In: Proceedings of the 5th IEEE International Conference on Mobile Agents (MA'2001), pages 152-168, Atlanta, GA, USA, December 2001. Springer LNCS, Vol. 2240.

Using Predicates for Specifying Targets of Migration and Messages in a Peer-to-Peer Mobile Agent Environment

Klaus Haller and Heiko Schuldt

Swiss Federal Institute of Technology (ETH)
Institute of Information Systems
ETH Zentrum
CH-8092 Zürich, Switzerland
{haller|schuldt}@inf.ethz.ch,
WWW home page: http://www-dbs.inf.ethz.ch

Abstract. Mobile agent systems are a powerful approach to develop distributed applications since they migrate to hosts on which they have the resources to execute individual tasks. Existing mobile agent systems require detailed knowledge about these hosts at the time of coding. This assumption is not acceptable in a dynamic environment like a peer-to-peer network, where hosts and, as a consequence, also agents become repeatedly connected and disconnected. To this end, we propose a predicate-based approach allowing the specification of hosts an agent has to migrate to. With this highly flexible approach, termed *P2PMobileAgents*, we combine the benefits of execution location transparency with those of code mobility. Similarly, also the recipients of messages can be specified by predicates, e.g. for synchronisation purposes. For providing meta information about agents and hosts we use XML documents.

1 Introduction

Mobile agents are a programming paradigm for distributed systems. In particular, this approach tries to reduce the communication costs and to evade the problem of network latency. Mobile agents consisting of code and an execution state are transferred from host to host to achieve these goals. They move in order to process the data available on hosts they reside on instead of sending the data to the host which is processing them. Hence, mobile agents are individual software entities performing tasks autonomously while hopping from host to host, which know on which host they find the data they need. With this point of view in mind, mobile agents require only simple mechanisms for messaging among themselves or choosing a host at run-time – since the latter already has to be defined when a mobile agent is coded.

On the other side, mobile agents are proposed for executing workflows since their early days [5]. Semantically corresponding steps are coded within a single mobile agent, so each agent can be considered as an independent transaction.

		Code and Data Mobility	
		no	yes
Execution Location	no	”simple” program	Mobile Agent
Transparency	yes	RPC	P2PMobileAgent

Table 1.

However, when processing takes place on shared resources or data, also synchronisation of different, originally independent mobile agents is required. Under this perspective, sophisticated methods for specifying targets of messages, especially for synchronizing agents accessing shared resources, and for migration are essential.

Existing mobile agent systems usually deal with the problem of implementing basic technologies, such as strong migration, which are however still subject to research. Instead, we focus on extending an existing mobile agent framework with sophisticated methods for specifying message targets and hosts an agent should migrate to. To be generic, we use predicates for specifying agents and hosts in a declarative way.

Also, agents are commonly seen as autonomous entities being able to cooperate in a bilateral way. Thus, there is no particular need for centralized services. Because of this, a peer to peer approach for agent cooperation is obvious.

Consider the following example, which we will use throughout the paper: An agent interested in buying stocks wants to figure out a place with a stock exchange agent in order to watch the stock prices and eventually to buy stocks. Moreover, it is interested to move to the place with the lowest load.

To point out in which way this adds new ideas to the mobile agents world, we want to compare our approach, called *P2PMobileAgents*, with other mobile agents and with remote procedure calls as illustrated in Table 1. We can use two criteria for classification: Code and data mobility on one side and execution location transparency, meaning that the programmer does not have to know on which host (sub)tasks are going to be executed in future, on the other side.

Mobile agents belong to the class that represents code and data mobility, but no execution location transparency is provided, since a programmer has to specify explicitly where he wants the agent to migrate to. Conversely, remote procedure calls (RPCs) hide the fact that a code fragment is executed on a foreign host. But in contrast to mobile agents, code is not passed through the network, only parameters. So execution location transparency is supported, but neither code nor data mobility.

Our *P2PMobileAgents* as an extension to the mobile agent approach certainly support code and data mobility, but they additionally offer execution location transparency. Places, to which an agent has to migrate to continue its execution, are specified using predicates. The *P2PMobileAgents* framework evaluates such predicates at run time, and so combines the advantages of mobile agents and RPCs. The remainder of this paper is structured as follows: We present an overview of the messaging and migration mechanisms of existing mobile agent systems and the possibilities for specifying message recipients.

Therefore, we introduce our system architecture in Section 2. In Section 3, we discuss the query language which is used for describing the features of entities (places and agents), before we explain how queries are evaluated (Section 4). We conclude this paper with a short summary in Section 6, after discussing related work in Section 5.

2 System Architecture

Within the $P2PMobileAgent$ project of the Database Research Group at ETH, we are implementing a mobile agent platform providing the possibility to choose and specify agents and hosts in a peer-to-peer environment by using predicates. In what follows, we present our system architecture.

Our system consists of places on which static and dynamic agents are executed. Resources available on places are encapsulated and can only be used via static agents bound to particular places. A peer-to-peer network supports communication between different places. Additionally, meta information for all agents and places is provided.

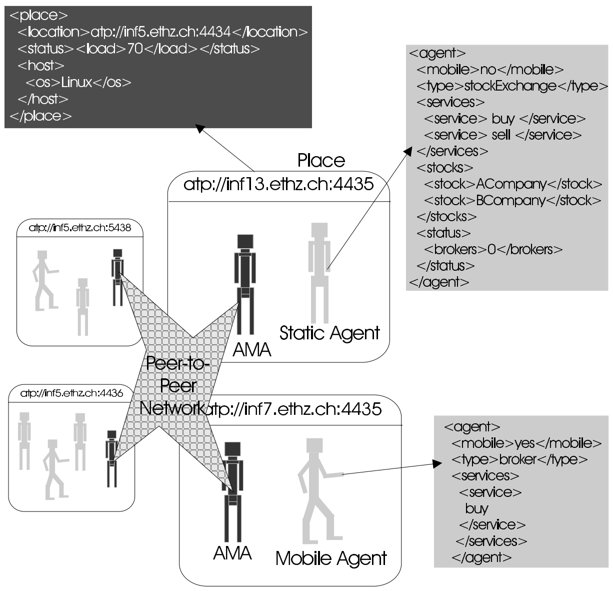
An example is shown in Figure 1: There are places connected by a peer-to-peer network and also both static and a mobile agents running within these places. The static agent existing at the place identified by "$atp://inf13.ethz.ch:4435$" represents a stock exchange and supports the buying and selling of stocks. In addition, there is a mobile agent at the place "$atp://inf7.ethz.ch:4435$", a broker offering the service "buy stocks".

We assume that the broker agents task is to buy stocks of the "ACompany". This requires him to monitor the trend of this stock and to be able to react immediately. Especially the last requirement demands that the agent is executed on the same place as the stock exchange agent.

For this reason, the broker agent has to migrate to a place with such a static stock exchange. Therefore, the agent has to find out via the peer-to-peer network which place is appropriate for its particular demands. The information about places and agents, called meta information in the remainder of this paper, has to be provided by each place and agent, respectively.

The task of managing meta data is realized by an additional type of agent, one per place, called *AgentsManagementAgent*, AMA (see Figure 1). The concept of an AMA is essential for the implementation of our new features: Together, all AMAs form a peer-to-peer network. Additionally, they are responsible for the meta data management. It is subject to the next two subsections to explain how these AMAs realize this task, before certain system issues are discussed in the last subsection.

Already at this place, we want to point out that no programmer using the $P2PMobileAgent$ framework notices the AMAs: They are started with the places. Also, communication with these AMA is transparent for mobile agent programmers, since the latter use an interface providing the extended capabilities of our system.



```
<place>
  <location>atp://inf5.ethz.ch:4434</location>
  <status><load>70</load></status>
  <host>
    <os>Linux</os>
  </host>
</place>
```

```
<agent>
  <mobile>no</mobile>
  <type>stockExchange</type>
  <services>
    <service> buy </service>
    <service> sell </service>
  </services>
  <stocks>
    <stock>ACompany</stock>
    <stock>BCompany</stock>
  </stocks>
  <status>
    <brokers>0</brokers>
  </status>
</agent>
```

```
<agent>
  <mobile>yes</mobile>
  <type>broker</type>
  <services>
    <service>
      buy
    </service>
  </services>
 </agent>
```

Fig. 1. Agent System

2.1 Meta Data Management and Querying

The *P2PMobileAgent* system allows to specify destinations for migration or messages by predicates. Therefore, meta data describing agents and places is needed. This subsection describes how this meta data is managed, focusing on the aspect of both structure and storage.

Structuring the Meta Data A key factor for allowing sophisticated queries to select places and agents is to structure the meta data needed for that purpose appropriately. To find the most suitable solution, the characteristics of this kind of information have first to be classified.

Typically, meta data comprises the description of services an agent provides or the actual load of a place. For this kind of information, a global schema can be defined. However, if groups of agents cooperate to solve a problem, it is sometimes important to be able to make information about their internal state available to the public. Yet, no universal schema can be found for this kind of information, since this particular state information differs from agent to agent not only with respect to the individual instances but also with respect to the granularity.

Nevertheless, to be able to process queries efficiently, we cannot rely on unstructured data. Hence, storing the meta data in a semistructured way is sensible. In particular, we use a subset of XML (eXtended Markup Language) [13] for this purpose, which doesn't contain the conept of parameters.

We assume that agents have to have a shared knowledge about the structure of the internal information (and in particular using a common vocabulary) available from other agents they wish to interact and cooperate with. Alternatively, ontologies would be needed, but this is definitely not the objective of our work.

Although the amount of information encapsulated may vary from agent to agent and from place to place, there is a minimal set of information mandatory for agents (i.e., their type, that is wether or not they are mobile) and for places (e.g., their address), respectively.

Figure 1 also shows such XML documents storing semistructured data. They consist of entities enclosed in tags which also can be used in a nested way. We want to illustrate this concept by looking at the static agent representing a stock market. The entity $< agent/stocks >$ with its subentities $< stock >$ names the stocks which are subject to trade. Knowing this structure and the tags exploited, every other agent can access and use this information.

Meta Data Storage and Processing Information about the place itself is managed by the AMA while information about agents are provided by the agent. Hence, this illustrates that there are two possibilities for query processing: it could be done by the agents or the AMAs.

If the individual agents evaluate the queries, agents become more complex – and a lot of code is redundant for all agents. Currently being present at some place, they would need information about all other agents of the same place. Additionally, the load of the agents would rise, because they cannot simply concentrate on their normal task. Concentrating the query evaluation on the AMAs allows us to benefit from the fact that they can aggregate information from all local agents making query evaluation faster. This applies especially if a query is not restricted to the information about either a single place or a single agent but also includes information on several agents. "Give me the place where an agent of the type stock exchange is executing!" is a good example for this type of query.

To evaluate queries, the required information has to be made available to the AMAs. Therefore, we have to distinguish two different kinds of information: static and dynamic.

A piece of information of an agent or place is called static if it cannot change during its whole life cycle, e.g. the services an agent provides or its type (see for instance the entities $< agent/services/service >$ or $< agent/type >$ in Figure 1). This kind of information can be cached by the AMA to improve the efficiency of the system.

On the other side, there is dynamic information which cannot be cached by the AMA. Instead, every time an agent has to be asked by the AMA to deliver the current value when the latter needs this information for query evaluation purposes. An example of this dynamic meta data is the number of brokers connected to a stock exchange broker (see $< agent/status/brokers >$ in Figure 1).

Dynamic information is always part of the element $< status >$, so that the system can determine whether data is dynamic or static. If there are different kinds of agents, some may store the same information as static whereas others store them as dynamic information. Because the same kind of information should

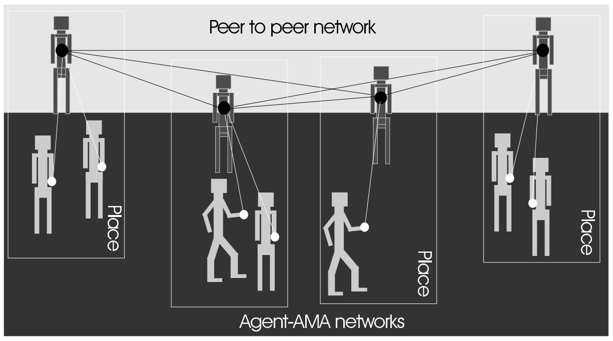


Fig. 2. Network Structure

always be addressable by the same tag name, the tag level $< status >$ is masked for queries. So there is no need to know (for queries) whether the data is static or dynamic.

2.2 Peer-to-peer Network

The goal of mobile agents issuing queries over places and other agents is to support their migration within the network. To this end, we want so support an open environment, since we are not interested in network topologies with any kind of central management. So, a peer to peer network fits to our requirements. All nodes are equal in this topology and have to provide the same functionality. Usually, not every node is aware of all the other nodes, but sees only a fraction of the whole system. Hence, communication between two nodes might involve several intermediate nodes.

This network is used for querying on agents and places and not for direct communication. This service is provided solely by the AMAs, so there is no reason that every "normal", i.e., mobile agent participates. Thus, the network is spawned by the universe of all AMAs.

Therefore, our system has two layers as shown in Figure 2: There is a client/server-like communication between agents of one place and their AMA and a peer-to-peer network between the AMAs of the different places.

Communication between an agent and its AMA is set up when the agent is started on the place, independently whether it is a new agent or whether the agent has just migrated to this place. The reason is that the AMA caches static information about the agent as it is described in Section 4. After the termination of an agent or after its migration to another place, the connection to the AMA is closed.

2.3 Implementation Details

Our system of $P2PMobileAgents$ is based on Aglets [9], the mobile agent framework developed by IBM which evolved to an open source project. Using this

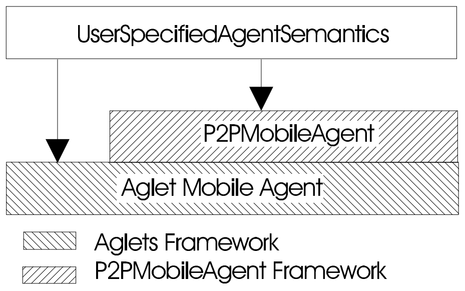


Fig. 3. Agent Layers

framework makes it possible to concentrate on implementing new concepts for specifying communication and migration targets instead of dealing with basic problems of mobile agents.

The Aglets framework provides mechanisms for weak migration and for sending messages. Each Aglet has a unique ID and by knowing this ID of an agent, communication mechanisms of the framework can be used for sending messages directly to it. In addition, the Aglet system allows to receive events when something important happens, e.g., when a new agent is launched or arrived at a certain place. These places are usually Tahiti servers, a program also being part of the Aglet system.

In this subsection, we concentrate on how the functionality for sending messages and migration based on the specification of targets using predicates can be added to the existing framework by using its basic features. The most important question is how to integrate the functionality into the existing Aglet framework. For this reason, it is either possible to extend and modify existing classes or to build a new component on top of the existing one.

In order to facilitate maintenance and since it is more convenient for programming, we have chosen the latter possibility with the restriction that the extended functionality for agents is realized by adding two new methods to the Aglet class. These methods implement mechanisms to send messages, and to migrate to targets specified by predicates.

In short, we realize the$P2PMobileAgent$ system by adding a new layer to the Aglet system. This layer provides support for querying on agents and places.

Figure 3 illustrates this for the case of an agent query. In the Aglet system, an agent inherits from the class Aglet. In the$P2PMobileAgent$ system, a new subclass is introduced, which provides the functionality for sending messages to agents and for migrating to places both specified by predicates. All mechanisms of the Aglet framework are not hidden and still available. In Figure 4, we illustrate the layered architecture for places. The context class of the Aglets system is the basis, on top of which the Tahiti server is running. Similar to the agent case, there is still the possibility to address these tiers by agents. In the $P2PMobileAgent$ approach, we added a new tier, the AgentsManagementAgent level.

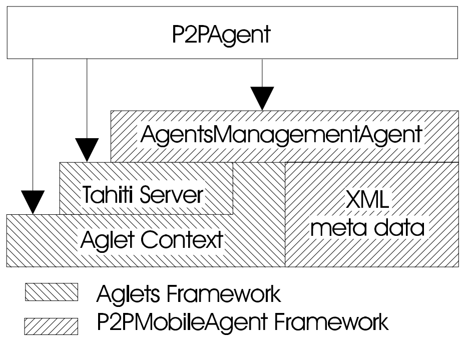


Fig. 4. Place Layers

AMAs are also Aglet agents that use the services provided by the lower level, especially the event handling and communication primitives and that integrate them them with the new XML meta data to provide the ability to query about agents and places. This functionality is used by the new layer of the agents that implement the new features for sending and migration.

In a peer-to-network, the Aglet system provides the possibility to send messages to agents if their IDs and addresses are known, but linking the places together to share knowledge about places or agents is not provided by the Aglet system framework.

So this peer-to-peer network formed by the AMAs is used for communication during the processing and evaluation of queries on hosts and agents, but not for communication between agents. If a message is to be sent to a particular agent fulfilling the query conditions, the agent is determined and its proxy is returned to the agent having initiated the query. Having a proxy, it is possible to send messages directly to the agent, although it might migrate in the future without indirection via the AMA peer-to-peer network.

3 Query Language

In this section, we address how to access the meta data. Therefore, we define a query language. The task of this query language is to find agents or places with particular properties. The result set of such queries consists of places an agent should migrate to or of a set of agents which are the recipients of a message.

In the context of querying XML documents, various query languages have been proposed, [8] is a compilation and comparison of the most important ones. Rather than relying on a fully-fledged XML query language, we follow a slightly simplified own approach. The reason is, that existing query languages provide a set of sophisticated features which considerably exceed the requirements of our $P2PMobileAgent$ approach.

The features of the existing query languages are:

1. Queries consists of a pattern clause, a filter clause and a constructor clause, including the possibility for information passing between different clauses. Also, nesting, grouping, indexing, and sorting is supported.
2. A join operator.
3. Tag variables and path expressions.
4. Handling of alternatives.
5. External functions like aggregation, etc.
6. Navigation operators for dealing with references.

In contrast, these are the requirements of the $P2PMobileAgent$ system:

1. Specifying which documents of which places are of interest.
2. How to deal with the problems of a peer-to-peer environment.
3. Only flat result sets and no need to build up complex types.
4. Simple expressions like predicates describing, e.g., agent tag values.

These requirements reflect that in our case, the information itself is stored in a distributed way in a peer-to-peer network. The existing query languages are well suited for complex queries to be evaluated on top of a single database. In our case, the queries itself are less sophisticated. Instead, we need to deal with a complex environment.

So, of course, a full fledged query language could also be used – after extension – for this purpose but with the drawback of carrying on an considerable amount of unneeded overhead.

For these reasons, we have constructed a query language tailored to the $P2PMobileAgent$ requirements. This query language supports two levels in the definition of a query. These levels are discussed in the two following subsections: The meta level deals with the environment, describing what documents are of interest under which circumstances. In the second subsection, we present how to specify the criteria driving the selection of entities that form the result set and by this, representing the target agents or places for messages or migration purposes. Then, we present an example of such a query.

3.1 Formulating a Query: The Meta Level

Prior to the evaluation of a query against an XML document, the appropriate place or agent, respectively, managing this document, has to be found under certain constraints. These aspects are subject to the meta level part of a query.

The four following criteria describe the meta level of a query and are mandatory for each query:

1. *result type:* The result set of a query is either formed by hosts or by agents, depending in which context the query is used: recipients of a message have to be agents, whereas places are expected in case of migration.
   RESULT_TYPE::= AGENT | PLACE

2. *cardinality of result set:* Depending on the kind of problem, an agent issuing a query might only be interested in receiving at most one element in the result set, e.g., if it is booking a flight, it wants to send the message to only one travel agency agent. Instead, if the agent wants to receive offers before booking a flight, it is interested in contacting all travel agencies. So we also have to consider the possibility to retrieve all entities complying with the specified condition. To be able to restrict the costs for executing a query, an agent can even restrict the number of entities in the result set to some individual maximum value.
   CARDINALITY ::= ONE | ALL | MAX(int)
3. *search space:* The concept of mobile agents is based on the assumption that it is cheaper under certain conditions, to transfer code instead of data. So it is useful to be able to restrict communication to the local place, although it might be required to search the whole peer-to-peer network in other cases. Considering peer-to-peer networks with the dimension of the Internet, it is reasonable to allow a more differentiated granularity then "local" or "global". For this purpose, we also introduce the possibility to restrict the number of tiers the message should be forwarded to. The number of tiers thereby represents the number of intermediate AMAs having forwarded a request.
   SEARCH_SPACE ::= LOCAL | GLOBAL | TIERS(int)
4. *time-out interval:* In a large and dynamic network, it is not reasonable to introduce a protocol that allows a book-keeping mechanism to decide whether or not all places have evaluated a query. Instead, it is better to define a time-out interval that restricts how long an agent having initiated a query accepts answers that shall be added to the result set before it proceeds its execution. The parameter in this case is an integer value that specifies how many milliseconds the agent waits for new results at most. Depending on the values of the "cardinality" and "order criteria" parameters, it is even possible that the agent is able to proceed its program execution earlier.
   TIMEOUT ::= MS(int)

3.2 Formulating a Query: Specifying the Selection Process

Additional criteria for carrying out a query are conditions and order criteria. The latter ones describe which entities shall be chosen if too many entities fulfil the conditions.

1. *conditions:* A condition consists of one or more predicates. Such predicates can be combined by using boolean operators. A predicate specifies expected element values for the agent (only if the result is an agent) or for places, thereby making it possible to specify requirements of places or to restrict the set of agents that are evaluated to the agents on the same place. Additionally, it can be required that only entities (places resp. agents) are of interested, if other agents reside on the same place.
   CONDITION ::= PREDICATE | (PREDICATE LOG_OP PREDICATE)|
   NOT (PREDICATE)

PREDICATE ::= PLACE(EXPRESSION) | AGENT(EXPRESSION)* |
EXISTS_ALSO_AGENT(EXPRESSION)
EXPRESSION ::= EXPRESSION | NOT (EXPRESSION) |
(EXPRESSION LOG_OP EXPRESSION)
LOG_OP ::= AND | OR
EXPRESSION ::= VALUE COMPA_OP VALUE
VALUE::= CONSTANT | TAG
COMPA_OP ::= > | < | =
*only if $RESULT\_TYPE$ is AGENT

2. *order criteria:* If the cardinality of the result set is not "all", it is possible that more entities fulfil the requirements specified in the field "condition". To this end, we introduce order criteria, like the ORDER-BY-clause in SQL queries [6]. This allows us to find the best fitting entities. Tags are used as sorting criteria for the result set. If the result set consists of places, only tags of the place are allowed, if the result set consists of agents, tags of the agent itself and also tags of the place can be used. In the latter case, these tags are dressed by "place.*", e.g. "place.os".
ORDER_CRITERIA ::= Tag (ASC | DESC) [, Tag ....]

3.3 Example of Place Query

In this subsection, we continue the discussion of the example started in Section 2: there is a mobile agent representing a broker, which should migrate to a place where a static agent exists that implements a stock exchange. In Figure 5, a query is presented that formulates this requirement in a query.

Because the agent wants to migrate, it is interested in a host and thus specifies "PLACE" in the field "RESULT_TYPE" (1). It is only desired to move to one place and not to duplicate itself to migrate to different places, so the value of the field "CARDINALITY" is set to "ONE" (2).

There is no restriction on how many places should be involved in this query, so the whole peer-to-peer network is specified as the search space (3). Nevertheless, the agent wants to consider only places that are found within the 15 ms timeout interval (4).

Then the requirements for the acceptable places have to be specified: our mobile broker agent needs a particular agent on the destination place, so it uses a predicate of the kind "EXISTS_ALSO_AGENT" (5). The tag "TYPE" is required to be "StockExchange" (6). Additionally, this agent has to be static (7) so that it is guaranteed that it is still there after the mobile broker agent has migrated to the new place. Moreover, the stock exchange has to be in operation (8).

It is possible that different places fulfil these conditions. If the agent issuing the query does not specify any order criteria, the first place fulfilling these conditions would be chosen as migration target. But in case it aims at migrating to the host with the lowest load, this has to be added as an order criterion (9).

```
RESULT_TYP= PLACE  (1)
CARDINALITY=ONE   (2)
SEARCH_SPACE=GLOBAL  (3)
TIMEOUT = 15  (4)
CONDITION = EXISTS_ALSO_AGENT  (5)
  ( type = StockExchange AND  (6)
    mobile = NO AND  (7)
    status.open = YES )  (8)
ORDER_CRITERIA = load ASC  (9)
```

Fig. 5. Sample Query

4 Query Execution

This section concentrates on the evaluation of queries. Especially, it deals with the interaction between agents and their AMA.

First, we discuss in detail how searching for agents is performed. Since searching for places does not significantly differ, we only give a brief summary of these differences. In both cases, the query execution is embedded in the execution of a migration resp. messaging procedure invocation, which takes place synchronously. We illustrate these algorithms by continuing the example started in the last section.

4.1 Searching for Agents

Searching for agents takes place in the context of sending messages. There is no specification of a particular target agent but rather a description of the kind of agent the message should be delivered to. By using the presented query language (see Section 3), the matching agents have to be found. Therefore, the peer-to-peer agent system starts to search for such agents in the network.

First, the query is submitted to the local AMA. There, it is distributed to all AMAs on other places the local AMA knows (i.e., has direct links to). These AMAs also forward the query to all AMAs they know, until the upper bound of tiers is reached – if one is specified in the field "SEARCH_SPACE".

Aside of forwarding the query, each AMA also evaluates which of the agents at its place fulfil the specified requirements. To this end, it caches all static information it gets during the communication with newly launched agents, either static or mobile agents. But if dynamic information is required for evaluating a query, meaning that information appears within the tag <status> in the describing XML document, the local agent has to be asked by the AMA to deliver this information.

If an agent matches the requirements of the query, the information to contact it are delivered directly to the agent that initiated the query.

After the time-out interval is exceeded, the initiating agent does not expect to be informed about any further agents matching the criteria of its query meaning that they are ignored. It finally evaluates the query and sends the message to the selected agent(s).

A global unique ID is attached to each query, such that each AMA can store the last queries it evaluated. By this, communication and execution costs are reduced since it can be avoided that a query is executed and forwarded several times at/by the same place, because a place can be mostly reached via different paths in the network. But by means of the ID of a query which is evaluated against the IDs of executed queries, this phenomenon can be avoided.

The approach followed here has a side-effect: queries can only be evaluated on a snapshot of the overall system. Hence, agents currently migrating while a query is launched are not found, because at this moment they are not registered at any AMA implying that the snapshots need not necessarly be consistent. The AMA of the place they are leaving does not know them and even if it would know them, it would not be able to communicate. The new AMA the agent is migrating to does not know them because they have not started to communicate with each other.

4.2 Searching for Places

Searching for places is initiated whenever the host or the hosts an agent should migrate to are specified by a query. As a consequence, the starting point is the XML document describing the place, even though also references to agents are possible.

4.3 Example

In this section, we discuss the evaluation of our sample query presented in Section 3.3. Therefore, we first have to present a network configuration: As shown in Figure 6, there are four places with agents. On place 1, there is the agent initiating the query discussed above. On place 2, there is an agent running that encapsulates a database. On the places 3 and 4, there are different stock exchange agents running. Certainly, there is an AMA on every place.

At the time 0, the query is launched by the initiator agent by transfering it to its AMA. The AMA forwards this query to every AMA it knows and adds a globally unique query ID to this query. The AMAs on place 2 and 3 receive the query and both forward it to the other AMAs they know (except for the sender). Since all AMAs keep a log of the IDs of arriving queries, the new query can be discarded by places 2 and 3 when it arrives the second time, such that (only) place 4 gets newly involved into this query.

Now, each AMA first checks whether it has processed this query before by comparing the query ID with the last queries it has processed. After having evaluated the first condition of the query, the AMA on place 2 knows that it cannot fulfil the requirements because there is no stock exchange agent running on its place. The AMAs on place 3 and 4 notice that they have such a static agent, but they have to evaluate the third condition that requires that these agents are currently in operation. Since this is a dynamic information, the AMAs have to ask the individual agents. Both stock exchange agents are active so the AMAs on place 3 and 4 send their place ID to the AMA of the first place because these places fulfil the requirements of the initiating agent. They also send their actual load information because it is needed as an order criterion.

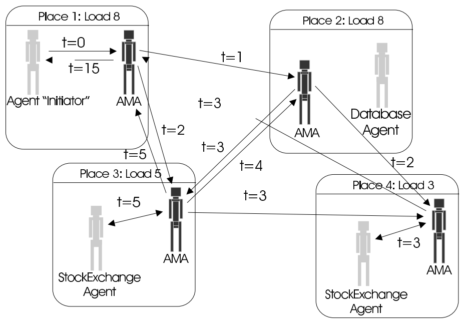


Fig. 6. Query Execution Example

After the time-out interval has passed, the AMA on place 1 evaluates the answers it has received. There are two possible places, but only one is desired, so it has to choose one out of them. Because place 4 has a lower load than place 3, the AMA of place 1 informs the initiator agent that it should migrate to place 4.

5 Related work

Our work contributes to two aspects: migration and communication. Both of them are vital to a mobile agent system, so nearly every mobile agent system published addresses these issues. Related work on these two topics is discussed in the following two subsections separately.

5.1 Communication

For the purpose of inter-agent-communication, several different approaches have been published under the term "coordination" or "coordination language".

Basic algorithms for finding mobile agents, e.g., using logging, registration, and advertisement for relocating are discussed in [1]. All together, they assume that the agent which is to be found is known – a different focus then that we have.

Besides sending messages to an agent with a known identifier (including multicast messages), also some concepts with a higher level of abstraction have been proposed.

The concept of events implies that an agent has to register for a special kind of event. After this registration, it is informed every time such an event occurs. That kind of event-based interaction is commonly referred as "publish and subscribe" [10]. This approach can be found, e.g., in Concordia [12], Mole [2], and – in a limited way – also in the Aglet system.

Sessions are an approach proposed in [2]. It supports 1:1 communication of agents that may stay on different places, but which are not allowed to migrate during a session is established. The most interesting idea related to our approach is the idea of badges: a set of strings is attached to every agent. It can be used to restrict the agents that are allowed to establish a session. As the most important difference to our approach, there is no structure within these badges and especially there are no tag value pairs like in XML which complicates sophisticated queries or even prohibits them. Also, in our case we support 1:n communication and there is no synchronisation needed between the communicating agents. This makes it much more easier to establish a connection if it is not known where the partners are.

The black board approach allows agent interaction with the help of a shared local data space. As a severe disadvantage, agents have to know the name of a certain blackboard in order to access the relevant information. A system supporting this approach is described, e.g., in [7].

An interesting special case of the black board approach is linda-like coordination as it is used for instance in the MARS [3] project. Here, associative methods are used for accessing information of the black board. Recently, it was decided to use XML for description purposes [4].

In order to summarize the above discussion, to our best knowledge, there is no other system offering such sophisticated methods for specifying agents on a high level of abstraction and without the requirement that a communication partner synchronizes or moves to the same place.

5.2 Migration

Taking a look at the second aspect, namely migration, we have not found any other system in the literature that allows for a declarative specification of the place a mobile agent should migrate to. Hence, the approach followed in our work where agents dynamically choose the place they should migrate to by using predicates is novel and considerably exceeds existing approaches in terms of flexibility.

6 Summary and Outlook

In this paper, we have presented a new approach for specifying both places to migrate to and agents being the recipients of messages. By using predicates, this specification can be done in a declarative and thus very flexible way.

The architecture is based on Aglets and extends this framework with new features like a peer-to-peer network, the possibility to describe agents and places using XML documents, and by allowing agents to query over these documents. Hence, it brings mobile agent systems and peer-to-peer configurations together.

Fundamental to our architecture is the concept of AgentsManagementAgents (AMAs), implementing the services mentioned above. AMAs are responsible for forming the peer-to-peer network, for the management of the meta data and also for the evaluation of queries.

In our context, queries consist of two parts: Firstly, queries comprise a meta level that describes how the evaluation process is driven. Secondly, predicates are used for driving the actual selection process.

Such queries are sent to the local AMA from agents interested in migrating to other hosts or in sending messages to other agents. This AMA pushes the query over the network to the other AMAs which also forward the query and evaluate whether or not this place or agents on this place fulfil the requirements of the query. Information about matching agents or places are sent to the AMA of the initiating agent (this is also important for maintaining the peer-to-peer network, i.e., for increasing the number of direct links to other places/AMAs). This AMA chooses the fitting entities.

With this approach, we support location transparency as well as code and data mobility and so combine the advantages of mobile agents and RPCs.

In our future work, we aim at adding additional guarantees known from databases to mobile agents. In particular, we are looking at these agents as a special transaction. Hence, agents accessing shared data need to be synchronized but also need support for failure handling strategies within the same framework. To this end, our goal is to apply ideas of transactional processes [11] to mobile agent systems.

The intended result will be a mobile agent framework allowing an easier way to program mobile agent groups. Combined with execution guarantees, it will extend the basic framework and is supposed to allow for new kinds of application in industrial strength.

References

1. J. Baumann: Mobile Agents: Control Algorithms, Springer, Berlin, Germany, 2000
2. J. Baumann, F. Hohl, N. Radouniklis, K. Rothermel, M. Strasser: Communication Concepts for Mobile Agent Systems, 1st Int. Workshop on Mobile Agents, Berlin, 1997
3. G. Cabri, L. Leonardi, F. Zambonelli: Reactive Tupel Spaces for Mobile Agent Coordination, 2nd Int. Workshop on Mobile Agents, Stuttgart, Germany, 1998
4. G. Cabri, L. Leonardi, F. Zambonelli: XML Dataspaces for Mobile Agent Coordination, Symposium on Applied Computing, Como, Italy, 2000
5. T. Cai, P. Gloor, S. Nog: DartFlow: A Workflow Management System on the Web using Transportable Agents, Tech.Rep. PCS-TR 96-283, Dartmouth College, 1996
6. C.J. Date, H. Darwen: A Guide to the SQL Standard, 3rd Edition,Addison-Wesley Publishing Company, Reading, MA, 1992
7. P. Dmel, A. Lingnau, O. Drobnik: Mobile Agent Interaction in Heterogeneous Environments, 1st Int. Workshop on Mobile Agents, Berlin, Germany, 1997
8. M. Fernandez, J. Simon, P. Wadler: XML Query Languages: Experiences and Exemplars, 1999, available from http://www-db.research.belllabs.com/user/simeon/xquery.ps
9. D. Lange: Programming and Deploying Java Mobile Agents with Aglets, Addison Wesley Logman, Reading, MA, 1998
10. R. Orfali, D. Harkey, J. Edwards: Client/Server Survival Guide, 3rd edition, John Wiley, New York, 1999
11. H. Schuldt: Transactional Process Management over Component Systems, infix, Berlin, Germany, 2001
12. D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, B. Peet: Concordia: An Infrastructure for Collaborating Mobile Agents, 1st Int. Workshop on Mobile Agents, Berlin, Germany, 1997
13. http://www.w3.org/XML