FOAM: Fragmented Objects for the Implementation of Mobile Agents

Martin Geier, F. J. Hauck

Mai 1999 TR-I4-99-04

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany

FOAM: Fragmented Objects for the Implementation of Mobile Agents

Martin Geier, Franz J. Hauck

{geier,hauck}@informatik.uni-erlangen.de

IMMD IV, University of Erlangen-Nürnberg
Martensstr. 1, D-91058 Erlangen, Germany

Abstract. Mobile agents are a promising approach to distributed software. The typical programming paradigm considers a mobile agent to be a mobile object or a group of objects migrating together. Important issues of this approach are abstraction, encapsulation, a well-defined interface, and the clear definition of a migration entity. Putting the whole functionality of an agent into one migration entity is not feasible for real world applications, because one important benefit of a mobile agent, namely the efficient communication at the destination site, does not compensate migration costs. FOAM, a fragmented object model for mobile agents, offers a solution to this problem. Agents are split into smaller and mobile fragments, which still belong to the same object and agent respectively. Agents can scale with respect to mobility as also just parts of the agent can migrate. Agents can have replicated components and have multiple points of presence. Finally, an agent can be fragmented according to the different phases of its task. This reduces migration costs if not all phases are needed for a certain run of the agent and increases the expenditure to spy out critical information of the agent.

1. Introduction

One of the most promising approaches to realize distributed software is the technique of mobile agents [White96]. A mobile agent is an entity, which is capable to migrate around in a distributed system while it is fulfilling its task instructed by the user. The typical migration entity of a mobile agent is an object of the object based programming paradigm or a whole group of objects migrating together. The advantages of this object-based approach are its qualities of encapsulation, abstractions and well defined interfaces. Further advantages of the object-based approach for mobile agents are the conceptionally easy integration of multithreading in form of active objects [Papa89] and the well defined migration entity, which is implied by the object itself.

There are many advantages proposed in the literature about mobile agents. One of the most important benefits that mobile agents can gain is the *low usage of bandwidth* by moving the computation to the data rather than the data to the computation. This allows the mobile agent to deal with vast volumes of data as it is typical in the area of information retrieval. However, mobile agents are still an area of research. Therefore, typical implementations of mobile agents are small and more or less pathological. To implement “real world” applications like Electronic Commerce or Personalized Server Behaviour [HCK95] agents will become larger and larger. Then, the question arises whether mobile agents, implemented as a single migration entity, are still the adequate programming paradigm. The consequence would be, that the migration costs become higher and the praised bandwidth reduction is lost.

The problem of large mobile agents could be solved with current techniques applying one of the following options:

- *Confine migration*. Decisions to migrate the mobile agent have to be done with respect to the size of the agent, e.g., the mobility of an agent could be restricted by using special design patterns specifying the way of the agent through the distributed system from the first. The obvious disadvantage is that the agent cannot decide itself where to go—especially the dynamic adaption by spontaneous migration decisions is prevented. This seems to be a contradiction to the model of mobile agents.
- *Keep agents small*. We could keep agents small by moving semantics into a nonmobile application at a source host which is connected to the mobile agent. The mobile agent therefore gets just a part of the work to be done, e.g., it collects and pre-filters data, which is then filtered and prepared at the source host.
- *Use many small agents*. The usage of many small agents seems to be a feasible approach for reducing bandwidth usage. Instead of one big mobile agent we use a swarm of small agents as in the SWARM simulation system [MBL+96]. These small agents define their own migration entities and the agents are loosely coupled. The advantage of this approach is the scalability of migration: if certain agents, which are distributed over different address spaces, communicate a lot with each other, they can be easily co-located in one address space as migration is cheap because of the small size of the agents. However, for using many agents a special framework is needed which regulates the distribution and optimises the communication between the agents. Until now there is no such framework available. Also, the swarm of agents is represented by multiple mobile objects which have separate identities. For the client of the swarm it is not clear which of the agents to ask for a progress report, etc. If the user wants to deal with one entity that controls the activity of the swarm there has to be a central component for that.

In this paper we present a solution to the problem of large mobile agents. Our approach is similar to the swarm of sub-agents, but maintains the single identity of one single agent. We use a fragmented object model called FOAM, which evolves the classical monolithic object model to the needs of distributed systems. The agent is split into multiple components called *fragments*, which can migrate independently. However, these fragments form a single object. Additionally, the agent may evolve over time by creating additional fragments or by deleting them.

This paper is structured as follows: In Section 2, we introduce our fragmented object model and the *AspectIX* middleware architecture, which implements this model. We will also show how the classical mobile agent is implemented in such an object model. Section 3 presents the additional possibilities of the fragmented object model for agents. Agents can scale with respect to mobility, can be replicated, and can have multiple points of presence. Finally, in Section 4 we draw our conclusions.

2. FOAM and the AspectIX Architecture

The **F**ragmented-**O**bject **A**gent **M**odel (FOAM) is based on the object model of the *AspectIX* architecture [HBG+98], which in turn is based on the object models introduced in *FOG* [MGN+94] and *Globe* [SHT97]. The traditional monolithic object model is extended for the needs of distributed systems. Whereas in an implementation of a monolithic model a single object resides at exactly one location at a time, a distributed object of a fragmented object model is split into multiple parts called fragments in FOG and *AspectIX*, and called local objects in Globe. These fragments can be distributed over different address spaces. They communicate with one another to implement the object semantics. A client using the object always needs a local fragment in its own address space, on which local method calls can be invoked. The fragment is a representative of the distributed object. It can implement the complete or partial object functionality, or forward any requests to a fragment with server semantics. With multiple fragments in different address spaces a distributed object can be present at multiple location at the same time.

Fragments can themselves be considered to be objects. The difference between a fragmented distributed object and a set of communicating objects is their identity. The fragments of a fragmented object belong to the one object they implement, i.e. they have the same object identity, whereas the objects of an object set have individual identities and are not related to each other for the underlying middleware system or any client using them.

2.1 The *AspectIX* Architecture

The *AspectIX* middleware architecture supports distributed objects on the basis of the aforementioned fragmented object model. *AspectIX* is CORBA-compliant [OMG98], i.e. a distributed object can be transparently accessed by any other CORBA object, and distributed *AspectIX* objects can access other CORBA objects. *AspectIX* objects are implemented by multiple distributed fragments whereas pure CORBA objects are implemented as a single server fragment and multiple stub fragments.

The fragments of a distributed object can communicate with one another using special mechanisms offered by the *AspectIX* Middleware, e.g., multicast messages, stream I/O, and IIOP-based RPC. The architecture also provides services for naming objects and locating fragments.

In *AspectIX,* a fragment is split into two parts (Fig. 1), a fragment interface and a fragment implementation. The fragment interface is a generic object that is automatically generated during the development process. It only depends on the IDL description of the interface of the distributed object, and its main purpose is to be a stable reference point which delegates method calls to the fragment implementation. The fragment implementation, on the other hand, implements the desired semantics of the fragment. This separation into fragment implementation and interface allows the fragment to exchange the fragment implementation without loosing the validity of the client reference, which is only bound to the fragment interface.

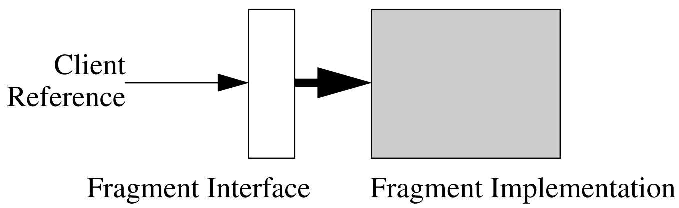

Fig. 1 Components of a Fragment

2.2 Conventional Agents and FOAM

FOAM uses the *AspectIX* platform for mobile agent systems. In this sub-section we will show how conventional mobile agents can be implemented with FOAM. We use the term *conventional* for mobile agents following the monolithic approach. In fact, monolithic mobile agents can be easily implemented with FOAM. The mobile agent is realized as one distributed object with a single fragment that implements the semantics of the mobile agent. All other fragments have stub functionality and allow remote access to the main agent fragment, which is located at exactly one host in the distributed system. The mobile agent migrates to another host by first extending itself to the destination address space and then by shrinking away from the source address space (see also [GSB+98]).

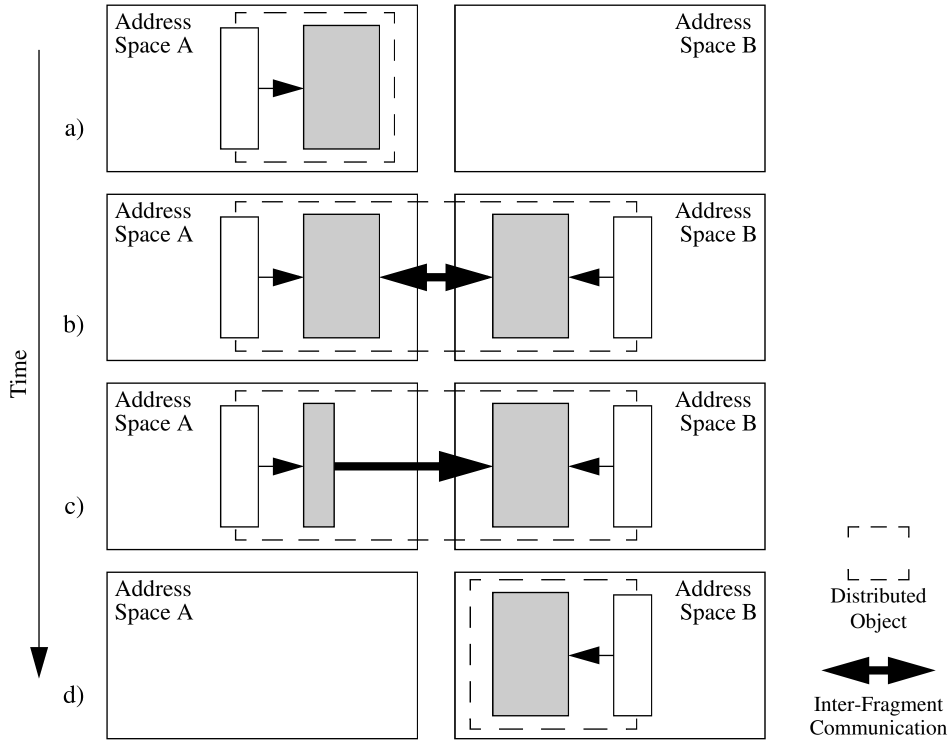

Fig. 2 Phases of the Migration of a Monolithic Agent in FOAM

The different phases of a migrating monolithic agent in FOAM are displayed in Figure 2. First, the agent consist of only one fragment implementing the functionality of the agent (Fig. 2a). Second, the agent extends itself by creating a new fragment at the destination site (Fig. 2b). Third, after transferring the whole state from the original fragment to the new one, the old fragment can be replaced by a simple stub acting as a forwarding entity (Fig. 2c). Finally, if no further access to the mobile agent is needed from the original site, the fragment on this site can be deleted (Fig. 2d).

On the client side, migration is fully transparent as the reference to the fragment interface remains valid. Method calls to the agent may experience a short delay until the main fragment is fully transferred and until the new fragment at the destination host is in a consistent state. In FOAM we do not need any further mechanisms to keep contact with the mobile agent as in other architectures [LC96, OBJ97]. The reference to the mobile agent remains stable and a client can still communicate with the agent after migration via method calls.

It seems to be more complicated to realize a mobile agent with FOAM as there is the separation between the implementation and the interface, and the virtual cover of the distributed object. In fact, it is not. The realization of the fragment implementations is equivalent to the implementation of the relevant objects of the mobile agent. The fragment interface just depends on the methods of the mobile agent and therefore can be automatically generated from the type of the mobile agent (specified for example with a CORBA-IDL). The communication between stub fragments and main fragment uses RPC-based communication mechanisms of *AspectIX*.

3. Mobile Agents built with FOAM

The power of the FOAM approach is to use the fragmentation of the distributed object to define more fine grained migration entities of the mobile agent. Unlike conventional agents, mobile agents built with FOAM can consist of more than one fragment implementation. The agent functionality can be distributed over multiple fragment implementations which communicate with one another. During its lifetime the agent may contain different numbers of fragment implementations. Some fragment implementations may extend the agent at one time; at another time some fragment implementations may be deleted. This process in under full control of the agent. Each of the fragment implementations is considered to be individually mobile.

The question how to split up the agent functionality into fragment implementations has to be driven by mobility and distribution reasons. It does not make any sense to use multiple fragments for the sake of modularity alone how it would be done in the decomposition process in a monolithic object model. This approach miscarries, because the fragments would be not independent enough from each other which can result in higher bandwidth usage for inter-fragment communication. The developer of a fragmented agent has to decide which parts of the agent functionality have to be mobile for benefitting from efficient communication to local objects. At the same time the bandwidth usage for inter-fragment communication has to be minimized. The developer may restrict the size of the mobile fragments by placing large and not necessarily mobile code blocks into nonmobile fragments. Also, the developer may identify code blocks that are only needed under certain conditions. These code blocks may be placed in different fragment implementations that are only instantiated if the code has to run as part of the agent functionality.

Using FOAM, these possibilities in agent design offer a variety of advantages which are elaborated in the following sub-sections.

3.1 Scalability of Migration

One of the most important advantages of mobile agents is the improvement of bandwidth usage. With mobile agents, the computation moves to the data rather than the data to the computation as in RPC-based solutions. As the size of the mobile agent is the limiting factor for this advantage it is not always easy to decide which approach is the more efficient one. Especially if the agent is collecting data, this decision could change dynamically. FOAM offers two concepts for supporting this situation:

First, FOAM offers more scalability in the dimension of migration: in ordinary systems there is only a decision between RPC semantics and the usage of mobile agents. The first means no migration; the second means complete migration of the agent. In FOAM there are more than two possibilities—only the relevant parts of the agent may migrate in form of mobile fragments. The two ordinary decisions RPC and mobile agent are just special cases in FOAM: if no fragments are mobile the system is equivalent to the RPC-based concept; if the agent has only one mobile fragment including the complete agent functionality it is equivalent to the conventional mobile agent paradigm.

Second, FOAM offers dynamic adaption of the migration behaviour: the developer of the agent offers the finest granularity of migration entities with the size of the fragments. At runtime, the agent itself can decide, which fragments are mobile and therefore in which granularity the migration will occur. This decision can be changed dynamically depending on the input, the state and the environment of the agent.

In a scenario of Electronic Commerce, an agent may contain a fragment implementation that can investigate the interesting offers of a shop. This fragment does a pre-selection. Another fragment implementation may collect the interesting offers transmitted by the first fragment, and finally decide, which goods are to buy. The first fragment is mobile and contacts multiple shops; the second remains on the client site (see Fig. 3). As the first fragment does not need to contain the whole selection process it can be much smaller. Thus it is easier to migrate. On the other hand, the mobile fragment can communicate with the shop very efficiently.

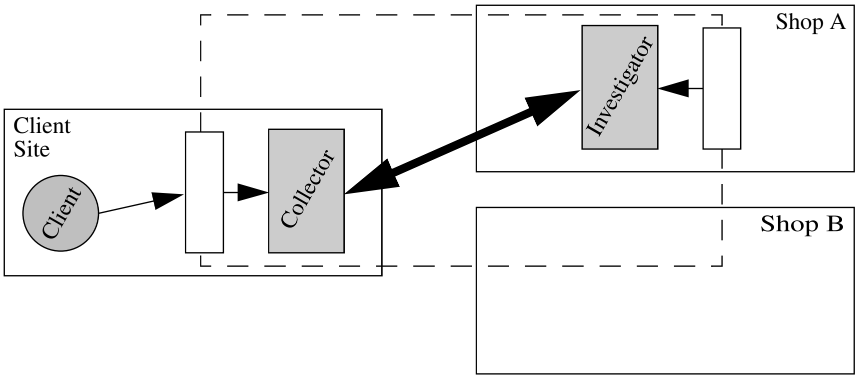

Fig. 3 Collector and Investigator Fragment of an Electronic Commerce Agent

If the available network offers higher bandwidth and there are only a few shops to process the agent may decide to instantiate a fragment at the shop site which combines the functionality of the formerly mentioned two fragments (see Fig. 4). In this case, the client of the agent will not notice any difference. We should note that regardless where the client resides it can always contact the agent using a local fragment that is either a stub fragment contacting the other fragments or a part of the agent functionality as it would be the case if the nonmobile and finally selecting fragment remains at the client site.

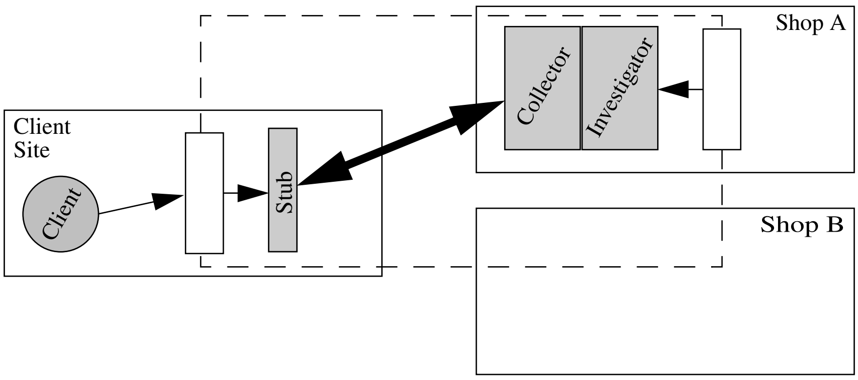

Fig. 4 Electronic Commerce Agent with a Single Fragment for Collector and Investigator

3.2 Replication

If a conventional agent moves from host to host collecting data and the current host is crashing, the mobile agent is lost and all its collected data is lost too. The developer of an agent may decide to replicate parts of the agent to cope with node failures. In FOAM, replication can be easily achieved. As the agent contains multiple fragments anyway and as these fragments belong to the same object, the developer can just instantiate multiple fragment implementations of the same type, which have to run some sort of consistency protocol to keep their states consistent.

In our aforementioned scenario of Electronic Commerce, the developer may decide to replicate the nonmobile fragment that collects the interesting shop offers transmitted by the mobile investigator fragment (Fig. 5). This investigator fragment can even use a multicast communica-

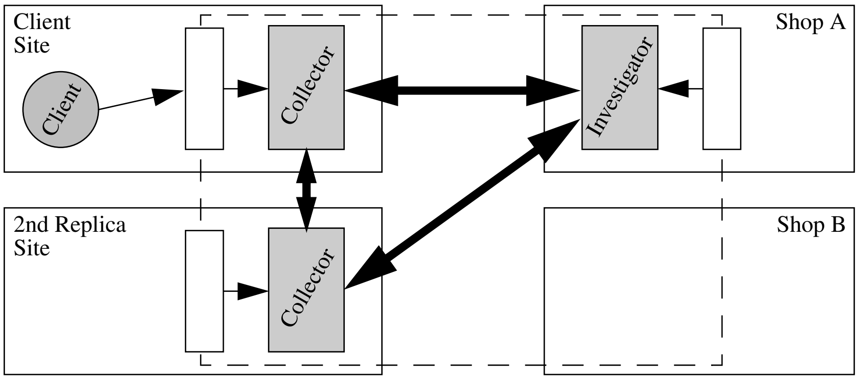

Fig. 5 Replicated Collector Fragment of the Electronic Commerce Agent

tion protocol to update the replicated collectors. In case of a node failure the collector fragments can be recovered as they are replicated.

3.3 Multiple Points of Presence

In many application domains the agent needs multiple feedback of different hosts in a undefined sequence to be able to fulfil its instructed task. In our Electronic Commerce example this task could be to search out a shop which offers the requested goods. In FOAM there could be multiple investigator fragments at the same time, each processing another shop. At an extreme, there could be one of these fragments for each shop. Thus, the agent is present at multiple locations at the same time (Fig. 6). Even more important is that the agent can decide how to process shops depending on certain conditions, e.g., available bandwidth for transmitting job offers, remaining time until a good has to be bought, costs of investigation at the shop sites, etc. An agent could decide to process one shop after the other at the rush hours, and all shops in parallel at night, depending on the request of the client.

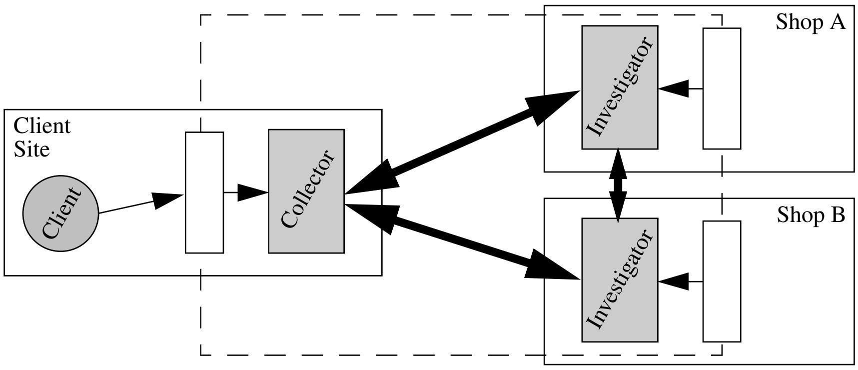

Fig. 6 Multiple Points of Presence of an Electronic Commerce Agent

Another example is a complex trading. An agent found more than one interesting offer of the same good and starts haggling for the price with the shop agents. For this process, it may use information collected by other fragments that investigate other shops concurrently.

In these cases the agent needs to have multiple points of presence. FOAM allows this by just having multiple fragments at different locations. The advantage of FOAM is that the fragmented agent is still considered to be one single object that can be transparently accessed by a client so that the client does not need to know anything about the internal structure of the agent. The agent can dynamically decide on multiple points of presence. Moreover, the agent can be truly distributed and needs no central component, which would be a single point of failure. In this case distributed algorithms have to be used (e.g. distributed consensus algorithms) to fulfil the requested semantics.

3.4 Phase-Based Fragments

As in FOAM fragment implementations can replace themselves by other implementations, the agent developer can design multiple fragment implementations representing different phases of the task of the agent. If a fragment has reached the end of a phase it replaces itself by the fragment implementation that realises the next phase. Thus, the fragments represent the structure of the workflow of the task to be done. The agent can benefit from such a structure as soon as only a fraction of the workflow is really to be done, e.g., if the workflow contains conditional clauses which result in different following phases or even the end of the process (see Fig. 7). Instead of moving the whole workflow code to another host, as made with conventional agents, a fragmented agent only needs to load the fragment implementations which correspond to the really needed phases. Thus FOAM-based agents benefit once again from a better migration to communication cost ratio.

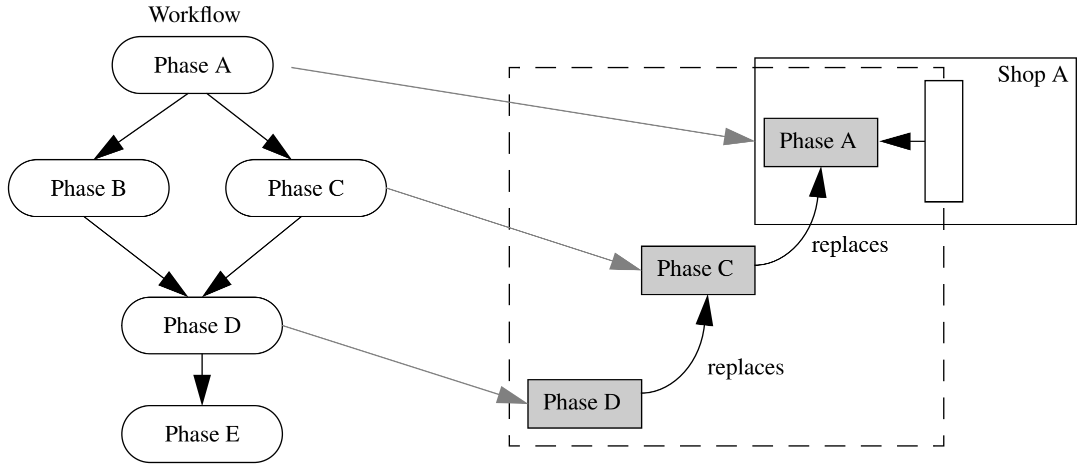

Fig. 7 Mobile Agent with Phase-Based Fragments

The phases of the investigator fragment of our Electronic Commerce example could be:

1. Evaluating if the shop offers the requesting goods
2. Haggling for the price
3. Buying the goods

A first fragment implementation realises phase 1 and checks if the requested goods are offered at the shop. If it does not find the requested goods, the process can be stopped and the fragment can delete itself. In this case just a minimum of code had to be transferred. Otherwise the fragment replaces itself with a haggling fragment implementation. This fragment implementation could cooperate with haggling fragments at other shops to find the optimal price. If it decided to buy the goods, the fragment implementation replaces itself by a buying fragment which performs the money transfer, e.g., by handing out credit card information. The replacements are fully transparent for the shop because the fragment interface of the agent remains always stable.

Another advantage of the phase-based approach is the possibility to hide critical data of the agent in specific phases. Each fragment just carries the needed information to realise the semantics of its representing phase. If this fact is considered in the development process of an agent, the expenditure to spy out its critical information can be increased. A malicious host only faces the current agent phase, the current fragment implementation respectively. The agent only changes the phase, if the result of the preceding phase is acceptable and there is enough trust to the shop for the next phase. The exposure of code and data is thus minimized to what is really needed (“need-to-know” principle). Especially if the agent travels to an unknown host, the danger of spied data is first reduced to the first fragment which should not keep critical data (maybe an itinerary). Of course this approach still cannot completely protect an agent of malicious servers. But to spy out the critical data of the agent the destination host would have to implement the whole bargain semantics and would have to know the expecting results of the agent in each phase.

4. Conclusion

The monolithic approach for implementing mobile agents is only feasible as long as the agents do not grow too big. The restriction of agent mobility for big agents also restricts one inherent advantage of mobile agents, e.g., low usage of network bandwidth. Agent systems do not scale with respect to the size of the agent.

With FOAM we offer a fragmented object model for mobile agents. A mobile agent is split into multiple fragments which still belong to the same distributed object—the agent. Each fragment can be mobile and thus benefit from efficient communication with objects at the same location. As a fragment does not need to include the complete agent functionality the migration entities can be much smaller than with the monolithic approach. A FOAM-based agent system can scale with the size of the agent and can still benefit from the advantages of mobile agents.

Additionally, a fragmented mobile agent can be present at multiple locations at the same time, and parts of the agent may be replicated. The agent does not necessarily have to hop from host to host but multiple fragments can be active at the same time at multiple locations. These fragments may interact to fulfil the task of the agent (e.g., they bargain with different vendors to find out the cheapest offer). The significant advantage of the FOAM solution is that all these fragments appear to have the same identity. For clients, they belong to one distributed object. There is no need for any external and central component controlling the fragments. The fragments can have an internal central component but they also can be organised in a completely decentralised manner.

We analysed how to split up an agent into several fragments. The decomposition of a fragmented agent has to be driven by distribution and mobility aspects which requires new programming techniques. Additionally, the workflow of the agent task may be used for decomposition. Fragments may replace themselves by other fragments implementing the next phase of the workflow. Especially if the workflow has conditional clauses between the phases, this optimises the bandwidth usage. Moreover if this fact is considered in the development process of an agent, the expenditure to spy out its critical information can be increased by exposing just the absolutely necessary parts of an agent to the destination host.

The *AspectIX* architecture, which FOAM is based on, is currently implemented in Java. We expect first experiences with fragmented agents on top of this prototype in a couple of months.

5. References

GSB+98 M. Geier, M. Steckermeier, U. Becker, F. J. Hauck, E. Meier, U. Rastofer: "Support for mobility and replication in the AspectIX architecture". In: *Object-Oriented Technology,* ECOOP'98 Workshop Reader, LNCS 1543, Springer, 1998, pp. 325-326.

HBG+98 F. J. Hauck, U. Becker, M. Geier, E. Meier, U. Rastofer, M. Steckermeier: *AspectIX: An aspect-oriented and CORBA-compliant ORB architecture.* Techn. Report TR-I4-98-08, IMMD IV, Univ. Erlangen-Nürnberg, Sep. 1998.

HCK95 C. G. Harrison, D. M. Chess, A. Kershenbaum: *Mobile Agents: Are they a good idea?* IBM Research Divsion, Watson Research Center, 1995.

LC96 D. Lange, D.T. Chang: *IBM Aglets Workbench: Programming Mobile Agents in Java*, *A White Paper Draft*. IBM Corporation, September 1996.

MBL+96 N. Minar, R. Burkhart, C. Langton, M. Askenazi: *The SWARM Simulation System: A Toolkit for building Multi-Agent Simulations*, Juni 1996.
http://www.santafe.edu/projects/swarm/

MGN+94 M. Makpangou, Y. Gourhant, J.-P. Le Narzul, M. Shapiro: "Fragmented Objects for distributed abstractions". In: T. L. Casavant, M. Singhal (eds.), *Readings in Distributed Computing Systems*, IEEE Comp. Society Press, 1994, pp. 170–186.

OBJ97 ObjectSpace: *ObjectSpace Voyager Core Package Technical Overview*. Technical Report, Juli 1997. Available at http://www.objectspace.com/

OMG98 Object Management Group: *The Common Object Request Broker Architecture*. Rev. 2.2, OMG Doc. formal/98-02-01, Feb. 1998.

Papa89 M. Papathomas: *Concurrency Issues in Object-Oriented Programming Languages*. In: D. Tsichritzis (ed.), Object Oriented Development. Geneva Univ., July 1989, pp. 207–245.

SHT97 M. van Steen, P. Homburg, A. S. Tanenbaum. *The architectural design of Globe: a wide-area distributed system.* Techn. Report IR-422, Vrije Universiteit Amsterdam, March 1997.

White96 J. White: *Mobile Agents Whitepaper*. MIT Press, Menlo Park, 1996.
http://www.genmagic.com/agents/Whitepaper/whitepaper.html