

# Adaptability in Mobile Agent Systems using Reflection

Thomas Ledoux & Noury M.N.Bouraqadi-Saâdani

École des Mines de Nantes

4 rue Alfred Kastler

F-44307 Nantes cedex 3, France

[Thomas.Ledoux@emn.fr](mailto:Thomas.Ledoux@emn.fr) & [Noury.Bouraqadi@emn.fr](mailto:Noury.Bouraqadi@emn.fr)

## Abstract

We present in this paper our current investigations dealing with reflection for code mobility. Our study is about reifying different concerns of mobile agent systems in order to ease their customization. Reflection enables the construction of open mobile agent systems and helps separation of concerns. Following previous ideas introduced for reflective middleware, we consider run-time adaptability of agent mechanisms. The main idea is that network introspection can be used to dynamically choose the best execution policy. Based on some well-known design patterns, our approach enables this run-time adaptability of the infrastructure of mobile agent systems.

## 1 Introduction

Reflection is the process of reasoning about and acting upon itself [Smi82] [Mae87]. Several works have shown the benefits of reflection for building open distributed systems [Bla98] [Hay98] [Kon98] [Led99]. Indeed, reflection allows deriving new behaviors from initial ones by introducing some variations of the computational model. Separation of concerns, extensibility and flexibility are some of the main advantages one can expect from reflection.

Technologies and architectures traditionally used to develop distributed applications exhibit a variety of drawbacks when applied to large scale distributed settings such as Internet and telecommunications (e.g. high net latency). Code mobility [Fug98] is a promising approach to address these issues. In this position paper, we focus on the mobile agent paradigm [Lan98], which represents the most complex case of code mobility.

This position paper is presented as follows. In section 2, we expose the context of our study and focus on the agent transfer process. In section 3, we give some elements to build a reflective mobile agent system. We sketch out two examples of possible adaptability. In the first one, we propose to "open" mobile agent systems to supply different models of code mobility. In the second one, we introduce the dynamic adaptability of agent execution policies according to the network characteristics (e.g. network traffic, CPU performance). Finally, we draw a short conclusion.

## 2 Elements of a Mobile Agent System

A mobile agent is a software entity that has the ability to travel through the nodes of a network. When it moves from one host to another, the mobile agent system manages the agent transfer process usually following *fixed policies* (i.e. ad hoc policies which are proposed by the system and which cannot be changed). In this section, we briefly present two important protagonists in an agent transfer: agents themselves and their associated resources. We also mention the mobility mechanisms involved.

## 2.1 Agent

An *agent* [Fug98] [Lan98] can be viewed as the composition of:

- a *code* (which provides the static description for the behavior of a computation);
- a *data space* (which is the set of references to resources that can be accessed);
- an *agent state* (which contains private data that cannot be shared);
- an *execution state* (which is run-time state including program counter, call stack).

Existing mobile agent systems follow two forms of mobility, depending on the agent constituents that can be migrated. On the one hand mobile agent systems that support the migration of the execution state (*strong mobility*), and on the other hand, those that do not support it (*weak mobility*).

## 2.2 Resources

A resource represents an entity that can be shared between several agents, such as a system variable, a file, a printer, a display, etc. We can model a resource as a 3 value tuple *Resource* =  $\{I, V, T\}$  [Fug98], where:

- *I* is a unique identifier (e.g. currently used printer)
- *V* is the value of the resource (e.g. a laser printer 600 dpi)
- *T* is the type of the resource (e.g. a laser printer)

Resources can be bound to an agent through three forms of binding (*by identifier, by value, by type*), which constrain the data space management mechanisms that can be exploited upon migration. However, Fuggetta et al. notice that the nature of the resource determines also the data space management mechanisms. They distinguish different kinds of resources:

- Free transferable  
resource can be migrated over the network (e.g. a data file)
- Fixed transferable  
resource could be migrated over the network, but it is not the case (e.g. an open file, a big file, or a crucial file)
- Fixed not transferable  
resource cannot be migrated over the network (e.g. an OS printer handle)

In summary, the policies of resource relocation and binding reconfiguration that are involved in the data space management are constrained both by the nature of the resources and the forms of the binding to such resources. These relationships and the associated mechanisms are described in [Fug98].

## 3 Towards a Reflective Mobile Agent System

Based on a "black box" philosophy, traditional mobile agent systems do not allow the reification of the agent and resource entities, providing then *ad hoc* mechanisms. We are strongly convinced that these entities must be "opened" to allow adaptability of different models and mechanisms of code mobility. We propose to use reflection to reason about and act upon the agent transfer mechanism.

### 3.1 Code and execution state management adaptability

As we mentioned in 2.1, existing mobile agent systems support either strong mobility or (more commonly) weak mobility. However, for the same agent system, a given fixed policy could turn out not to be optimal for a particular application (e.g. strong mobility is not always needed). We propose to use reflection in order to provide some variations on the execution state management model.

More generally, other paradigms such as remote evaluation (e.g. SQL request) or code on demand (e.g. Java applets) can be seen as a specialization/optimization of the agent paradigm. Reflection allows for different granularities in the migration process to elaborate a fine-tuned model for code mobility. Table 1 presents the different possibilities of customization allowed by reflection.

<i>Entities to migrate</i>	<i>Policies</i>
Code only	Remote evaluation, code on demand
Code + agent state	Weak mobility
Code + agent state + execution state	Strong mobility

**Table 1-Models of code mobility**

### 3.2 *Data space management adaptability*

As we suggested in 2.2, Fuggetta et al. propose a framework to deal with the mechanisms of resource relocation and binding reconfiguration which are involved in the management of data space. For example, a *fixed transferable* resource, which is bound *by value* to an agent, can support a binding reconfiguration either *by copy* or *by network reference*.

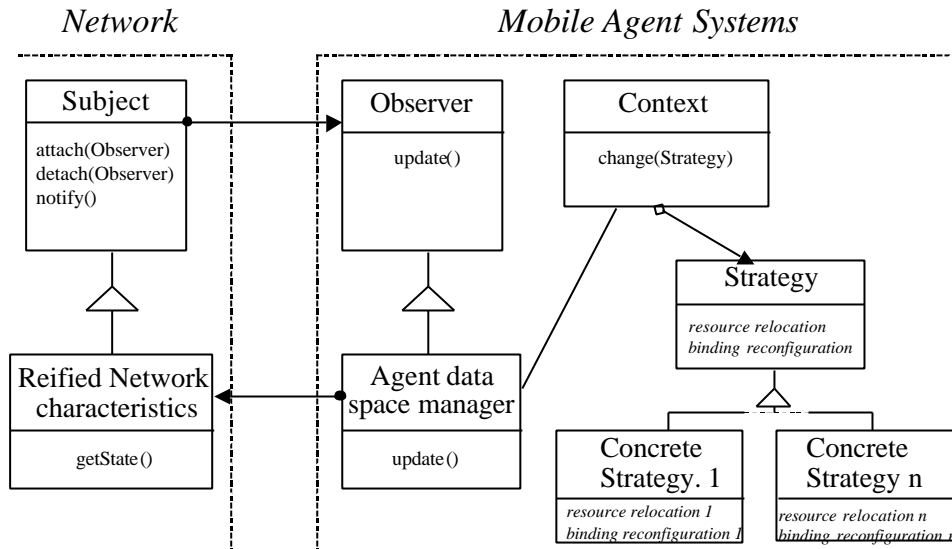
Reification of resources and relationships between agent and resources are fundamental to specify an open architecture. However, the choice of the suitable mechanism (e.g. by copy, by move, by reference) is still an open issue. A fixed policy can lead to unexpected performance and reliability problems if mobile agent systems are not aware of capacities of the underlying infrastructure. Network traffic, network reliability, CPU performance, access to memory and resources topology are subject to change, i.e. network characteristics are not defined statically.

Thus, we propose to introspect these characteristics in order to dynamically choose the best execution policy of resource relocation and binding reconfiguration. The following examples illustrate this idea for a *free transferable* resource:

- if we deal with low-reliable network (i.e. a disconnection can occur), mobile agent systems should avoid the “by reference” mechanism.
- if we deal with low-bandwidth network and moderate network traffic, mobile agent systems should avoid the “by move” mechanism.

With the hypothesis that network characteristics are reified, we propose an approach based on the composition of some well-known design patterns: Observer and Strategy [Gam95]. This composition was introduced in [Bra99] to encourage the dynamic behavior changes of an object, initiated by other objects.

In the solution we propose (cf. Figure 1), the Observer pattern is used to define a one-to-many dependency between the network reification and the agent/resources reification. When a network characteristic changes, it notifies the mobile agent system. Then, following the Strategy pattern, the mobile agent system updates its strategy by analyzing the context change and detects the best execution policy. In conclusion, our approach enables run-time adaptability of the infrastructure of mobile agent systems in order to guaranty a better Quality of Service (QoS).



**Figure 1-Composition of design patterns**

#### 4 Conclusion

In this position paper, we propose reflection to bring adaptability into mobile agent systems. In order to be flexible, execution policies of mobile agent systems should adapt to the heterogeneous communication infrastructure where network characteristics are not defined statically. We propose a general framework based on reflection and using some well-know design patterns, which allow an agent to be notified of modifications of the network characteristics in order to dynamically adapt its own execution policies.

#### References

- [Bla98] BLAIR G.S., COULSON G., ROBIN P., PAPATHOMAS M. — An Architecture for Next Generation Middleware. In *Proceedings of Middleware'98*, Springer-Verlag, p.191-206, N. Davies, K. Raymond, J. Seitz Eds, September 1998.
- [Bra99] BRAUX M., NOYÉ J. — Changement dynamique de comportement par composition de schémas de conception. In *Proceedings of LMO'99*, Hermès, Villefranche sur Mer, France, January 1999.
- [Fug98] FUGETTA A., PICCO G. P., VIGNA G. — Understanding Code Mobility, in *IEEE Transactions on Software Engineering*, 24(5), 1998.
- [Gam95] GAMMA E., HELM R., JOHNSON R., VLISSIDES John — *Design Patterns*, Addison-Wesley Reading, Massachusetts, 1995.
- [Hay98] HAYTON R., HERBERT A., DONALDSON D. — FlexiNet - A flexible component oriented middleware system. In *Proceedings of ACM SIGOPS European Workshop*, Sintra, Portugal, September 1998.
- [Kon98] KON F., SINGHAI A., CAMPBELL R. H., CARVALHO D., MOORE R., BALLESTEROS F. J — 2K: A Reflective, Component-Based Operating System for Rapidly Changing Environments. In *ECOOP'98 Workshop on Reflective Object-Oriented Programming and Systems*, Brussels, Belgium, July 1998.
- [Lan98] LANGE D. — Mobile Objects and Mobile Agents: The Future of Distributed Computing? In *ECOOP'98 Proceedings*, Brussels, Belgium, July 1998.
- [Led99] LEDOUX T. — OpenCorba: a Reflective Open Broker. In *Proceedings of Reflection'99*, LNCS 1616, p.197-214, Springer-Verlag, Saint-Malo, France, July 1999.
- [Mae87] MAES P. — Concepts and Experiments in Computational Reflection. In *Proceedings of OOPSLA'87*, ACM Sigplan Notices, p.147-155, Orlando, Florida, October 1987.
- [Smi82] SMITH B.C. — Reflection and Semantics in a Procedural Programming Language. PhD thesis, MIT, January 1982.