Fragmented Objects for the Implementation of Mobile Agents

Martin Geier, F. J. Hauck

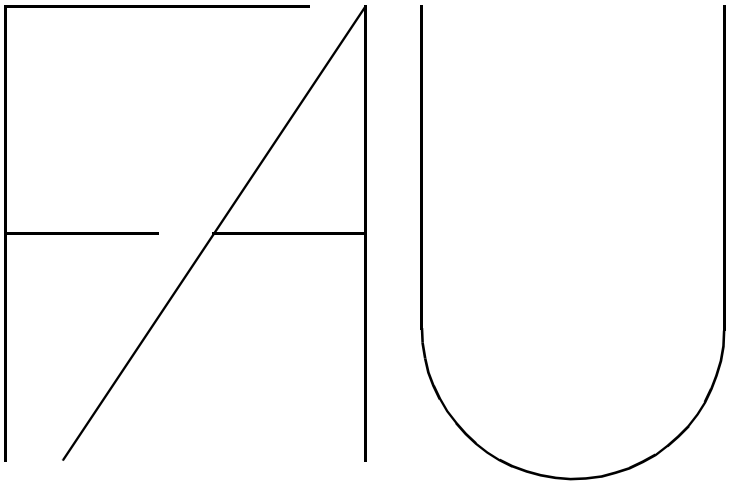
March 2000 TR-I4-00-05

Technical Report

Computer
Science Department

Operating Systems — IMMD IV

Friedrich-Alexander-University
Erlangen-Nürnberg, Germany



This technical report is an update of TR-I4-99-04.

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, 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. FOMAS, 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. The design of a fragmented agent is a crucial point, as the defragmentation has to be done for mobility and distribution reasons. We present the approach of phase-oriented agents (POMAS) as a design method for fragmented agents where agents are split into time-bounded sub-tasks. A mathematical model provides the break-even point at which the phase-oriented approach outplays the monolithic agent approach. These results are confirmed by a prototypical implementation.

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 programming paradigm for mobile agents is object-based programming. It defines the migration entity of a mobile agent as an object 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 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 *reduction of network load* (in comparison to the static client/server approach) 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.

Concerning performance evaluation, work has been done in comparing the mobile agent paradigm to the static client/server approach that is based on RPC-based programming paradigm. The results show that the rentability of using mobile agents is depending on the speed of the network and the amount of communication between the client and the server [IsHa99]. Taking this fact into consideration, research has been done to decide more dynamically whether to use RPC's or to migrate the agent to the server. One example for this work are the "strategically

mobile agents" presented in [ChKa97] where the decision wheter to migrate depends on an application characterization into different features like the volume of interaction or the migration size of the mobile agent.

However, mobile agents are still an area of research. Typical implementations of mobile agents are small and more or less pathological. To implement "real world" applications as they are needed for 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. The result of this approach is a static structure which is difficult to adapt to different requirements of distribution or migration.
- *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 sizes of the agents. However, for using many agents a special framework is needed which regulates the distribution and optimizes 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. We use a fragmented object model called FOMAS, 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 capabilities of the fragmented object model for agents. Agents can scale with respect to mobility, they can be replicated, and can have multiple points of presence. In Section 4 we discuss the approach of phase-oriented mobile agents as a design method for fragmented agents and present

a mathematical and prototypical evaluation of this approach. Finally, in Section 5 we draw our conclusions.

2. FOMAS and the AspectIX Architecture

Fragmented-**O**bjects for **M**obile **A**gent**s** (FOMAS) is based on the object model of the *AspectIX* architecture [HBG+98], which in turn is based on the object models introduced by *FOG* [MGN+94] and *Globe* [SHT99]. 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 and invokes method calls at this representative of the distributed object. A fragment 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 locations at the same time.

Fragments can themselves be considered to be objects on a different level of abstraction. The difference between a fragmented distributed object and a group 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 group have individual identities and are not related to each other for the underlying middleware system or any client using them. This can also manifest itself in the design of a fragmented object as presented in Section 4.

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, 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. As clients are only bound to the fragment interface, the fragment implementation can be transparently exchanged without loosing the validity of the client reference.

2.2 Conventional Agents and FOMAS

FOMAS uses the *AspectIX* platform for mobile agent systems. In this sub-section we will show how conventional mobile agents can be implemented with FOMAS. We use the term *conventional* for mobile agents following the monolithic approach. In fact, monolithic mobile agents

can be easily implemented with FOMAS. 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]).

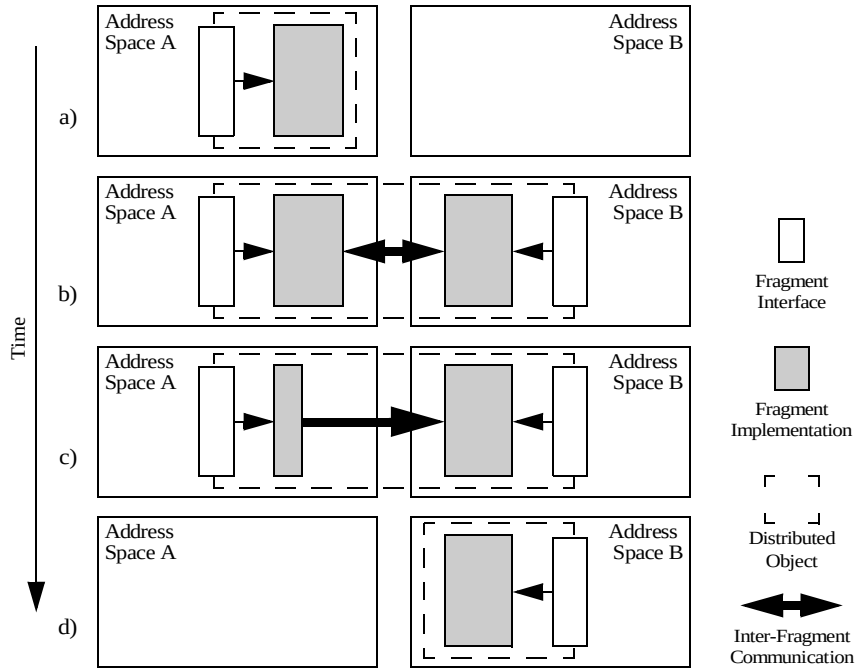


Fig. 1 Phases of the Migration of a Monolithic Agent in FOMAS

The different phases of a migrating monolithic agent in FOMAS are displayed in Figure 1. First, the agent consist of only one fragment implementing the functionality of the agent (Fig. 1a). Second, the agent extends itself by creating a new fragment at the destination site (Fig. 1b). 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. 1c). Finally, if no further access to the mobile agent is needed from the original site, the fragment on this site can be deleted (Fig. 1d).

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 FOMAS we do not need any further mechanisms to keep contact with the mobile agent as it is necessary in other architectures [LC96, OBJ97]. The reference to the mobile agent remains valid and a client can still communicate with the agent after migration.

It seems to be more complicated to realize a mobile agent with FOMAS 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 a conventional mobile agent. The additional fragment interface just depends on the public methods of the mobile agent and therefore can be automatically generated from the IDL-description of the mobile agent.

3. Mobile Agents built with FOMAS

The power of the FOMAS 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 FOMAS 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 as 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's functionality. How all this conditions can be taken into account in a design for a fragmented mobile agent is presented in Section 4. In general, using FOMAS offers 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 reduction of network load. 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 is more efficient. Especially if the agent is collecting data, this decision could change dynamically. FOMAS offers two concepts for supporting this situation:

First, FOMAS 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 FOMAS there are more than two possibilities—only the relevant parts of the agent may migrate in form of mobile fragments. The two ordinary decisions are just special cases in FOMAS: 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, FOMAS 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. 2). 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.

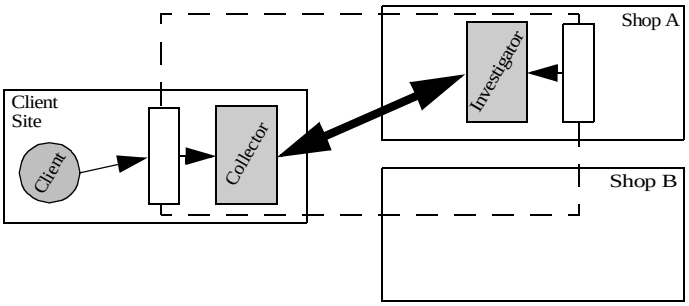


Fig. 2 Collector and Investigator Fragment of an Electronic Commerce Agent

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. 3). In this case, the client of the agent will not notice any difference. 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.

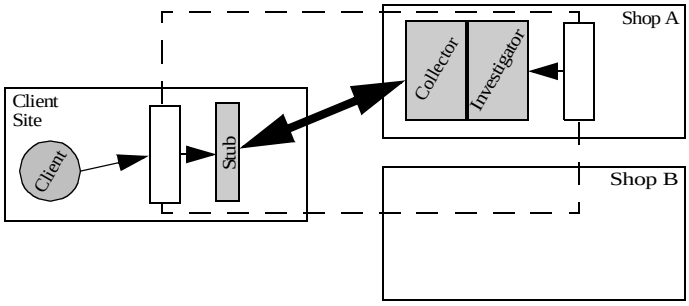


Fig. 3 Electronic Commerce Agent with a Single Fragment for Collector and Investigator

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 FOMAS, 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, the developer may decide to replicate the nonmobile fragment that collects the interesting shop offers transmitted by the mobile investigator fragment (Fig. 4). This investigator fragment may use a multicast communication protocol to update the replicated collectors. In case of a node failure the collector fragments can be recovered as they are replicated.

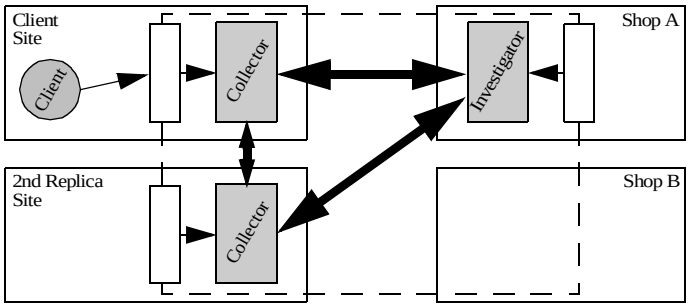


Fig. 4 Replicated Collector Fragment of the Electronic Commerce Agent

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 FOMAS 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. 5). 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.

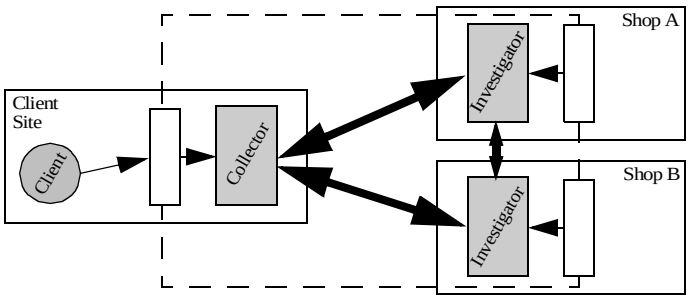


Fig. 5 Multiple Points of Presence of an Electronic Commerce Agent

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. FOMAS allows this by just having multiple fragments at different locations. The advantage of FOMAS 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.

4. Phase-Oriented Mobile Agents

One major challenge of the fragmented agent approach is finding an adequate design for the realisation of FOMAS agents. Especially the decomposition technique which has to be driven by

mobility and distribution reasons instead of modularity reasons is important for the prosperity of FOMAS. In this Section we present the approach of socalled **P**hase-oriented **M**obile **A**gent**s** (POMAS) which considers the structure of a fragmented agent:

The idea of the POMAS design is to decompose the mobile agent into time bounded sub-tasks which represent different phases of the task of the agent. Each sub-task is implemented in an own fragment. If a certain phase has reached the end, the correlating fragment implementation replaces itself by another implementation representing the next phase of the mobile agent. This can be done transparently as mentioned in Section 2.1.

The connection between the phases is separated of the fragment implementations and described in a workflow description. The needed workflow engine itself is either implemented as an own fragment of the FOMAS agent or the workflow engine just exists “virtually”: then the corresponding decisions to be done are weaved into the different fragments representing the phases, e.g., with mechanisms from aspect-oriented languages [KLM+97].

The fragments of a POMAS agent 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. 6). 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 POMAS agents benefit from a better migration to communication cost ratio.

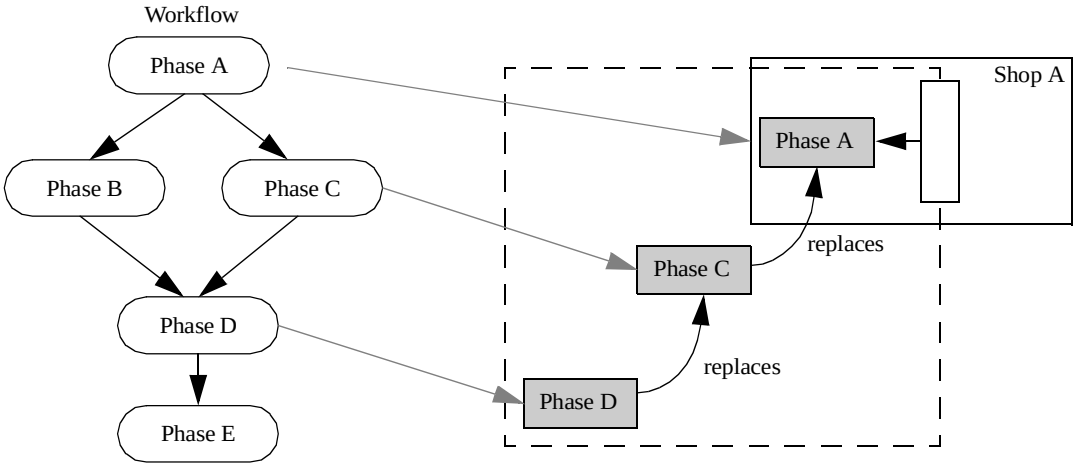


Fig. 6 Mobile Agent with Phase-Oriented Fragments

The choice of the next phase can be done dynamically at runtime. Each phase returns a result at the end which is passed to the workflow engine. The engine determines the next phase dependent on the result of the currently ended phase and optionally on additional parameters, e.g., the current latency of the network link between the fragment repository and the server. This additional parameters are determined by the current phase and transferred to the workflow engine while passing the result of the phase.

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 realizes phase 1 and checks whether 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 the same.

We did two different approaches to evaluate the concept of fragmented objects for mobile agents: first we developed a mathematical modell of the POMAS approach and second we proved the mathematical results by measurements with a prototype.

4.1 Mathematical Evaluation

The main focus of the mathematical modell is to find the break-even point where the fragmented agent model outplays the monolithic approach of a mobile agent. For this we use the POMAS-approach and assume the following conditions:

- A single-hop scenario (which implies one source host and one single destination host).
- No inter-fragment communication (we concentrate our comparison on the pure migration costs).
- No code-metrics considerations (we abstract of different costs for marshalling of different constructions of the agent).

	Meaning
B	Bandwidth [kByte/s]
L	Latency [s]
G_m	Size of the monolithic agent [kByte]
M	Time for marshalling [s]

Table 1: Parameters for the monolithic agent

	Meaning
B_i	Available bandwidth for fragment i [kBytes/s]
L_i	Latency for fragment i [s]
G_i	Size of fragment i [kByte]
M_i	Time for marshalling for fragment i [s]
C_p	Number of fragments needed to fulfill the job
C_f	Total number of fragments of the POMAS agent

Table 2: Parameters for the POMAS agent

The monolithic agent. We use the parameters and notations of table 1 for the monolithic case. The following formula expresses the relation of needed time for the migration and the parameters of table 1:

$$T = L + \frac{G_m}{B} + M$$

The POMAS agent. Table 2 shows the parameters which are used for the POMAS approach. We assume, that the connection between source and destination host is not setup again for every single fragment. Then the following formula expresses the needed time to successfully finish the requested job of the agent:

$$T = L + \sum_{i=1}^{C_p} \frac{G_i}{B_i} + M_i$$

We assume, that there exists a partition of the agent into fragments of the same size. For the needed marshalling time of each fragment we do a worst-case estimation by using the marshalling time of the whole agent. Then we get the following formula:

$$T = L + C_P \times \left(\frac{G_i}{B} + M \right)$$

The visualisation of this formula is a function band over C_P and G_i which expresses how much time is used for the transfer of C_P phases depending of the size G_i of the fragments. Fig. 7 charts the function band for a phase-oriented agent with 10 fragments on a network with 10 MBit/s bandwidth. As latency we used 1 second and as marshalling time we used 400 ms.

Comparison of monolithic and phase-oriented agents. We can now overlap the curve of the monolithic agent with the same parameters over the function band of figure 7 and get the expected break-even point as the cut of the function band with the curve representing the monolithic approach:

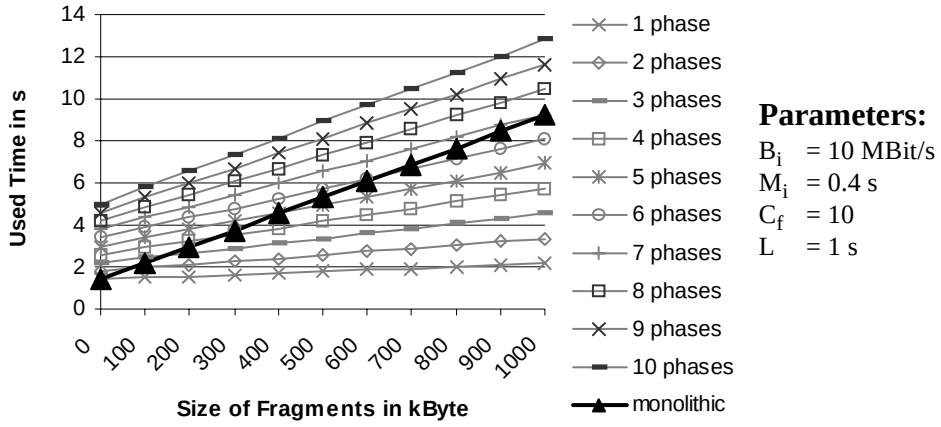


Fig. 7 Mathematical evaluation of POMAS agents

By equating the two formulas we can express the break-even point by the following equation:

$$C_P(G_m) = \frac{C_f G_m + C_f MB}{G_m + C_f MB}$$

The result is as expected: the bigger the mobile agent is, the more phases can be transferred without reaching the time used for the monolithic pendant. In Figure 7 you can see, that for a fragment size of 300 kBytes the phase based agent is more efficient until up to 3 phases are transfered whereas at a fragment size of 1 MByte already 6 phases could be transfered without reaching the break-even point. On the other hand, for too small agents (proportional to the available bandwidth) the monolithic approach is in advantage as the overhead of fragmentation does not compensate the migration costs of the single fragments.

4.2 Prototype and Measurements

To evaluate that the mathematical model correlates with reality, we implemented a protoype for POMAS agents. We use Voyager [OBJ97] as platform for our prototype because the AspectIX architecture is still under development and Voyager already offers the needed capabilities for migration. The prototype offers the definition of fragments of a POMAS agent and the description of a workflow which is used to connect the fragments as described in Section 4. The workflow engine is implemented as an own nonmobile object and every fragment communicates with the engine to initialize the phase exchange.

For the correlating measurements of the mathematical model every fragment has the same size but is implemented as an own class, so that the code is not cached in the classloader of the des-

tination host. The size of the fragment is varied by initialising a simple byte array in every fragment with the correct size. We therefore do not consider any code metrics as it is done in the mathematical modell as well. Every fragment just migrates to the destination host and then immediately initialises the next phase exchange. We therefore reduce the evaluation to the bandwidth usage of the POMA agent. The monolithic agent is realized by just one fragment with a simple byte array of size $G_i * C_f$ (count of POMA fragments * size of POMA fragments).

We did measurements on different networks (56k Modem, ISDN, 10MBit, 100MBit) to evaluate the approach and to see the correlation between offered bandwidth and the break-even point. Every single measure point is averaged to lessen the dispersion by measuring it 5 times in a row.

The result is presented in figure 8. The graphs show the same tendency as the figure 7 of the comparison between the monolithic approach and the POMAS approach in the mathematical model and therefore confirm the result. There are still gaffes in the measurements which we interpret as following:

- The bandwidth of the available network on the 100MBit and the 10MBit link had to be shared with other applications. Because the measurements to lessen the dispersion were done successively, other aplications were able to affect the measurement adversely.
- Voyager and the underlying Java use dynamic buffers for memory management. On certain boundaries, the Java Virtual Machine has to allocate new memory which causes a delay in the execution of the evaluation application.

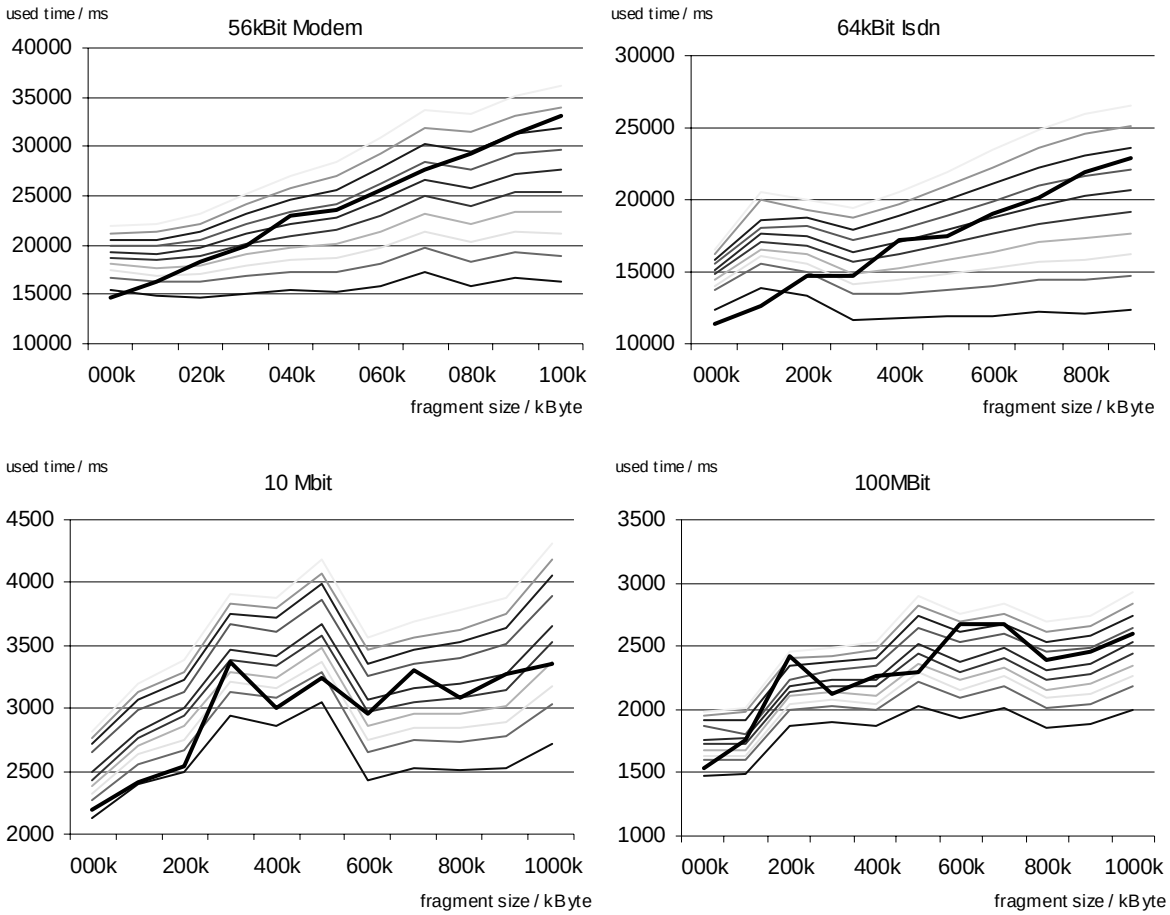


Fig. 8 Prototypical comparison between monolithic agents and fragmented agents on different networks

5. 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 FOMAS 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 FOMAS-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 fulfill the task of the agent (e.g., they bargain with different vendors to find out the cheapest offer). The significant advantage of the FOMAS 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. We offer a phase-oriented design approach to realize the decomposition which is driven by time bounded sub-tasks of the agent. A workflow defines the connection of the different phases, which are implemented as fragments of the agent. A fragment may replace itself by another fragment implementing the next phase of a workflow. Especially if the workflow has conditional clauses between the phases, this optimizes the bandwidth usage.

We use a mathematical model to find the break-even point at which the phase-oriented approach outplays the monolithic agent approach. The model shows that the bigger the mobile agent is, the more phases can be transfered without reaching the time used for the monolithic pendant. The phase-oriented approach therefore has its magnitude if the agents are big at the ratio of the offered bandwidth. We implemented a protoype to evaluate the mathematical model. The results of the measurements present the same tendency as the results of the model and therefore confirm them.

6. References

ChKa97 T.H. Chia, S. Kannapan: *Strategically Mobile Agents.* In: LNCS 1219, Proc. of Mobile Agents, 1. Int. Workshop, Berlin, April 1997.

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.

IsHa99 L. Ismail, D. Hagimont: *A performance evaluation of the mobile agent paradigm.* OOPSLA'99, ACM Sigplan, Vol. 34, No. 10, October 1999

KLM+97 G. Kiczales, L. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, J. Irwin: *Aspect-oriented programming.* Techn. Report SPL97-008 P9710042, Xerox Palo Alto Research Center, Feb. 1997.

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.

SHT99 M. van Steen, P. Homburg, A. S. Tanenbaum. *Globe - a wide-area distributed system*. IEEE Concurrency, Jan.-March 1999; pp. 70-78.

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