives, and then remotely loads the *Foreign2GMAS* layer from the agent's home *MAS*. All API calls made by the agent are intercepted by this layer, transformed into *GMAS* API calls, and then forwarded to the local *MAS* through its own *GMAS2Native* module.

An approach presented in [18] asserts that even though multi-agent systems are indeed incompatible, they all approach the agent development process in the same manner – an agent is implemented by inheriting some base *agent* class. The

base class is further assumed to encompass all the interaction between the agent and its native *MAS*. Based on these assumptions, migration interoperability can be achieved by splitting the agent implementation into a platform-independent part, called the *head*, and a platform-dependent part, called the *body*. The core functionality of the agent is included in its head. A body implementation is provided for each of the supported *MAS*s. The body offers some common API and, in addition, inherits the base agent class of its host platform. Therefore, when an agent needs access to some of the underlying services, it simply invokes the "local" body implementation through the well-known set of methods.

Java-based Interoperable Mobile Agent Framework (*JIMAF*) [11] is another system based on software layering for enabling the cross-platform agent mobility. It features an *Interoperable Mobile Agent* (*IMA*) model, according to which each agent is split into a platform-dependent and a platform-independent part, similarly to the *head-body* division described earlier. In addition, *JIMAF* consists of three layers:

- *Interoperable Mobile Agent Layer*: hosts platform-independent parts of agents
- *Adaptation Layer*: handles creation, migration, and lookup of agents, as well as the transfer of communication messages across heterogeneous environments
- *Platform-dependent Mobile Agent Layer*: hosts platform-dependent parts of agents and is re-implemented for each of the supported *MAS*s

JIMAF is publicly available [14], very well documented (see e.g. [11, 12]), and extensively benchmarked; when compared to *GMAS*, for example, it is reported to introduce significantly less overhead to the agent migration process [11].

2.2 Agent regeneration

Although effective, the solution described in the previous sub-section focus only on Java-based multi-agent systems. Java does, in fact, represent an excellent technology for agent development, due to its cross-platform nature, extensive support for network programming, serialization features, and so on. However, the use of other development technologies might prove to be useful in different domains, e.g. when performance is of the essence. See, for example, *Mobile-C* [22] which is implemented in C/C++, and is reportedly two times faster than *JADE* when it comes to agent mobility.

SOA-based MAS (*SOM*) [5] is an extensible, service-oriented, *FIPA*-compliant multi-agent system developed by authors. It is defined as a conceptual specification of web services, each of which is dedicated to handling a distinct part of the overall agent-management process. The most important benefit of the *SOA*-based design is increased interoperability: external clients and third-party tools can utilize the power of agent technology through web service interfaces, i.e. in a familiar fashion, and using the standardized communication protocol (*SOAP*).

However, there is a major downside of the *SOA*-based design. Because any modern programming language can be use to implement *SOM*, developing an

agent that can execute on any running instance of the system becomes almost an impossible task. This is the classic example of the agent regeneration problem described earlier. The solution proposed in [5] is to use a new agent-oriented programming language named *Agent LAnguage for* SOM (*ALAS*). Besides providing developers with programming constructs that hide the complexity of agent development, the new language offers agent regeneration through *hot compilation*. That is, when an agent arrives at a *SOM* instance written in programming language *X*, its *ALAS* source code is transformed on-the-fly into the *X* source code. This output is then fed into a "native" compiler, producing executable agent code for the target platform.

2.3 Heterogeneous mobility

The most complex agent mobility scenario includes an agent migrating across a network of completely different multi-agent systems – with incompatible APIs and running on different virtual machines or platforms. *Generative migration* [3] is one proposed solution for enabling heterogeneous agent mobility. Rather than on software layering, it relies on a pool of agent *building blocks*, platform-independent descriptions of reusable functional components. Each building block is characterized solely by a description of its interface, i.e without any details regarding the implementation. An agent is defined (or, rather, designed) by assembling and interconnecting these building blocks into a so-called *agent blueprint*. During the migration process, the agent's blueprint is transferred, along with its runtime state. Using these information, an *agent factory* can rebuild the agent's executable code, on-the-fly, specific to the underlying *MAS*.

Model-Driven Engineering (*MDE*) [4] is a promising software development methodology which simplifies the software design process, and increases the productivity by overcoming the incompatibility problems of different systems. The most notable realization of *MDE* is *OMG*'s *Model-Driven Architecture* (*OMG MDA*) [17], which includes several specifications: *Platform-Independent Modeling* (*PIM*), model querying, viewing, and transformations (*Meta-Object Facility Query/View/Transformations*, *MOF QVT*), *XML Metadata Interchange* (*XMI*), etc.

It has been recognized [19] that there is a direct mapping of generative migration components into the *MDA* concepts: agents could be defined using *PIM*s, executable code can generated from model transformations, while the transfer of definitions can be performed through the use of *XMI*. The repository of *PIM*'s can be kept by a *MOF*. This opens the door for standardizing the solution to heterogeneous mobility problems and enabling the wide-spread use of the generative migration approach.

3 Conclusion

Mobility enables software agents to physically leave their current host and continue task execution on another machine in the network. Currently, there exists

a large number of multi-agent systems that support agent mobility. However, often it is almost impossible for an agent written specifically for one *MAS* to migrate to a *MAS* developed by another vendor, due to the lack of interoperability. Over the years, a number of approaches have been proposed, trying to alleviate this issue.

This paper has provided an overview of solutions that have proven to be successful in solving problems specific to cross-platform, agent-regeneration, and heterogeneous migration patterns. Cross-platform agent mobility can be enabled through a form of software layering, often in combination with the separation of agent's code into a platform-independent and a platform-dependent part. The presented solution for agent-regeneration includes a new agent-oriented programming language and an on-the-fly compilation for the concrete target *MAS*. The most complex of all is heterogeneous mobility, which could be handled by using agent blueprints – recognized as a realization of *MDE*.

References

1. A. Grimstrup, R. S. Gray, D. Kotz, M. M. Carvalho, T. B. Cowin, D. A. Chacón, J. Barton, C. Garrett, and M. Hofmann, "Toward iInteroperability of mobile-agent systems", In *International symposium on mobile agents*, pp. 106–120, 2002.
2. Aglets Homepage, `http://aglets.sourceforge.net/`, Retrieved on June 15, 2011.
3. B. J. Overeinder, D. R. A. De Groot, N. J. E. Wijngaards, and F. M. T. Brazier, "Generative mobile agent migration in heterogeneous environments", In *Scalable computing: practice and experience*, 7(4):89–99, 2006.
4. D. C. Schmidt, "Model-driven engineering", Published by IEEE Computer Society, pp. 25–31, February 2006.
5. D. Mitrović, M. Ivanović, and M. Vidaković, "Introducing ALAS: a novel agent-oriented programming language", In *Symposium on computer languages, implementations and tools, SCLIT 2011*, Greece, September 19–25, 2011.
6. D. Pešović, M. Ivanović, and Z. Budimac, "Implementation of advanced workflow patterns in a workflow management system using mobile agents", In *Proceedings of the international conference on software engineering theory and practice, SETP-07*, pp. 205–212, 2007.
7. D. Pešović, M. Vidaković, M. Ivanović, Z. Budimac, and J. Vidaković, "Usage of agents in document management", In *Computer science and information systems, ComSIS*, 8(1):193–210, 2011.
8. D'Agents Homepage, `http://agent.cs.dartmouth.edu/`, Retrieved on June 15, 2011.
9. FIPA Homepage, `http://www.fipa.org`, Retrieved on June 15, 2011.
10. G. Cabri, Z. Leonardi, and F. Zambonelli, "Weak and strong mobility in mobile agent applications, In *Proceedings of the 2nd international conference and exhibition on the practical applications of Java (PA-Java'2000)*, 2000.
11. G. Fortino, A. Garro, and W. Russo, "Achieving mobile agent systems interoperability through software layering", In *Information and software technology*, 50(4):322–341, 2008.
12. G. Fortino, A. Garro, and W. Russo, "Programming heterogeneous agent-based applications through the JIMAF: a case study", In *Proceedings of languages, methodologies and development tools for multi-agent systems (LADS 2007)*, Durham, UK, 4–6 September 2007.

13. JADE Homepage, `http://jade.tilab.com/`, Retrieved on June 15, 2011.
14. JIMAF Homepage, `http://lisdip.deis.unical.it/software/jimaf/index.html`, Retrieved on June 15, 2011.
15. NOMADS Homepage, `http://www.ihmc.us/research/projects/nomads/`, Retrieved on June 15, 2011.
16. OMG MASIF Specification, `http://www.omg.org/cgi-bin/doc?orbos/97-10-05`, Retrieved on June 15, 2011.
17. OMG MDA Homepage, `http://www.omg.org/mda/`, Retrieved on June 15, 2011.
18. P. Misikangas, and K. Raatikainen, "Agent migration between incompatible agent platforms", In *Proceedings of the The 20th international conference on distributed computing systems, ICDCS 2000*, pp. 4–10, 2000.
19. T. Gherbi, I. Borne, and D. Meslati, "MDE and mobile agents: another reflection on the agent migration", In *Proceedings of the 11th international conference on computer modelling and simulation*, pp. 468–473, 2009.
20. N. Suri, J. Bradshaw, M. Breedy, P. Groth, G. Hill, and R. Jeffers, "Strong mobility and fine-grained resource control in NOMADS", In *Proceedings of the Joint symposium on agent systems and applications/mobile agents (ASA/MA 2000)*, Zurich, Switzerland, pp. 2–15, 2000.
21. U. Pinsdorf, and V. Roth, "Mobile agent interoperability patterns and practice", In *Proceedings of the 9th IEEE international conference on engineering of computer-based systems*, pp. 238–244, 2002.
22. Y.-C. Chou, D. Ko, and H. H. Cheng, "An embeddable mobile agent platform supporting runtime code mobility, interaction and coordination of mobile agents and host systems", In *Information and software technology*, 52:185–196, 2010.
23. Z. Budimac, M. Ivanović, and A. Popović, "Workflow management system using mobile agents", In *Proceedings of the 3rd East-European conference on advances in databases and information systems, ADBIS 1999*, pp. 168–178, 1999.