Generative Migration of Agents

F.M.T. Brazier; B.J. Overeinder; M. van Steen; N.J.E. Wijngaards

Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam;
de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{frances,bjo,steen,niek}@cs.vu.nl

Abstract

Agents, and in particular mobile agents, offer a means for application developers to build distributed applications. In current agent systems, mobility of agents is constrained by the environment of the agents: the agent platform (which supports agents) and the agent's code base (e.g., DESIRE, Java). Generative migration is needed to adapt an agent to conform to its destination agent platform and code base. In this paper generative migration is described as a process of "transparently adapting" an agent. An agent can continue to function at its new location on a completely different agent platform.

1 Introduction

Agents, and in particular mobile agents, offer a means for application developers to build distributed applications. In current agent systems, mobility of agents is constrained by the environment of the agents: the agent platform (which supports agents) and the agent's code base (e.g., DESIRE, Java). Within the same agent platform and code base, agent migration has been shown to be possible. However, many agent platforms exist, differing substantially in the support for agents. Write once - run everywhere is not yet true for agents...

This heterogeneity of agent platforms, combined with heterogeneity in code-bases of agents, leads to an interesting question concerning agent mobility: can an agent migrate across heterogeneous platforms? The answer is relatively simple, as an agent needs to be adapted to fit its destination agent platform and code-base.

In this paper generative migration (see Brazier et al. (2002)) is described as a process of "transparently adapting" an agent. Section 2 presents an overview of the principles behind agent factories: the entities responsible for processing blueprints. Section 3 describes the use of the agent factory for generative migration. Section 4 investigates implications of generative migration for agents and agent factories. Section 5 discusses transparent adaption by generative migration.

2 Agent Factory

An agent factory is a facility that creates, and modifies, software agents, see Brazier and Wijngaards (2001). It can be used to adapt agents so that they can use specific programming languages and run on different agent platforms. The design of an agent within an agent factory is based a *blueprint*. The blueprint of an agent contains a configuration of conceptual building blocks which specifies the agent's functionality and behaviour. The blueprint also contains one or more configurations of detailed building blocks, which specify an operationalisation of the conceptual functionality and behaviour.

A mapping is defined between building blocks at conceptual level and building blocks at detailed level. (Note that this mapping may be structure preserving, but that this is not necessarily ideal.) A detailed description of a building block includes the operational code. For each conceptual description, a number of detailed descriptions may be devised and vice versa. These detailed descriptions may differ in the operational language (e.g., C++, Java), but also in, for example, the efficiency of the operational code. The conceptual descriptions may differ in the modelling paradigm (e.g., UML, DESIRE), but also in, e.g., the detail in which an agent's functionality is modelled.

Building blocks themselves are configurable, but cannot be combined indiscriminately. The open slot concept is used to regulate the ways in which components are combined. An open slot in a component has associated properties at both levels of abstraction that prescribe the properties of the building block to be "inserted".

A prototype of the agent factory automatically (re-)designs an information retrieval agent: its blueprint and executable code. The blueprint of an existing simple information retrieval agent is briefly described by Brazier et al. (2002). This prototype agent factory itself is written in Java, and contains enough knowledge to (re-)design simple information retrieval agents.

3 Generative Migration

One of the strengths of the agent factory concept is that it provides a means to support migration of agents in heterogeneous environments. Section 3.1 discusses pre-conditions for successful migration of agents. Section 3.2 describes the approach in agent-factory-enhanced migration. Section 3.3 describes migration scenarios.

3.1 Migration pre-conditions

A mobile agent is simple an agent having the ability to move between different machines, e.g. see Tanenbaum and van Steen (2002). Agent migration entails transfer of both the agent's executable code and state. The concept of generative migration is geared to weak mobility: parts of the state of the agent are migrated to another host. Strong mobility, i.e. migrating running processes (including their memory usage, stack, heap, etc.), is not possible with generative migration, as in most cases agent executables change.

Generative migration requires (1) agent factories, (2) implementation independent representations of agent functionality, and (3) implementation independent formats of agent's state (e.g., XML, RDF or OIL may be used, see Horrocks et al. (2001)).

Assuming that both the source and the destination host both have access to an agent factory (for simplicity's sake), these agent factories need to have building blocks with comparable functionality. One solution is that the agent factories share the same libraries of conceptual building blocks, but each have different libraries of detailed building blocks. Another solution is to have conceptual building blocks in ZEUS (see Nwana et al. (1999)) and DESIRE (see Brazier et al. (1997)) with comparable functionality.

3.2 Migration using the agent factory

Migration using an agent factory diverges from standard mobility of agents in that it is *not* executable code with state that is migrated, but instead the agent's blueprint together with (parts of) the agent's state. Consider the following scenario for heterogeneous mobility (also described by Brazier et al. (2002)), depicted in figure 1.

An information retrieval agent A currently resides on host machine $H1$. This host runs the Ajanta agent platform, developed by Tripathi et al. (1999), and supports Java agents. The agent wants to move to another host: host $H2$. Host $H2$ runs the DESIRE platform, and its agents run code generated by the DESIRE execution environment, see Brazier et al. (1997).

In the process of migrating the agent A from host $H1$ to host $H2$, the agent first needs to store information on its (mental) state. Then the agent factory on host $H1$ sends the blueprint of the agent, together with the state information of the agent to the agent factory at host $H2$.

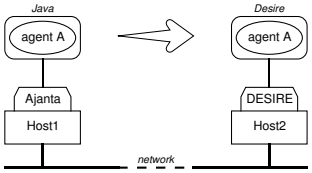

Figure 1: Example migration scenario in which agent A on Host 1 (written in Java, running on Ajanta) migrates to Host 2 (where it will be specified in DESIRE and running on DESIRE).

Host $H2$'s local agent factory receives the blueprint of the agent and state information. This agent factory constructs a DESIRE agent A on the basis of the blueprint of agent A. This DESIRE agent A (i.e., a functionally equivalent incarnation of the Java agent A) runs on DESIRE's virtual machine (the DESIRE-interpreter), and is able to incorporate information on its state.

3.3 Migration scenarios

Migration including re-generation of agents is a more complex process, requiring more resources, than migration without agent re-generation. Four migration scenarios are distinguished in Brazier et al. (2002):

- *Homogeneneous migration.* An agent migrates to another host without any changes in either the virtual machine or the agent platform. This situation is most common in practice, and does not warrant generative migration.
- *Cross-platform migration.* An agent migrates to another host with the same virtual machine, but a different agent platform. Generative migration may be used to adapt the agent to the target agent platform, e.g. by using wrapper interfaces. If both agent platforms have the same standard interface (e.g., advocated by OMG, see OMG (2000), and FIPA, see FIPA (2001)), the agent need not be adapted.
- *Agent-regeneration migration.* An agent migrates to a host with a different virtual machine, but the same agent platform. Generative migration is needed to regenerate the agent's executable code for the target virtual machine and the agent platform.
- *Heterogeneous migration.* An agent migrates to a host with a different virtual machine and a different agent platform (see the scenario described in Figure 1. In this situation, generative migration is needed to regenerate an agent for the target virtual machine and the target agent platform.

The authors are unaware of agent platforms supporting agent-regeneration migration and/or heterogeneous migration.

4 (Re)Generation versus Adaptation

Generative migration has implications for both agents and agent factories. Not only do agents need to understand the concept of *locality*, but also the concept of *incarnation*. Section 4.1 discusses implications for the role of agent factories. Section 4.2 briefly describes aspects of agent incarnations.

4.1 Role of Agent Factories

Agent factories are responsible for (re)generating an agent, while adhering to preferences of the agent and the (destination) agent platform. An incarnation of an agent needs to be designed which may be executed by a virtual machine at the target host, and which can interface with the agent platform at the target host. The agent (re)generative process is responsible for minizing the quantity and quality of the changes to the agent. The process of regenerating an agent mainly depends on available libraries of building blocks.

The agent regeneration process is facilitated by the two levels of abstractions distinguished within the agent factory: conceptual and detailed. In general, agent factories need to have building blocks with comparable functionality. In the general situation, a configuration of conceptual building blocks needs to be constructed. Agent factories sharing common libraries of conceptual building blocks (with equal functionality) is a specific case, providing common ground to the agent factories involved. In this case, the configuration of conceptual building blocks need not be changed. If an applicable configuration of detailed building blocks is present for the target host, the agent factory only needs to assemble this configuration into operational code. In this case the adaptation is *transparent* to the agent: the agent need not be aware of the fact that it has another incarnation than before.

When no suitable configuration of detailed building blocks is available, a configuration of detailed building blocks may need to be (re-)designed. The agent factory needs strategic knowledge to decide on a configuration of detailed building blocks which minimizes the loss of functionality and services for the agent. In addition, the agent factory may provide an agent with functionality to cope with the loss of specific functionality.

An agent may be fitted with functionality for introspection, awareness of its abilities, and understanding functionality and services needed to achieve (its) goals. Such an agent can adapt its behaviour, and pursual of goals, with respect to its current incarnation.

4.2 Agent incarnations

The incarnation of an agent is entails activation of both the "code and data" of the agent in another environment (the virtual machine possibly with another interface to an agent platform.). Figure 2 depicts the concepts related to

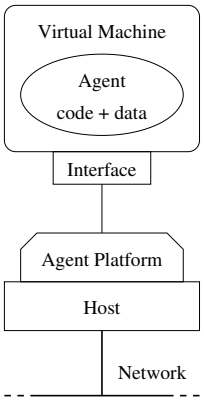

Figure 2: An agent's incarnation involves not only its code and data, but also its immediate physical environment.

the incarnation of an agent. An agent's code and data is executed in the context of a virtual machine; examples are the Java Virtual Machine and Prolog interpreters. The agent has access to an interface to its agent platform via its virtual machine. Through this interface the agent can access services provided by the agent platform, e.g. (group) communication, generative migration, etc.

The agent needs to conform to both its target virtual machine and its target agent platform interfaces. For reasons of privacy and security, it is assumed that the state of an agent does not need to be adapted: an agent can easily employ a programming language independent representation format for its state (e.g., on the basis of XML, RDF or OIL, see Horrocks et al. (2001)). The (re)generation process has the goal to generate an operationalisation of the (conceptual) functionality of the agent.

In the ideal adaptation process, the agent does not "notice" any changes to its incarnation. It still has access to all of its functionality, can resume execution from its state, and can access services provided by its current agent platform. When agent platforms, and virtual machines, differ extensively, it is up to the agent factory to mask these differences.

In less ideal situations, an agent may be aware of classes of services and functionality which may be unavailable during/for specific incarnations. The agent needs to adapt to this situation, e.g. by employing strategies for circumventing missing functionality and services (e.g., enlisting support by other agents).

An agent may specify preferences concerning its incarnation. With these preferences, an agent can specify, e.g., which functionality and/or services are of more importance to its (correct) functioning than other functionality and/or services. An agent may, in addition, specify that specific functionality and/or access to specific services may be (temporarily) unnecessary at a specific host.

The preferences specified by an agent may also state how the agent is to be informed of success or failure of generative migration. Does the agent expect a message

in a specific format? Does the agent expect information concerning its current (dis)abilities as a facts-base?

Security and trust are of importance in generative migration. The agent trusts an agent factory to generate the "right" incarnation. An agent cannot be easily protected against a malicious agent factory, which e.g. may introduce code to spy on the agent. The administrator of an agent platform trusts its agent factory and libraries to generate agents which cannot damage other agents, the agent platform or the hosts running the agent platform and agents.

5 Discussion

Mobile agents are currently restrained in their mobility by their environment. Current agent platforms expect a homogeneous environment, i.e. hosts running the same agent platform and the same virtual machine. This paper proposes an approach which transcends homogeneity of agent platforms and virtual machines: *generative mobility*. In generative mobility, a blueprint of an agent's functionality is transported, together with information on the agent's state. At its destination, an agent factory regenerates the executable code of the agent on the basis of its blueprint: a new incarnation of the agent. Upon activation, the agent may restore its state and resume execution.

Ideally, an agent factory is able to (re)generate an agent such that it retains all of its functionality and access to services: transparent adaption. However, this may not be possible in situations requiring heterogeneous migration. An agent needs to be aware of characteristics of its current incarnation, including limitations in functionality provided by its current incarnation and services offered by the current agent platform.

Generative migration is one of the services researched within the AgentScape project on worldwide scalable distributed agent operating systems. Currently a prototype of the agent factory (namely the libraries of components) is being built that supports generative mobility.

Acknowledgements

This research is supported by NLnet Foundation, http://www.nlnet.nl. The authors wish to acknowledge the contributions made by Hidde Boonstra, David Mobach, Oscar Scholten and Sander van Splunter.

References

F. M. T. Brazier, B. D. Dunin-Keplicz, N. R. Jennings, and J. Treur. Desire: Modelling multi-agent systems in a compositional formal framework. *International Journal of Cooperative Information Systems*, 6:67–94, 1997.

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 AIMS Workshop at SAC 2002*, 2002. to appear.

F. M. T. Brazier and N. J. E. Wijngaards. Automated servicing of agents. *AISB journal*, 1(1):5–20, 2001.

FIPA. FIPA agent platform, 2001. http://www.fipa.org.

I. Horrocks, F. van Harmelen, P. Patel-Schneider, T. Berners-Lee, D. Brickley, D. Connoly, M. Dean, S. Decker, D. Fensel, P. Hayes, J. Heflin, J. Hendler, O. Lassila, D. McGuinness, and L.A. Stein. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html, March 2001.

H. Nwana, D. Ndumu, L. Lyndon, and J. Collis. ZEUS: A toolkit and approach for building distributed multi-agent systems. In *Proceedings of the Third International Conference on Autonomous Agents (Autonomous Agents'99)*, pages 360–361, 1999.

OMG. Mobile agent facility specification. OMG Document formal/00-01-02, Object Management Group, Framingham, MA, January 2000.

A. S. Tanenbaum and M. van Steen. *Distributed Systems: Principles and Paradigms*. Prentice Hall, Upper Saddle River, New Jersey 07458, 2002.

A. Tripathi, N. Karnik, M. Vora, T. Ahmed, and R. Singh. Mobile agent programming in Ajanta. In *Proceedings of the 19th International Conference on Distributed Computing Systems (ICDCS'99)*, pages 190–197, Austin, TX, May 1999.