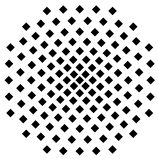

Universität Stuttgart
Fakultät Informatik

# A Comparison of Mechanisms for Locating Mobile Agents

**Authors:**
Dipl.-Inform. J. Baumann

Institut für Parallele und Verteilte
Höchstleistungsrechner (IPVR)
Fakultät Informatik
Universität Stuttgart
Breitwiesenstr. 20 - 22
D-70565 Stuttgart

**A Comparison of Mechanisms for Locating Mobile Agents**

*J. Baumann*

Bericht 1999/11
August 1999

# A Comparison of Mechanisms for Locating Mobile Agents

*Joachim Baumann*

Email: Joachim.Baumann@informatik.uni-stuttgart.de

Institute of Parallel and Distributed High-Performance Systems (IPVR)

University of Stuttgart - Germany

Breitwiesenstraße 20-22

D-70565 Stuttgart

## *Abstract*

In this paper we present different possible approaches for locating mobile agents and introduce a classification for them. We will use this classification to categorize mechanisms proposed in standards and implemented in mobile agent systems. Then we assess the different mechanisms regarding their fault tolerance, their message complexity and the migration delay they induce. We conclude by combining the different assessments to allow a comparison of all mechanisms.

**Keywords**

Mobile Agents, Distributed Systems Architecture, Distributed Algorithms, Communication Protocols.

# 1 Introduction

Due to its notable properties the mobile agent paradigm has received a rapidly growing attention over the last few years. The research community involved in the area of mobile agents is steadily growing, and more and more systems are developed in both academia and industry. Moreover, standardization efforts for mobile agent facilities and architectures are already in progress.

These architectures have to provide functionality for agent migration, communication of agents with other agents and with the underlying system, and for agent control (i.e. to start agents, to stop agents, to find agents etc.). In this paper we will concentrate on mechanisms that allow to locate a mobile agent. We will discuss the advantages and disadvan-

tages of the different existing approaches, provide a classification and examine the mechanisms' fault tolerance and message complexity.

The paper is structured as follows: we start with discussing the need for mechanisms to locate agents in Section 2, present a minimal agent, system and fault model in Section 3, and examine and classify the different possible approaches in Section 4. In Section 5 we will discuss existing mechanisms, some of which are combinations of different approaches discussed in Section 4, and categorize them. We will analyse the mechanisms' fault tolerance, message complexity, and their effects on the agent autonomy in Section 6. We conclude with an assessment of the qualities of the different mechanisms in Section 7.

## 2 Why Mechanisms to Locate Mobile Agents?

Regardless of the application employing mobile agents, the ability to get information about the computation in progress (i.e. the status information from the agents) is a crucial prerequisite to decide on further behaviour (e.g. to stop the agent or to start additional ones). To request such information from an agent, the agent's location has to be known, which implies that the agent has to be located by the system. In theory, global communication mechanisms such as a distributed tuple space could be an alternative (for an introduction to tuple spaces see CARRIERO AND GELERNTER (1989)) or event channels as defined by the OMG could be used (see BAUMANN ET AL. (1997) for usage examples). But the communication costs for maintaining distributed tuple spaces (i.e. for maintaining the global consistency of the tuple space) are extremely high, and implementations of OMG event channels normally provide only best-effort semantics. One exception is an event mechanism for mobile agents developed by BECK (1997) providing a reliable causal-order event channel, but the communication costs are extremely high (in fact, a reliable multicast tree is created that is extended and pruned with every movement of a member). It is useful if employed for agent coordination (see e.g. BAUMANN AND RADOUNIKLIS (1997)), but too expensive to be used for control mechanisms (see PAULUS (1998) for details). Thus other mechanisms are needed that provide the ability to locate an agent with lower costs.

# 3 Agent, System and Fault Model

## 3.1 The Agent Model

The minimal agent model used for this paper contains only properties that are either common to existing mobile agent systems, or can be implemented without problems on top of existing systems. The agent model is based on the concepts of places and agents (see Figure 1).

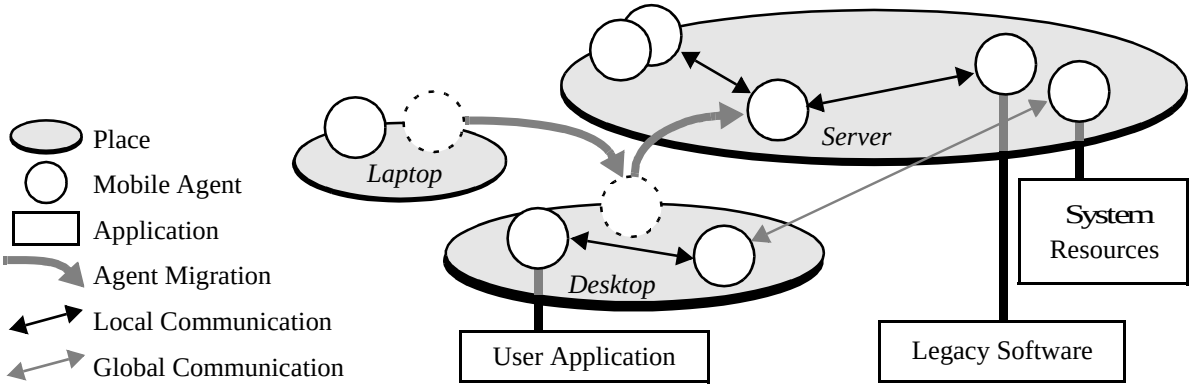


*Figure 1: The agent model*

Places provide the environment for executing mobile agents. Additionally they may provide abstractions of services of the underlying system. A place is entirely located on a single node of the underlying network. An agent system consists of a number of these places. Mobile agents are active entities, which may move from place to place to meet other agents and to access the places' services. An agent can be identified by a globally unique agent identifier, which is generated at the agent's creation time and is not changed throughout its life. The place on which the agent has been created can be derived from the identifier. Communication between agents may be local or global.

## 3.2 System and Fault Model

Throughout this paper we will use the terminology defined in JALOTE (1994). We assume a large, distributed system (e.g. the Internet), on top of which the mobile agent systems (of principally the same size) are executed. This distributed system consists of autonomous nodes that are connected to each other by a communication network. Each node consists

of a processor, private volatile and private stable storage. The nodes are loosely coupled, do not have shared memory and communicate via message passing. The communication network is assumed to be point-to-point.

In this large, distributed system we can distinguish node and network faults. We assume that nodes suffer from crash failures only. The failure causes the node to halt and to lose its internal volatile state. The stable storage survives failures. We assume a communication protocol is used that supports full connectivity between the nodes, and the delivery of messages in order, correct, and exactly-once as long as no network fault occurs. Furthermore, we assume the communication protocol to be fail-aware. Protocols providing this type of reliable datagram service are common, i.e. this assumption is close to reality. Consequently the following can be assumed: the communication network is fully connected and it provides reliable communication channels as long as no network fault occurs. Communication networks can suffer from crash failures that may cause the network to be partitioned. In the case of a network partition the communication channel between sender and receiver in different partitions fails, but continues to work between participants in the same partition. We assume that no failure is permanent, i.e. every encountered network or node failure is transient. Node and network failures are detectable, but not distinguishable.

# 4 Categorizing Mechanisms for Locating Agents

Different approaches for locating agents can be identified, depending on the assumptions about migrational behaviour, on assumptions about the size of the agent system, and on assumptions about the communication costs. In this section we present the different possible approaches that could be used for locating agents, and categorize them.

## 4.1 Possible Approaches

### 4.1.1 Preordained Migration Paths

If we assume that an agent migrates only along a preordained migration path, then by probing the different places along its path, we sooner or later have to find it. This can be done either sequentially (probing one place at a time) or in parallel (probing more than one place at a time). Additional information gathered from the places, e.g. whether the agent already visited this place, or the time an agent remains on a place on average, can help to identify the next place(s) to probe.

Simple examples for algorithms to identify the next place to probe are binary search (sequential case) and ternary search (parallel case).

### 4.1.2 Autonomous Migration

If we assume that the mobile agent has full autonomy, i.e. that it has no preordained migration path to follow, then probing is not practical. The following approaches are usable instead if the migration path of the agent is not known in advance.

#### 4.1.2.1 Logging

These approaches store information that ultimately leads to the place on which the agent resides. This can be done directly, in a database or indirectly, with path proxies.

**Global Database.** If every agent, upon every migration, informs a global database about its new location, then the problem of finding an agent is reduced to requesting its location from the database. This database might simply be a file on one node in the network. The disadvantage is that the migration of every agent now depends on the database's availability (this includes the availability of the node on which the database resides and the communication channel between this node and the agent place).

**Local Database.** This approach has the same characteristics as the approach employing a global database with the additional advantage, that distributing data over different local databases scales better (i.e. will not be one central bottleneck even with a high number of migrating agents). The disadvantage is that now the correct local database has to be identified. One simple method to do this is to define a relationship between (unique) agent name and database node.

**Untimed Path Proxies.** If an agent stores on every place it visits, when leaving, a pointer to the target place of its migration, then a path of *proxies* is created. Following this path of proxies eventually leads to the agent. Since the information is stored along the path, no additional communication is necessary for maintaining the path. A similar technique has been used in Emerald (object proxies, discussed in JUL ET AL. (1988)).

**Timed Path Proxies.** The problem of the untimed path proxies is the unbounded length of the path. This leads, depending on the length of the path, to a low availability and to a high message volume along the path if an agent has to

be located (see discussion in Section 6.1.1). This problem can be solved by assigning a *time-to-live* or *ttl* to the path proxies. After the ttl the path is shortened. This leads to slightly higher costs for maintaining the path (one additional message for every ttl interval). But this solution provides a better availability and lower message volume when an agent has to be located than the untimed path proxies (we elaborate on this in Section 6.1.2).

#### 4.1.2.2 No Logging

Here no information about the agent's location is saved, i.e. the mechanisms have to use a brute force approach to find the agent. The message volume created by these mechanisms is comparatively high (we examine the message complexity of the different mechanisms in Section 6.2).

**Sequential Brute Force.** Here the search is done sequentially, one place at a time. On average the number of messages sent equals the number of places, since on average the agent is found after half the places have been searched (each inquiry needs a request and a reply). The assumption here is that the agent does not migrate while the search is in progress. If that happens, then in the worst case the agent is not found (if the agent migrates from a place not yet examined to an already searched place).

**Parallel Brute Force.** If the search is done in parallel instead, i.e. a broadcast with the request is sent to every place in the system, then the answer is obtained faster, but the message volume equals twice the number of places. The number of messages can be reduced (down to the number of places plus one message for the reply), if only a positive reply is sent. Again the assumption is that the agent does not move while the search is in progress.

#### 4.1.2.3 Non Deterministic Approaches

These approaches do not allow to locate the agent always. Instead, the necessary information is only provided occasionally.

**Advertising.** Here the agent advertises its location (sends its place to a database) whenever it (more precisely the agent's programmer) deems it necessary. Whenever an agent migrates and does not advertise, it cannot be found with this approach.

**Energy.** In its life an agent consumes resources of the places on which it resides (e.g. cpu time) and uses services provided on a place (e.g. a directory service). This approach assumes that access to each service and use of every resource

are associated with a cost, which we call energy. An agent gets an amount of energy and uses this energy to access the services and to use the resources provided. As soon as its energy is depleted, the agent has to contact e.g. a database to request more energy, and incidentally the database is updated. Each time this happens the agent can be found until its next migration.

## 4.2 Classification

A graphical representation of these different approaches leads to the classification in Figure 2. Mechanisms for locating agents assume either preordained paths or autonomy of the agent migration. In the first case, the path can be probed either sequentially or in parallel. In the second case (agent autonomy) three different classes can be distinguished, the non deterministic mechanisms, mechanisms using logging to identify the location of the agent, and mechanisms using brute force to find the agent.

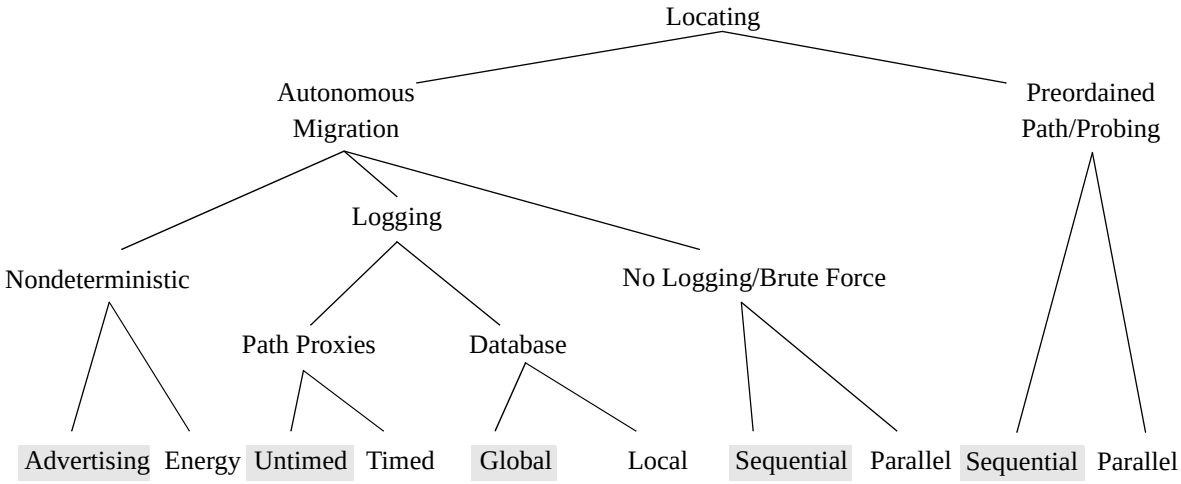


*Figure 2: Locating agents: a classification*

# 5 Actual Mechanisms

We will now examine mechanisms that have been proposed or have actually been implemented. We will start with the mechanisms for finding agents that have been proposed in the MASIF standard, the OMG Mobile Agent System Interoperability Facility (see MILOJICIC ET AL. (1998) for a description of the standard). Then we will examine the mechanisms implemented in the Aglets Workbench (an introduction can be found in LANGE AND OSHIMA (1998)), some of

which are exactly the mechanisms proposed in the MASIF standard (which is quite understandable, considering that two authors of the MASIF standard were part of the Aglets team). We will continue by investigating the mechanisms that have been implemented for the Mole system (see BAUMANN ET AL. (1998A) for an introduction to the concepts). Last we will discuss a mechanism that has been proposed by CHEN AND LENG (1997) for mobile agents with a preordained path. We will give a short introduction to the standard respectively the system, where appropriate.

## 5.1 MASIF

MASIF tries to define a minimal interoperable interface for mobile agent systems. Agents in the MASIF standard can migrate between places, and they have a unique identifier. Communication between agents is not addressed in the standard. Thus the MASIF agent model conforms to the model given in Section 3.1.

According to MILOJICIC ET AL. (1998) MASIF defines an interface named MAFFinder, which offers the following mechanisms to locate agents (in braces the category according to the classification is given):

- *Brute force search (autonomous | no logging | parallel).* This technique first identifies every place in a region (a set of places owned by the same person or organization), then checks each place to find the agent.
- *Logging (autonomous | logging | path proxies | untimed).* Whenever an agent leaves a place, it leaves a log entry on it (called mark) pointing to the target place of the migration. The log entries are garbage-collected after the agent dies (it is not discussed how this can be done).
- *Agent registration (autonomous | logging | database | global).* Every agent registers its current location in a global database. This database can be queried to find the agent.
- *Agent advertisement (autonomous | nondeterministic | advertising).* The agent advertises its location whenever it deems it necessary. To find an agent for which the advertised information is out-of-date, brute force search is used.

## 5.2 Aglets

The Aglets Workbench was created at the IBM Tokyo Research Laboratory (see IBM (1999)). An aglet is a combination of the applet model and the agent model, in principle adding mobility to applets. While this approach allows an applet programmer to quickly grasp the functionality of aglets, it constrains aglets to a mainly event-based model.

Aglets have immutable, globally unique names and can communicate with other aglets via messages. Communication can be local and global, synchronous or asynchronous. Aglets can migrate between *contexts* (the framework's term for a place) to access services provided by the context or to communicate with other agents. Thus the agent model used in the Aglets workbench also conforms to our agent model.

ARIDOR AND OSHIMA (1998) discuss requirements for a mobile agent infrastructure and present their implementation in the Aglets Workbench. The schemes for locating agents discussed here are similar to those given in the MASIF. The main differences are:

- the brute force search can be done either sequentially *(autonomous | no logging | sequential)* or in parallel *(autonomous | no logging | parallel)*.
- paths created by the logging scheme can be cut short.
- registration can be used with distributed data bases *(autonomous | logging | database | local)*.
- The advertisement scheme is not part of the Aglets Workbench.

## 5.3 Mole

Mole is one of the first Java based mobile agent systems and was developed at the University of Stuttgart. Agents in Mole have immutable, globally unique names. They can migrate between different places to access services or to communicate locally with other agents. Mole supports a large variety of communication mechanisms (see BAUMANN ET AL. (1997) for a detailed description). The agent model used in Mole concurs with our agent model.

Mole contains three different mechanisms for locating mobile agents, which have been presented by BAUMANN (1997) and by BAUMANN AND ROTHERMEL (1998B). The different mechanisms are:

- *Energy (autonomous | nondeterministic | energy)*. The implementation in Mole is a realization of an orphan detection scheme. Finding an agent works as described above, but additionally an agent is terminated as soon as it has no energy left (e.g. because no additional energy has been granted).
- *Paths (autonomous | logging | path proxies| untimed)*. The path concept is an implementation of untimed path proxies as described above.

- *Shadows (autonomous | logging | database | local), (autonomous | logging | path proxies | timed)*. The shadow concept is a combination of the distributed database approach (with the shadow acting as the local database) with timed path proxies to minimize the communication cost while maintaining a high availability. Additionally to providing the functionality to locate agents it allows orphan detection and termination. We will examine only the basic protocol.

  Each application creates one or more *shadows* (holding the information about the locations of the depending agents) on a place. In regular intervals (called *time to live* or *ttl*) the place on which the agent resides updates the information in the associated shadow (see Figure 3).

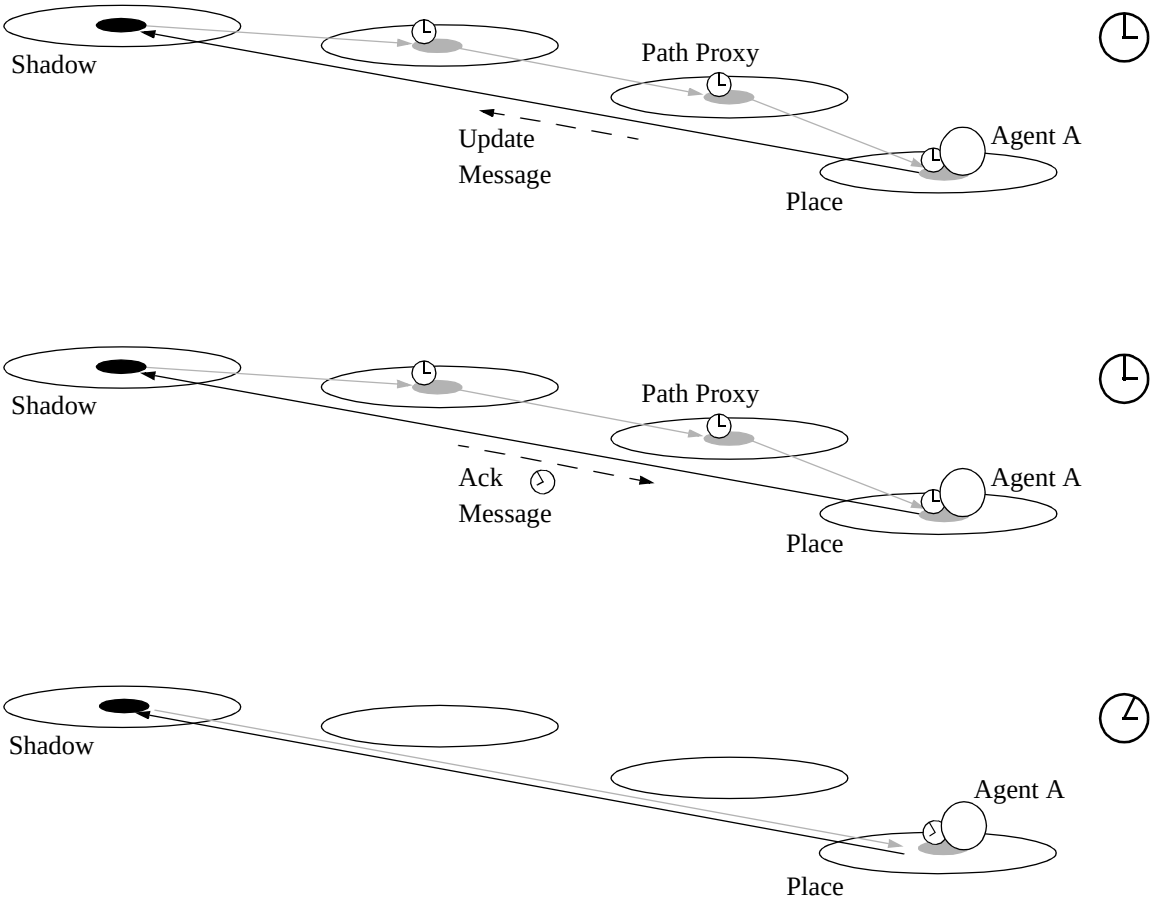


*Figure 3: Shadows: regular update of proxy paths*

Now the old path proxies are no longer needed. By keeping the *ttl* with every path proxy, this can be determined locally, i.e. without additional communication. Even if the path is broken (e.g. because a node crashed) the worst-case time until the agent is reachable again is the *ttl*, because after that time it will have contacted the shadow again. By changing the value for the *ttl* the mechanism can be adapted to changing costs and fault tolerance requirements.

## 5.4 Locating Agents with the Help of a Probability Function

CHEN AND LENG (1997) have proposed a mechanism that uses the knowledge of an agent's movements to guess its location with the help of a probability function *(probing | sequential)*. It assumes a predefined path from which the agent cannot deviate, and tries to compute the current place of the agent using the time interval since the agent left (assuming a binomial distribution of the execution time on each place).

## 5.5 Categorizing the Mechanisms

If we add the results of our discussion of existing mechanisms into the classification, we yield Figure 4. It can be seen that every system and standard provides mechanisms of widely different types.

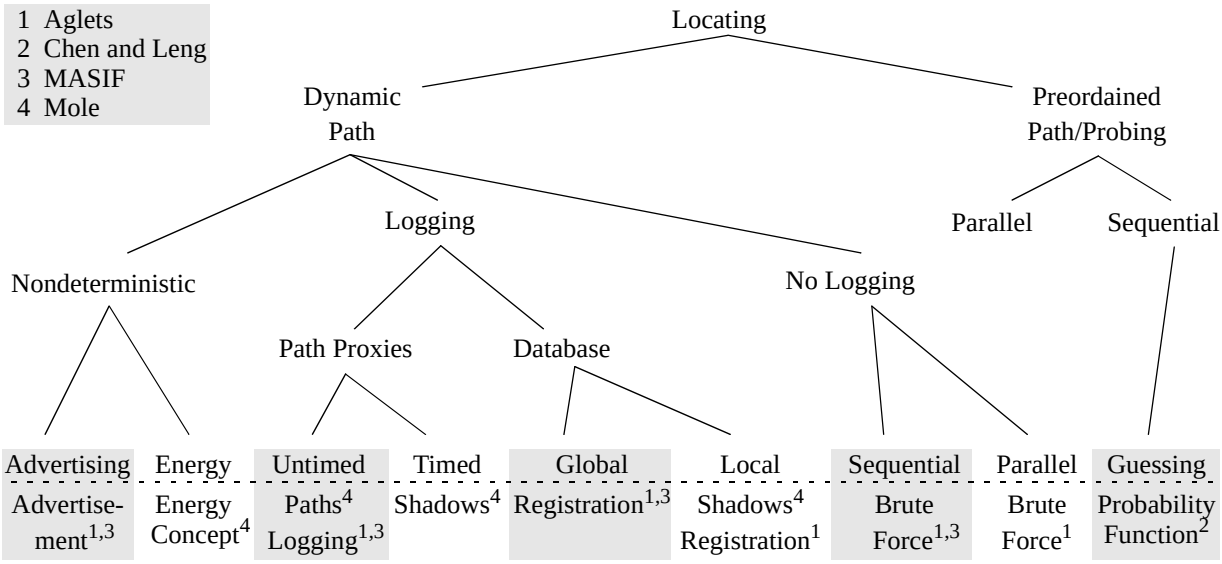


*Figure 4: Locating agents: the mechanisms existing in different systems*

# 6 Assessing the Mechanisms

To assess the discussed mechanisms we will determine their fault tolerance, their message complexity and the constraints they put on the autonomy of the mobile agents (i.e. whether the migration is delayed).

## 6.1 Fault Tolerance

To assess the fault tolerance of a mechanism the availability has to be computed. We will again use the terminology and methodology given by JALOTE (1994). For most of the mechanisms this is quite simple; these are advertisement, energy, global and local databases, sequential and parallel brute force, and the probing mechanism. In every one of these types, two nodes (the agent node and the node holding the database or energy source) and two communication channels are involved (one between inquirer and database and one between inquirer and agent place). Let us denote the failure rate for nodes with $\lambda_n$ and the failure rate for communication channels $\lambda_c$. This leads to the following equation for the availability :

$$\alpha = \frac{MTTF}{MTTF + MTTR} = \frac{1}{1 + MTTR(2\lambda_n + 2\lambda_c)} \qquad \textit{(Equation 1)}$$

If we assume that the mean time to repair (MTTR) is 0,5 hours and that the failure rate for nodes $\lambda_n$ and the failure rate for communication channels $\lambda_c$ is of the same value $\lambda$, then the availability $\alpha_\lambda$ is:

$$\alpha_\lambda = \frac{1}{1 + 2\lambda} \qquad \textit{(Equation 2)}$$

With an MTTF for the single components of 50 hours (failure rate $\lambda$ of 0.02/hour), the availability $\alpha_\lambda$ is 0.96.

### 6.1.1 Untimed Path Proxies

For untimed path proxies the fault sensitivity added by the paths as presented above manifests itself in the dependency on the availability of all nodes containing a proxy, i.e. on those nodes that are part of a path. Additionally the place on which the path ends (the agent place) and the place on which the path starts (the anchor node), afflict the reliability. Let *i* be the length of the path (i.e. the number of proxy nodes plus the anchor node and the agent node) and $r_{n_k}$ the reliability of a node *k*. The reliability *R(i)* of the path can then be calculated by the following equation:

$$R(i) = \prod_{k=1}^{i} r_{n_k} \qquad \textit{(Equation 3)}$$

If we simplify the equation by assigning the same reliability $r_n$ to every node in the path, then we get the following simple expression:

$$R(i, r_n) = r_n^{\,i} \qquad \textit{(Equation 4)}$$

The reliability leads directly to the MTTF of the path. Figure 5 shows the development of the MTTF with increasing path length and three different failure rates $\lambda$ for the nodes of 0.10, 0.02 and 0.01. The graph shows that, if e.g. each component of the path has an MTTF of 50 hours (failure rate $\lambda$ of 0.02/hour), the MTTF for a path of length 50 is only 1 hour.

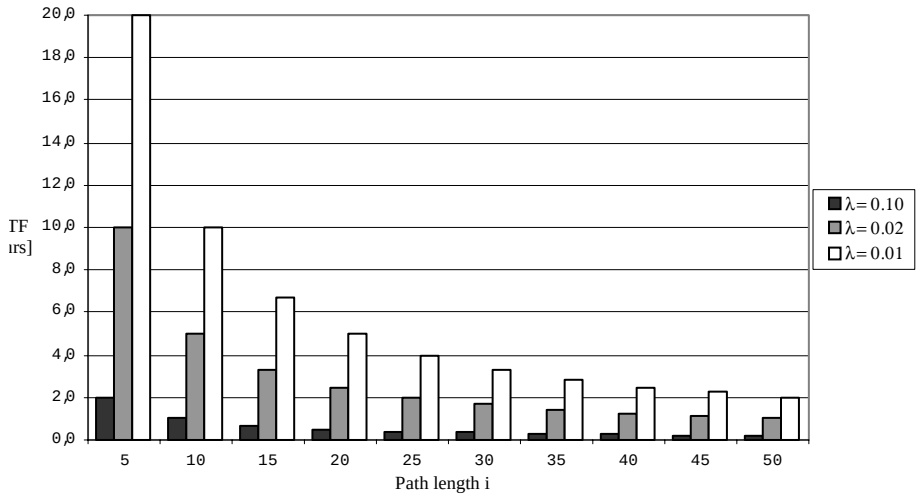


*Figure 5: Path concept: the MTTF of a path*

If we actually want to follow the path, then the reliability of the underlying communication network has to be regarded also. Let us make the same assumption as with the different nodes along the path, i.e. the communication channels between two nodes have all the same reliability $r_c$. If the path has $i$ nodes, then communication channels are needed;

between the path nodes (including anchor and agent node), and 1 each between inquirer and anchor node, and inquirer and agent node. Assuming an exponential distribution this leads to:

*(Equation 5)*

From this follows the MTTF of the path:

*(Equation 6)*

Let us now make the following additional assumptions: the path information is stored on stable storage, and the MTTR (mean time to repair) for both communication channels and nodes is 0.5 hours. We get the availability :

$$\alpha_{i,\lambda} = \frac{MTTF}{MTTF + MTTR} = \frac{\frac{1}{i\lambda_n + (i+1)\lambda_c}}{\frac{1}{i\lambda_n + (i+1)\lambda_c} + 0,5} \qquad \textit{(Equation 7)}$$

$$= \frac{1}{1 + 0,5(i\lambda_n + (i+1)\lambda_c)}$$

If we make a final assumption, namely that the failure rate for nodes $\lambda_n$ and the failure rate for communication channels $\lambda_c$ is of the same value $\lambda$, then the availability $\alpha_{i,\lambda}$ can be illustrated as in Figure 6. We can see that with an MTTF for the single components of the path of 50 hours (failure rate $\lambda$ of 0.02/hour) and an MTTR of 0.5 hours, the availability $\alpha_{i,\lambda}$ for a path of length 50 (i.e. 48 hops by the agent) is only 0.5. Even doubling the MTTF of the components leads only to an availability $\alpha_{i,\lambda}$ of 0.66.

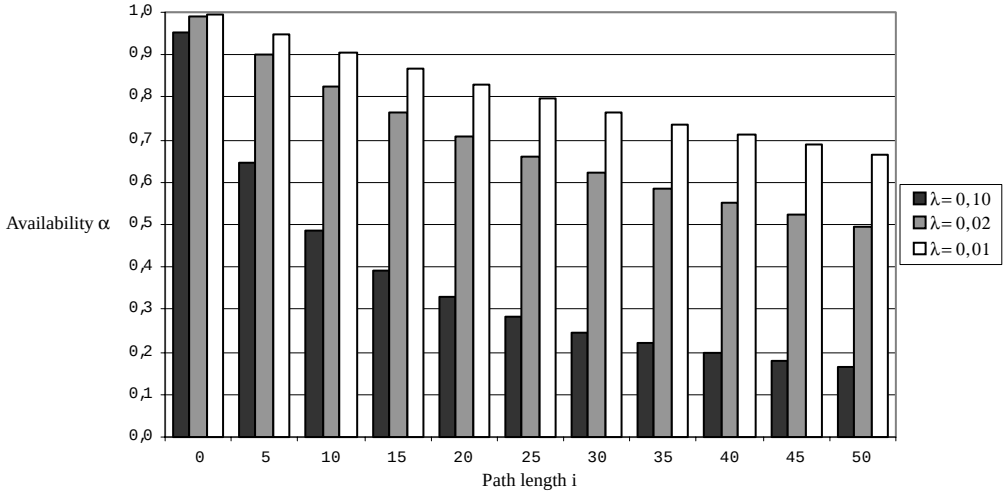


*Figure 6: Path concept: the availability of a path*
*(path information on stable storage, MTTR = 0.5 hours)*

A small remedy is the specification of a maximum path length as proposed by ARIDOR AND OSHIMA (1998), i. e. the path is shortened automatically after a certain number of hops. The disadvantage is the additional communication for shortening the path, i.e. a message from the source place of a migration to the anchor place, and then "shortenPath"-messages along the path, leading to $n + 1$ additional messages every $n$ hops. This mechanism can be used to recreate the path in the presence of faults, after the agent has made the $n$th hop. The problem is that it is unforeseeable when the $n$th hop takes place, if ever (i.e. the worst-case time bound for MTTR is still the MTTR of the single devices).

### 6.1.2 Timed Path Proxies

Again the proxy path (including anchor place and agent place) is the determining factor. By introducing the *ttl*, after which the agent has to contact the anchor place, it is guaranteed that even if the path is broken, the new place of the agent is known after the *ttl* has passed (as a worst-case bound), as long as either the network partition is short-term, or agent place and anchor place are in the same partition. Let us first examine the availability of the path. We again assume the same failure rate for all nodes containing a proxy. But with the automatic update of the path we essentially have a repair functionality with a maximum time to repair that equals *ttl*, and a minimum time to repair that equals 0, i.e. a mean time to repair that equals $\frac{ttl}{2}$. Thus we get the following availability $\alpha$ :

$$\alpha = \frac{MTTF}{MTTF + \frac{ttl}{2}} \qquad \textit{(Equation 8)}$$

If we make the same simplifications as before, namely one failure rate for all nodes $\lambda_n$ and one for all communication channels $\lambda_c$, we get the following first equation for the availability $\tilde{\alpha}_i$ in dependence of the path length *i*:

$$\tilde{\alpha}_i = \frac{MTTF}{MTTF + \frac{ttl}{2}} = \frac{\frac{1}{(i\lambda_n + (i+1)\lambda_c)}}{\frac{1}{(i\lambda_n + (i+1)\lambda_c)} + \frac{ttl}{2}} \qquad \textit{(Equation 9)}$$

$$= \frac{1}{1 + \frac{ttl}{2}(i\lambda_n + (i+1)\lambda_c)}$$

What hasn't been included yet is the shortening of the path. The availability as computed with Equation 9 assumes that after the contact with the anchor place the path length stays the same, when in reality it is shortened to 2 (anchor place and agent place). If we assume that the agent movement is linear, i.e. it stays on every place for the same time, then we yield the following equation (the average of the availability for all the different path lengths):

$$\alpha_i = \frac{1}{i-1}\sum_{k=2}^{i}\tilde{\alpha}_k = \frac{1}{i-1}\sum_{k=2}^{i}\frac{1}{1 + \frac{ttl}{2}(k\lambda_n + (k+1)\lambda_c)} \qquad \textit{(Equation 10)}$$

Let us now make the following additional assumptions (the same as in Section 6.1.1): firstly, the failure rate for nodes $\lambda_n$ and the failure rate for communication channels $\lambda_c$ is of the same value $\lambda$, and secondly, all agents have a *ttl* of 6 minutes, i.e. every agent contacts its anchor place 10 times an hour. Now the availability $\alpha_i$ can be illustrated as in Figure 7. We can see that e.g. with an MTTF for the single component of 50 hours (failure rate $\lambda$ of 0.02/hour), the availability of a path of length 50 (i.e. 48 hops by the agent) is 0.95. Thus even if, with these assumptions, an agent migrates nearly 50 times every 6 minutes, the probability of an inquirer being able to contact it is still 95%.

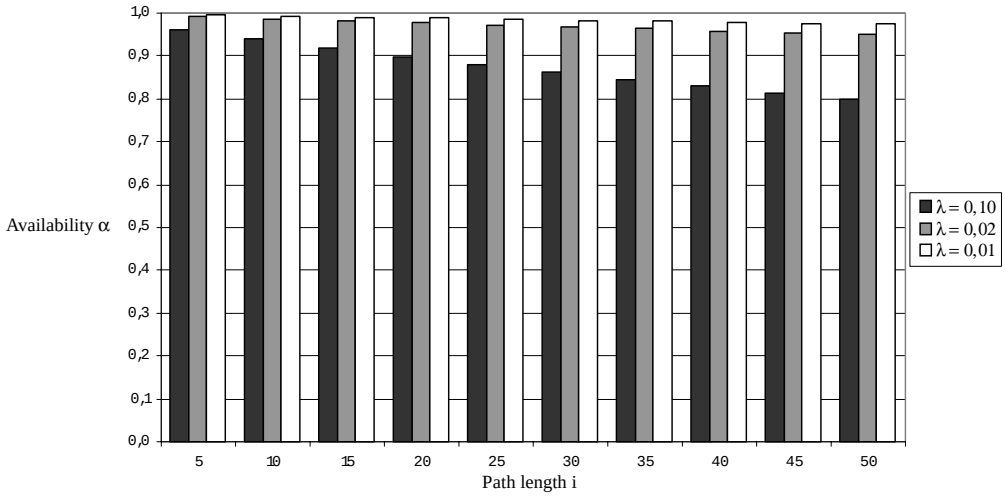


*Figure 7: Timed Path Proxies: the average availability of the path (ttl = 0.1 hours)*

## 6.2 Message Complexity

When examining the message complexity of the different mechanisms we have to distinguish between the message complexity for maintaining the information and the message complexity for actually using the mechanism.

### 6.2.1 Maintenance

**Advertising.** Since the message cost depends entirely on the decisions of the programmer when to advertise the new location, the message cost cannot be given precisely. But if the agent's location is never advertised, the message cost is 0, and if the agent's location is advertised every time it migrates, the cost is 2 messages per agent per migration (i.e. one message advertising the new location, and one acknowledgment message).

**Energy.** The message cost added by the protocol is 2 messages per agent per granted energy request, and 1 additional message for a denied request. If we assume that the common usage of the energy concept is for agents that wait for something, e.g. for a specific change in a remote database, then the request can be piggy-backed on the message signalling the change. This reduces the message cost to 1 message if additional energy is granted and to 0 if denied.

**Untimed Path Proxies.** Since the path information is maintained locally, no messages are needed to maintain the path. Thus the message cost for maintaining untimed path proxies is 0. Additional messages are needed though to remove the path of a no longer existing agent (the same cost as for using the mechanism to cut the path short). Shortening the path involves sending a "shortenPath" message along the old, superfluous path. This leads to a cost of $n + 1$ messages for a path length of $n$.

**Timed Path Proxies.** As has been described in Section 4.1.2.1 the path is shortened in regular intervals, i.e. every time the agent's *ttl* has dropped to 0. Shortening the path includes one message from the agent to the anchor place (the update message) and one from the anchor place to the agent (the acknowledgment message). The intermediate path proxies are removed without additional communication. This leads to a cost of 2 messages per agent and *ttl*.

**Database.** The database (global or local) is updated every time the agent migrates. Thus a message is needed for the update, and furthermore an acknowledgment message is needed to guarantee that the information has been added to the database. Thus the message cost is 2 messages per agent per migration.

**No Logging.** Since no information is maintained, the maintenance message cost for these mechanisms is 0.

**Probing.** Since no information is maintained, the maintenance message cost for these mechanisms is 0.

### 6.2.2 Using the Mechanism

**Advertising.** Locating an agent involves inquiring at the database, i.e. one request message and one message answering the request, if the information is not outdated. Otherwise the agent is not locatable.

**Energy.** Locating an agent involves inquiring at the database, i.e. one request message and one message answering the request, if the information is not outdated. Otherwise the agent is not locatable.

**Untimed Path Proxies.** A search request is forwarded along the path, and answered by the agent node sending a message back to the initiator node. If the path contains $n$ proxies, then $n + 1$ messages are sent along the path.

**Timed Path Proxies.** The message complexity is the same as with untimed path proxies, namely $n + 1$ messages along the path.

**Database.** Locating an agent involves inquiring at the database, i.e. one request message and one message answering the request.

**No Logging.** As has been pointed out already, the message complexity of sequential brute force and of parallel brute force differs. In the case of sequential brute force the number of messages for locating the agent on average equals the number of places, since on average the agent is found after half the places have been found (and each inquiry needs a request and a reply). With parallel brute force the message volume equals twice the number of places. This can be reduced (as has been discussed in Section 4.1.2.2) to the number of places plus 1 (if only a positive reply is sent back).

**Probing.** In the best case the first message probing for the agent locates it, and in the worst case the number of messages equals the number of places to be visited by the agent depending on the algorithm used for probing. If for instance a binary search algorithm is used, then the number of messages in the average case equals $ld(n)$.

### 6.2.3 Overall Message Complexity

To simplify comparing the different mechanisms we define three different degrees of message complexity: *low, medium* and *high* message complexity. Using the results of the above discussion for the message complexity for maintenance and for locating agents and our assumption about the possible size of the mobile agent system from Section 3.2 we yield Figure 8.

Since the overall cost can only be given correctly if the application is known exactly, we simplify again by defining concentric zones of low, medium and high message complexity. Following this classification the energy approach is of low message complexity, advertising, timed path proxies and untimed path proxies are of middle message complexity, and database approaches and approaches using no logging have a high message complexity.

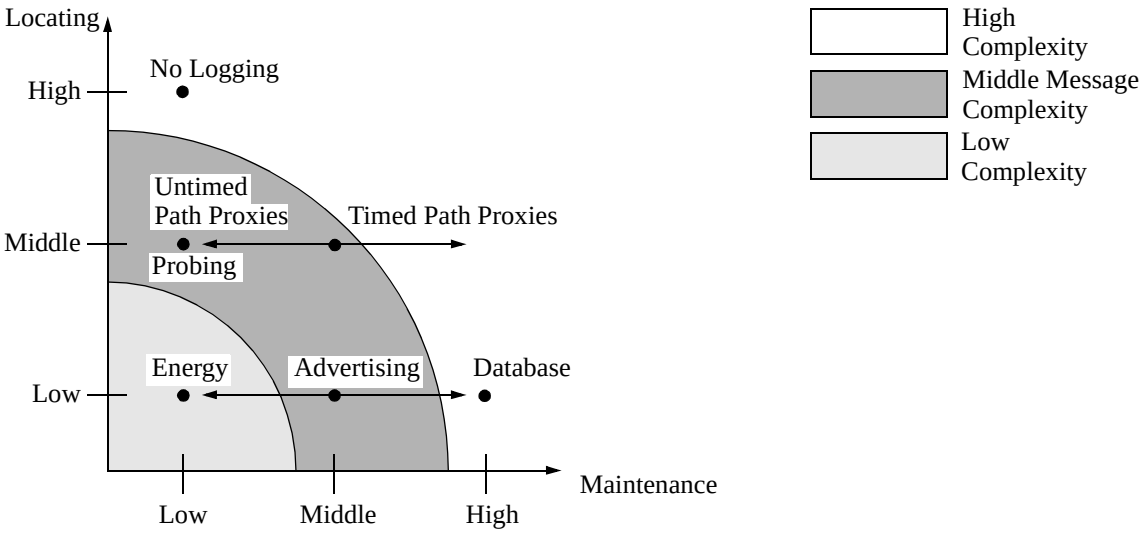


*Figure 8: The message complexity of the different mechanisms*

## 6.3 Migration Delay

If a mechanism for locating agents restricts the migration operation, then it violates the autonomy of the agent, and thus the mobile agent paradigm (constituting mobile agent autonomy). Hence we will examine the different approaches regarding their intrusion upon the migration. Let us distinguish the following three classes of migration delay:

- every migration is delayed. Mechanisms of the first class are those that force a contact with a third party before the agent migrates, i.e. the agent is delayed until the answer (the acknowledgment) is received. An example for this class is a mechanism using a database for registering the new location of the agent, that waits for the acknowledgment at the source place before allowing the migration.
- subsequent migrations are delayed. This class contains all those mechanisms that contact a third party after the migration takes place, i.e. the agent has to wait for the answer on the target place, and subsequent migrations can only take place after receipt of this answer. An example for this class is a mechanism using a database, that forces the agent to wait at the target place for the acknowledgment.
- migrations are not delayed. The third class of mechanisms never delays the agent, either because no third party is involved, or because no acknowledgment is needed. An example for this class is a mechanism using untimed path proxies.

We combine the first and second class (and name it *first/following*), since mechanisms of these classes can be transformed into mechanisms of the other class by sending the acknowledgment to the source place of the migration (to convert mechanisms of the second class to mechanisms of the first class) or to the target place (to convert mechanisms from the first to second class). The third class we name *no-delay.*

Advertising is in the first/following class, since every time the agent advertises its location it has to wait for the acknowledgment that the information has been received. All those mechanisms updating databases for every migration delay the migration (the update has to be acknowledged), i.e. they are also in the first/following class.

The energy concept never delays migration (we ignore the rare case of having to request additional energy) and is thus in the no-delay class. The mechanisms implementing untimed path proxies are in the same no-delay class (they never delay the migration; only a proxy is left on the source place). Timed path proxies (i.e. Shadows) also do not delay the migration (here the infrequent case that the path is shortened is ignored), hence they are in the no-delay class as well. Lastly, the mechanisms using no logging (i.e. Brute Force) and the probing mechanisms are of the no-delay class, because these mechanism do not modify the migration operation at all.

## 6.4 Merging the Assessments

Let us assume the following conditions: a failure rate $\lambda$ of 0.02/hour and an MTTR of 0.5 hours, a *ttl* of 6 minutes for the timed paths and 50 migrations of an agent during this time. With these values we get the following results for the availability of the different mechanisms:

- For advertisement, energy, centralized and distributed registration, sequential and parallel brute force, and the probing mechanism the availability according to Equation 2 is 0.96.
- For paths and logging the availability with the above conditions is 0.50 (see Figure 6).
- The shadow concept has an availability of 0.95 with the above conditions (compare with Figure 7).

Let us draw a line at 80%, i.e. an availability of 0.8 and higher is deemed as a high availability, and an availability of lower than 0.8 is considered a low availability (actually the results would be the same for any percentage between

51% - 94%). If we now combine the assessments regarding availability, message complexity and migration delay, we yield Figure 9.

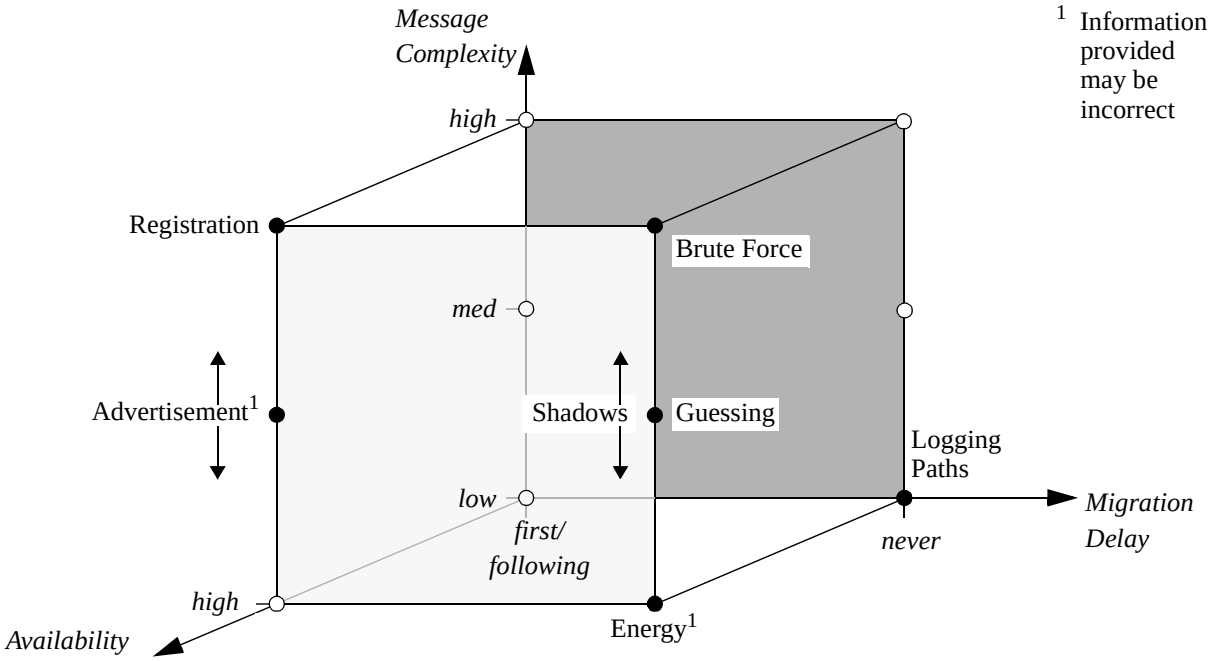


*Figure 9: Control mechanisms: combining the assessments for existing mechanisms*

It can be observed that the mechanisms implementing untimed paths (i.e. the path concept and the logging mechanisms) provide only a low availability. But their advantage is their low message complexity (i.e. no messages are used for maintaining the path) and that they do not delay the agent migration.

The energy concept seems to have perfect qualities, i.e. low message complexity combined with high availability and no migration delay, but it has the big drawback, that the information provided may be incorrect. The same drawback holds for the advertisement mechanism, making these two unsuitable for locating agents reliably.

The registration mechanism has a high availability, but its drawbacks are a high message complexity and the delay of the agent migration.

The brute force mechanism has the same high availability and does not delay the agent migration, but its message complexity is very high (at least half the number of places in the agent system). This makes the mechanism unusable in an environment as large as the one we assumed in Section 3.2.

The guessing mechanism has medium message complexity, high availability and does not delay the agent migration. But this mechanism assumes preordained paths, i.e. it changes the agent model. Whether this constraint is acceptable has to be decided by the agent programmer, but it definitely makes this approach a less general one.

The shadow mechanism is more general, since it makes no assumptions about agent migration. At the same time it has a high availability, medium message complexity and introduces no delay on migration. Furthermore, the message complexity of the shadow concept is adjustable even at runtime, allowing the mechanism to react to a changing environment.

The results of comparing the existing mechanisms for locating agents are:

- Brute force has a extremely high message complexity and is unusable in real-world mobile agent systems.
- The energy concept and the advertising mechanism are unsuitable for locating mobile agents reliably.
- The registration mechanisms (both with global and local databases) have a high message complexity compared to other mechanisms, i.e. they should not be used as long as alternatives exist. Furthermore, they delay the agent migration, thus interfering with the autonomy of the agent.
- The mechanisms employing the path concept provide only a low availability compared to the other discussed mechanisms. This more than negates their advantages of low message complexity and of no interference with agent autonomy (i.e. no delay of agent migrations).
- The guessing mechanism is only usable in situations where the constraint put on the agent model, i.e. that the agent migration follows a preordained path, is acceptable.
- The shadow mechanism needs only moderate message complexity compared to most of the other mechanisms while providing a high availability and without interfering with the agent autonomy.

# 7 Conclusion

In this paper we have discussed different possible solutions for locating agents in a mobile agent system. We have introduced a classification of these possible approaches and have categorized mechanisms proposed in the MASIF standard and mechanisms implemented in the Aglets Workbench and in Mole. The comparison of the different mechanisms yielded as a result that the shadow mechanism (implemented in Mole) provides the best combination of properties (i.e.

message complexity, availability and non-interference with the agent autonomy) for the general usage. Furthermore this mechanism provides termination and orphan detection for mobile agents, an additional feature that none of the other mechanisms provides (see BAUMANN AND ROTHERMEL (1998B) for the details).

The other mechanisms can still be the better choice for specific applications that provide additional knowledge. In these cases e.g. the advertisement scheme might perform much better by using the application-specific knowledge to decide when to update the location information.

# 8 References

**ARIDOR AND OSHIMA (1998)** Aridor, Y. and Oshima, M. (1998), "Infrastructure for mobile agents: requirements and design", in *Proceedings of the Second International Workshop on Mobile Agents '98*, K. Rothermel, F. Hohl, Eds., Lecture Notes in Computer Science 1477, Springer-Verlag, Berlin, Germany, pp. 38 - 49.

**BAUMANN (1997)** Baumann, J. (1997), "A protocol for orphan detection and termination in mobile agent systems", Technical Report Nr. 1997/09, Faculty of Computer Science, University of Stuttgart, Germany.

**BAUMANN *ET AL.* (1997)** Baumann, J. and Hohl, F. and Radouniklis, N. and Rothermel, K. and Straßer, M. (1997), "Communication concepts for mobile agent systems", in *Proceedings of the First International Workshop on Mobile Agents '97*, K. Rothermel, R. Popescu-Celetin, Eds., Lecture Notes in Computer Science 1219, Springer-Verlag, Berlin, Germany, pp. 123 - 135.

**BAUMANN AND RADOUNIKLIS (1997)** Baumann, J. and Radouniklis, N. (1997), "Agent groups for mobile agent systems", in *Distributed Applications and Interoperable Systems*, H. König, K. Geihs and T. Preuß, Eds., Chapman & Hall, London, UK, pp. 74 - 85.

**BAUMANN *ET AL.* (1998A)** Baumann, J. and Hohl, F. and Rothermel, K. and Straßer, M. (1998), "Mole - concepts of a mobile agent system", *WWW Journal 1*, 3, Baltzer Science Publishers, pp. 123 - 137.

**BAUMANN AND ROTHERMEL (1998B)** Baumann, J. and Rothermel, K. (1998), "The Shadow approach: an orphan detection protocol for mobile agents", in *Personal Technologies 2*, 3, Springer-Verlag, London, UK, pp. 100 - 108.

**BECK (1997)** Beck, B. (1997), "Terminierung und Waisenerkennung in einem System mobiler Software-Agenten", Diploma Thesis Nr. 1472, Faculty of Computer Science, University of Stuttgart, Germany.

**CARRIERO AND GELERNTER (1989)** Carriero, N. and Gelernter, D. (1984), "Linda in context", *Communications of the ACM 32*, 4, pp. 444 - 458.

**CHEN AND LENG (1997)** Chen, W.-S. E. and Leng, C.-W. R. (1997), "A novel mobile agent search algorithm", in *Proceedings of the First International Workshop on Mobile Agents '97*, K. Rothermel, R. Popescu-Celetin, Eds., Lecture Notes in Computer Science 1219, Springer-Verlag, Berlin, Germany, pp. 162 - 173.

**IBM (1999)** IBM Tokyo Research Laboratory (1999), "Aglets Workbench: programming mobile agents in Java", web page, URL: http://www.trl.ibm.co.jp/aglets

**JALOTE (1994)** P. Jalote (1994), *Fault Tolerance in Distributed Systems*, PTR Prentice Hall.

**JUL *ET AL.* (1988)** Jul, E. and Levy, H. and Hutchinson, N. and Black, A. (1988), "Fine-grained mobility in the Emerald system", in *ACM Transactions on Computer Systems 6*, 1, pp. 109 - 133.

**LANGE AND OSHIMA (1998)** Lange, D. B. and Oshima, M. (1998), *Programming and Deploying Java Mobile Agents with Aglets*, Addison-Wesley, Reading, Massachusetts.

**MILOJICIC *ET AL.* (1998)** Milojicic, D. and Breugst, M. and Busse, I. and Campbell, J. and Covaci, S. and Friedman, B. and Kosaka, K. and Lange, D. and Ono, K. and Oshima, M. and Tham, C. and Virdhagriswaran, S. and White. J. (1998), "MASIF: the OMG mobile agent system interoperability facility", in *Proceedings of the Second International Workshop on Mobile Agents '98*, K. Rothermel, F. Hohl, Eds., Lecture Notes in Computer Science 1477, Springer-Verlag, Berlin, Germany, pp. 50 - 67.

**PAULUS (1998)** Paulus, M. (1998), "Agentengruppen für mobile Agenten", Diploma Thesis Nr. 1664, Faculty of Computer Science, University of Stuttgart, Germany.