A Multi-agent Approach to Manage a Network of Mobile Agent Servers

Peter Braun
Faculty of Information and Communication Technologies
Swinburne University of Technology
Hawthorn, Victoria 3122, Australia
pbraun@it.swin.edu.au

Jan Eismann, Christian Erfurth, Arndt Döhler, Wilhelm R. and Rossak
Computer Science Department
Friedrich Schiller University Jena
D-07740 Jena, Germany
{eismann, cen, arndt.doehler, rossak}@informatik.uni-jena.de

Keywords: Mobile agents

Received:

In this paper we present an approach to construct and evolve a network of mobile agent servers. It can be seen as a service that is indispensable for mobile agents to move through the network automatically. Without such a service the programmer of a mobile agent must code the agent's itinerary into its business logic. Our approach has a two-level structure, where agent servers within a subnetwork are combined into a domain, and domains can be connected to each other using a client/server or peer-to-peer technique. Our approach is multi-agent based, that is several stationary and mobile agents communicate to each other to build and evolve the logical network. Main characteristics of our approach are its robustness in failure situations and its high performance, which is shown by results of a first evaluation.

Povzetek:

1 Introduction

Mobile agents are small software entities that can roam the Internet in order to fulfill a user-given task. They allow for task processing that is dynamically distributed over the network by searching for agent servers which offer appropriate services in a network of interconnected platforms [8]. In the last years mobile agents have been a very fast growing area of research and development. While most research was done in the area of code security [22], control algorithms [1], and mobile agent coordination [20], we consider mobile agents to be foremost a promising design paradigm for the architecture of distributed systems.

Many mobile agent toolkits have been developed, for example Aglets [15] by IBM, Concordia [17] by Mitsubishi, and Grasshopper [2] by IKV++. All these toolkits can be considered to have almost product status. Nevertheless, with regard to specific basic system services most mobile agent toolkits are still in their infancy. As an example, many standard services found in middleware components for distributed applications, as for example CORBA [12], are missing even in the above mentioned toolkits. Our aim is to introduce standardized services for distributed agent server networks and to evaluate these services in the Tracy [8] mobile agent toolkit.

1.1 Logical Agent Server Networks

The basic concept we employ is that of a *logical agent server network*. An agent server is the environment on a single computer system that allows the receipt and execution of mobile agents. We define a logical network as an undirected graph in which vertices represent agent servers and an edge exists between a pair of vertices if there is the possibility to transmit mobile agents between the corresponding servers. Not all agent servers must be able to exchange mobile agents due to different transmission protocols, firewalls or private subnetworks that are only reachable via a gateway server. A logical network is a necessary prerequisite for a mobile agent to move through the network automatically. On each server it can ask through a stationary agent, or a service, for the neighboring agent servers and decide to which it will migrate to next.

Without such a network service the agent's programmer has the obligation to code the agent's itinerary into its business logic. While this is sufficient in some applications and in small networks, it is not reasonable to define an agent's route in a world wide network, for example. In such an environment mobile agents must be able to find their itinerary on their own. They must be in a position to react on unreliable network connections and unreliable agent servers and, therefore, modify their itinerary on the fly.

In its minimal variant, a logical network is equal to the

minimal spanning tree guaranteeing that each agent server can be reached. The drawback of this solution is the on average higher number of edges on a path between two nodes, which results in a higher number of migration steps and therefore in a higher execution time of the agent. A smaller number of edges on a path between two nodes results from redundant edges. In a network where all agent servers are connected with each other, called a clique, the maximal number of edges on each path equals one. Both solutions lack any structure that could support the agent in optimizing its itinerary while traversing.

A logical agent server network is the foundation for more sophisticated services of this kind. They all need information about neighboring agent servers or possibly all agent servers currently reachable. One of the research topics is to develop algorithms to plan (semi-)optimal routes for mobile agents with regard to application capabilities (data, user-level services) offered on agent servers and network quality information.Currently we are developing a network performance measuring component for the Tracy toolkit. This component is to measure transmission time to other agent servers periodically and providing this information to mobile agents for planning their optimal route through the network. It uses information of the logical agent server network to find agent servers to which the network quality should be tested. On an even higher level of abstraction the logical agent server network is used to propagate information about applications offered on one agent server to other servers in the network. A mobile agent can use this information to plan its route according to the task that it should fulfill.

Currently, in all of the above mentioned mobile agent toolkits, it is only possible to manage a single *stand-alone* agent server by using some kind of console or graphical user interface. In a logical agent server network, the administrator can obtain an immediate view of all agent servers. In the Tracy toolkit there is already an approach to use this information in combination with its graphical user interface which can be dynamically connected to other running agent servers to administrate them, e.g. to start and stop agents or for checking the status of the server.

1.2 Similar Approaches

As far as we are aware, there is only one agent toolkit, Grasshopper, that offers a service that is distantly related to a logical agent server network. The *region registry* is a central component within a single domain to hold a list of all agent servers. It is not only used to store information about other agent servers, but also information about all agents that are currently residing on all agent servers within a region. Thus, the region registry can be described as a kind of agent tracking service which is also a good approach to find other agents to communicate to. However, multiple region registries cannot be connected, with the consequence that mobile agents cannot find agent servers beyond their region (local domain).

The Grasshopper approach is a good solution either for local area networks or for a small number of agent servers spread over a large region. It offers more than a logical agent server network insofar as it also provides information about the agents in the local domain.

Other agent toolkits, for example Jade [3], provide another approach. A Jade platform contains of several so-called *containers*. One agent server, named the *main container*, maintains a directory of all containers, agents, and services that are provided within a platform. When a new Jade container is started, the administrator has to decide, whether this agent server shall work as main container. Otherwise, the address of the main container has to be defined to let the new agent server register with it afterwards.

1.3 Our Solution

In this paper we present our approach to constructing logical agent server networks. The basic architecture of our approach to mobile agent networks consists of domains, which are limited to subnetworks. All agent servers within a single domain enlist at a central server, which is called *domain manager*. Domains can be connected to each other so that mobile agents can also reach agent servers in other domains. Connecting and disconnecting of agent servers to the network works fully automatic and dynamic.

One characteristic of our approach is its robustness in failure situations. For example we can guarantee that at any time there exists a domain manager for each domain. If a domain manager crashes (because its host agent server crashes) all remaining agent servers vote for becoming the new domain manager. If the original domain manager is relaunched, it can reclaim this role.

Our approach is multi-agent based. The domain management service is completely implemented using stationary and mobile agents. Is does not depend on any specific mobile agent toolkit. Although we have implemented our approach on top of the mobile agent toolkit Tracy it is designed to be portable to any other toolkit with minimal effort. For the rest of this paper, to ground our argumentation and our examples, we will use Tracy as a reference system.

Tracy is a general-purpose mobile agent toolkit. It was designed as an extendable toolkit that consists of a kernel and optional plugins. The kernel only provides those services that are similar in all agent toolkits, for example to execute agents and control agents' life-cycle. All high-level services like agent communication, agent migration, partial solutions of the agent security problem, etc. are implemented as additional plugins. We refer to [7] for more information about Tracy's architecture. A detailed introduction to use and program Tracy is given in [8].

1.4 Related Work

A logical agent server network can be seen as an overlay network and the inceptions of creating and maintaining overlay networks go back to the beginnings of the Inter-

net. The Internet was originally conceived as a peer-to-peer system, in which every network node was a peer and the overall number of nodes was very small. In the following, the Internet has grown enormously, making it impossible to maintain the huge number of nodes as in a peer-to-peer system. Overlay structures were conceived as indispensable means to keep the number of nodes manageable.

A well known example for a overlay structure of a peer-to-peer system is the Domain Name System (DNS) [18, 19], which divides the domain name space in a hierarchical manner and forms a distributed database system. The DNS as a whole works and scales very well, but it was designed for a static network with only few dynamics and only with immobile nodes.

In the late 1990s the upcoming file-sharing systems revive the use of the peer-to-peer paradigm. Napster [21] is a peer-to-peer system with a centralized search facility. This solution scales well in practice because of centralizing search while distributing download. However, centralized solutions lead to a single point of failure and cannot scale completely.

Gnutella [11], another peer-to-peer file-sharing protocol, came originally without any network structure and was a robust, fully decentralized approach. The search algorithm was a simple broadcast with hop limit mechanism (horizon) and led to high network load, so it scales not with the possible number of requests. A second drawback of this solution was the lack of known entrance points into the network at the first time of join Gnutella.

Present Gnutella clones and successors uses additional mechanisms to prevent the drawbacks of the original solution. eDonkey [9] and KaZaA [14] for example use high potential nodes as super-nodes which are connected to each other and thus form a backbone network. A super-node manages a couple of clients, their connections and search requests and forwards the requests to other super-nodes eventually. This approach joins central and decentral features to a hybrid network structure which may scale well, if the number of peers and super-nodes retains well balanced.

The Tracy domain service provides likewise a hybrid overlay structure. It consists of local centralized, quick-reacting domains, that can intercept high network dynamics. Domains can be loosely coupled to each other and are combined in a global hierarchical structure. Agent migration happens fully decentral as in a common peer-to-peer manner. This structure guarantees robustness against breakdowns and scalability in a wide range.

1.5 Structure of this Paper

The rest of this paper is structured as follows: In the following section we will present basic concepts of our approach. Sec. 3 contains a detailed description of our stationary agent which is used to build up the network management service. Sec. 4 focuses on the concept of priorities by which we model different types of agent servers within the network. In Sec. 5 we describe possible failure situations and their solution within our approach. Sec. 6 contains a concise description on how mobile agents can employ the network management service. Sec. 7 provides first results of performance measurements. Finally, the last section gives a summary and an outlook to further development.

2 Basic Concepts

In this section we introduce the basic concepts of our approach. We start by describing the topology of our logical network. Then we argue the usage of stationary and mobile agents for our approach.

2.1 Topology

The topology or architecture of the created logical network can be best described as an interconnection of several domains, see Fig. 1. One domain consists of several agent servers that are connected in one subnetwork and that are able to exchange mobile agents. This restriction derives from a feature of the Tracy mobile agent toolkit that allows mobile agents to be sent using several transmission protocols, like TCP, UDP, or SSL. Agent servers in one domain must, therefore, at least share one transmission protocol to communicate successfully.

Note, that in Tracy each computer system can host more than one agent server. Thus, the number of agent servers per domain is not limited to the number of computers per subnetwork (which equals 255). Usually, there is only one domain per subnetwork. However, several domains can be used, for example if there are agent servers without a common transmission protocol. Another way to split agent servers within a subnetwork into two domains is described later in Sec. 3.2.

In each domain there exists the so-called *domain manager node* which is an agent server, that is responsible to manage all other agent servers in this domain, which are called *domain nodes*. Every domain node has its unique domain manager node, and vice-versa the domain manager node knows all domain nodes that are currently active in its domain. If an agent server starts or stops, it has to register and deregister with the domain manager node. Compare Sec. 3 for more details.

A logical agent server network finally consists of several domains which are connected with each other via their domain manager nodes. Thus, no domain node has a direct connection to any agent server in another domain. There are several ways to connect domain manager nodes to each other. One is to use a central unique so-called *master node* to receive names of other domain manager nodes. We describe this and other techniques later in Sec. 3.2. As indicated in Fig. 1 there is a hierarchy of nodes in our network model: At the lowest level there are domain nodes that represent usual agent servers. A more specialized type of node is a domain manager, which is responsible to manage all nodes in a single domain. At the highest level there

is the master node which is responsible to manage domain managers. In the notion of object-oriented analysis we can state, that a master node is a domain manager node, and a domain manager node is a domain node.

2.2 Multi-Agent based

Our approach to constructing agent server networks is implemented as a multi-agent solution, where one corresponding agent–the domain information agent–exists on each agent server. This domain information agent can play the role of domain node, domain manager, and even master, as is necessary.

It is quite obvious to use agents for this task, as the whole solution is provided as a service in a mobile agent system. Another solution would have been to implement a new protocol on top of TCP or to use Java RMI. By utilizing agents we preserved the independence of the agent toolkit and offer a clear communication interface for other agents that want to utilize the provided information.

The domain information agent is a single stationary agent on each agent server. In the Tracy toolkit only stationary agents are permitted to access the local file system, open network connections, etc. in contrast to mobile agents which will usually not have these rights. This domain information agent employs mobile agents to connect and disconnect agent servers, to inspect network connections, and to inform agent servers in failure situations. The only exception is that we do not use mobile agents in the very first step, that is to find the domain manager node within the subnetwork. This is done by a multicast, see Sec. 3.2.

The reasons to employ mobile agents are the following: In Tracy, being a purely mobile agent toolkit, we are forced to use mobile agents to send messages between agent servers. Tracy does not offer the ability for agents to communicate to remote agents by sending messages (messages being different from agents). On the other hand, exchanging information between remote agents can be obtained by a very small mobile agent. Fortunately, there is no penalty for using mobile agents for those tasks in the Tracy system, because migration is performed very fast [6, 4]. Remote communication using Java Remote Method Invocation would not be faster. In addition, it would have lead to a complete new communication channel in addition to the core agent server, introducing more dependencies and basically redundant services.

There is one scenario that illustrates nicely the main advantage of mobile agents, that is moving code close to the data instead of moving the data to the code. When a domain manager node is in need of other domain manager nodes to be connected to, the master node can be searched for. It holds a list of other domain manager nodes which can be filtered according to some *neighbor* relation. As this data base can be very large in size, it is obviously a better strategy to choose suitable neighboring agent servers directly at the master node. The standard solution to offer a specific interface lacks the flexibility to define the term *neighbor*. Furthermore, to transmit the whole data base to the domain manager node would cause unnecessary network traffic. We decided to utilize mobile agents to find neighbors implementing the notion of a *neighbor* inside the agent's decision capability. This decision algorithm can be modified by the user and is therefore adaptable to different requirements.

3 The Domain Information Agent

This section gives a detailed description of the main component of our domain manager service, which is the stationary *domain information agent* (DIA). The DIA functions according to the roles assigned to it. Here, we will explain how a DIA determines its role during launching and present different techniques to connect domain information agents to each other.

3.1 Constructing Domains

As described in the last section each agent server holds a specific role in the logical agent server network: domain node, domain manager node, and master node. In our solution we provide only one type of agent, the domain information agent, which has to take care of each of theses roles. This method is obvious due to the necessity of changing roles dynamically in failure situations between a domain node and a domain manager node. For the sake of simplicity, we do not introduce new names for the DIA's roles. Thus, if an agent server is currently a domain manager node, the respective domain information agent has to provide the functionality of this higher level role, too.

The determination of the DIA's role is done semi-automatically when launching the agent. First, the agent is in the role of a domain node, assuming that there is a DIA on another remote agent server which holds already the role of the domain manager node. To find this domain manager node and later register with it, the new agent sends out a UDP multicast message to all computer systems in the same subnetwork. If there is a domain manager node in this subnetwork, it receives the multicast message and answers with a single UDP package containing its URL. This URL can be used to address migrations to. In the second step, the new agent now checks if both agent servers can exchange mobile agents by using the same transmission protocol. In the third step, the new agent sends a mobile agent to the domain manager node to register the new domain node over there. The mobile agent returns to indicate that the registration process was successful. This process works fully automatic and due to the usage of UDP messages and very small mobile agents, the whole registration process concludes in less than 40 ms on average in a 100 Mbit/s network (see Sec. 7 for more performance results).

If no domain manager agent has answered the UDP multicast message or both agent servers do not offer at least one equal transmission protocol, the new agent passes into

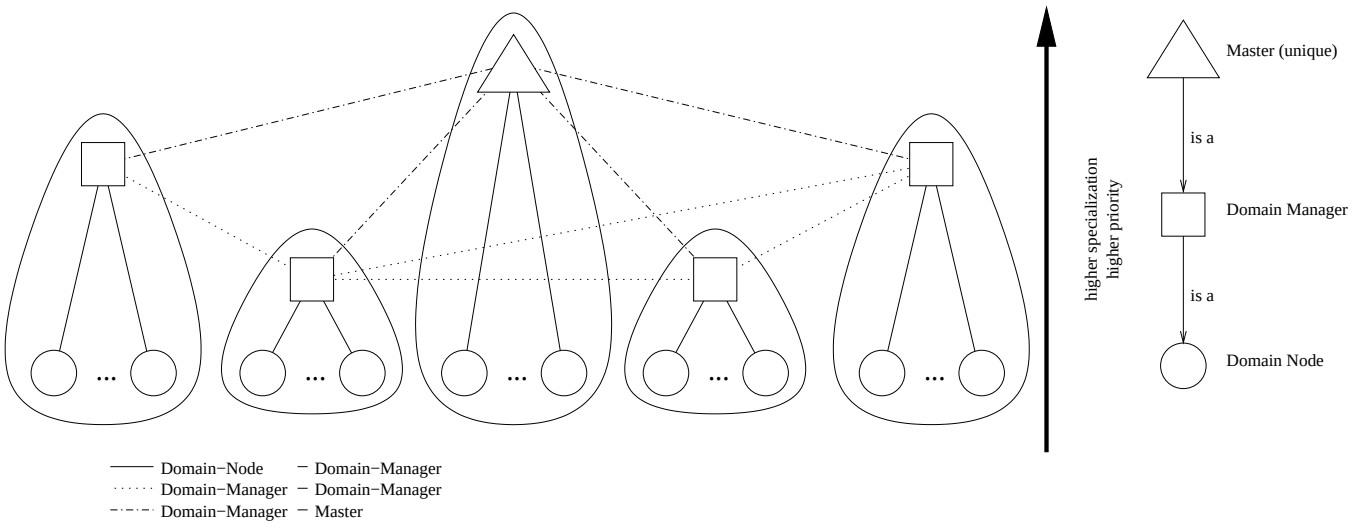

Figure 1: Topology of our logical agent server network. An edge between a pair of vertices indicates that the corresponding agent servers know each other.

the role of a domain manager node itself. As we mentioned briefly in the last section, it can occur that two domains exist in one subnetwork at the same time for the following three reasons: First, if the new agent sends out the UDP multicast message to another UDP port than the domain manager agent receives messages on, the registration process will not start. Second, both agent servers do not speak at least one equal transmission protocol. Third, UDP is an unreliable communication channel. The multicast message as well as the answer package may get lost with the effect that the registration process will fail. Actually, in our implementation, the UDP multicast is resent three times to compensate UDP's unreliability.

As a consequence of a failure in the registration process, a new domain is created within a subnetwork. Agent servers within this subnetwork could now be registered at two separate domain manager nodes. In the current implementation the choice of domain manager nodes is driven by the first-come-first-serves principle, that is the domain manager node that answers the UDP multicast first, is chosen. When the new agent is in the role of the domain manager node it has to connect to other domain manager nodes, see Sec. 3.2 for more information. The only drawback of having two domains in one subnetwork results from slightly increased migration times to agent servers in the other domain inside the same subnetwork. The agent must search for the agent server it wants to visit by first migrating via two domain manager nodes instead of reading the information locally at its current agent server.

The highest level role a domain manager agent may assume is that of the master node. Whereas the above described process works fully automatic, the decision on the master node is done manually by the administrator of the agent server that should become master node. A master keeps its role over its whole life-time. No other domain node or domain manager node may become master node. The master node is one specific node whose name is known to some/all domain manager nodes. From this master they can obtain information about other domain manager nodes to connect to. A master nodes skips the search for a domain manager node and accepts its role immediately. After that, it behaves identically to a domain manager node within its domain.

When shutting down an agent server, we have to distinguish two cases. If the agent server was in the role of a domain node, the domain information agent has to deregister from the domain manager node. If the agent server was in the role of a domain manager node (but not in the role of a master node), it has to hand over this role to another agent server in this domain. The simplest idea is that the current domain manager node randomly selects from the list of all registered domain nodes one agent server that will become the next domain manager node. With one mobile agent that visits all registered agent server within this domain, the resignation of the old and the taking over of the new domain manager node is reported to all domain information agents. In the current implementation the whole selection process is not implemented this way, but is driven by the priorities concept that we explain in Sec. 4. If the master node is shut down, no other domain manager will take over this role. Thus, if the master node is used by domain managers to connect to other domain managers, a master node failure will cause severe problems to maintain the inter-domain structure of the network.

3.2 Connecting Domain Managers

We are now going to present our approach to connect separated domains to one single network. This process is also designed to work fully automatic and allows for dynamic

adaptation of connections. Therefore, our solution does not enforce the construction of a specific network topology at this level of connection. It depends on the connection strategy which topology is going to emerge, for example to either build up a fully connected graph out of all domain manager nodes or to connect only neighboring nodes according to some kind of distance metric.

In the first approach we use the master node to obtain information about other domain manager nodes. After the domain information agent starts in the role of a domain manager it sends a mobile agent to the master node to filter the remote data base of known domain manager nodes. As we mentioned above, the usage of the mobile code approach in this context allows for effective and very flexible remote filtering. The definition of the notion *neighbor* is encapsulated within the mobile agent and thus can easily be adopted to local requirements at the domain manager node. The selection process also influences the degree of connectivity of the resulting graph.

If the mobile agent can obtain at least one neighboring domain manager node it is guaranteed that no islands will exist. The drawback of this approach is the client/server like architecture which contradicts the mobile agent philosophy to some extent. The master node is a bottleneck and single point of failure that should usually not exist in a mobile agent network.

We also support another approach to connect domain manager nodes, one which is comparable to the connection strategies implemented in peer-to-peer systems, like Gnutella [13] or FreeNet [16]. A fresh domain manager node must connect at least to one other domain manager node. This address can be defined by the administrator, or, including our master-approach, obtained by the master node.

During life-time, a domain manager node searches for other domain managers using mobile agents traversing the network. It selects other domain managers according to its distance metric. When shutting down, it saves the current list of neighbor in the local file system to reconnect to them at the next time being launched.

4 The Concept of Priorities

With the simple concepts introduced so far, some problems arise in realistic application situations. As can be deduced from the definition of roles within the logical network, the life-time and the quality of each type is different. We assume that a domain manager node has a longer life-time and a higher reliability than a domain node, which can be a mobile device using a wireless connection. The master's life-time and reliability is assumed to be even higher than that of a domain manager node. However, this idea cannot be found in the approach presented so far.

One shortcoming would result from the selection process which starts when a domain manager node is shutting down. Instead of choosing an arbitrary node this selection process should prevent that a short-living node or unreliable (e.g. on a mobile host) becomes domain manager for a foreseeable short time only. The other drawback is strongly related to this. If the domain manager node is restarted again, it should be able to take over the role of a domain manager node from the present one. Two agent servers starting accidentally at the same time would thus cause a collision problem that should be prevented.

We introduce now a concept of priorities to influence the role of an agent server within the logical network. The priority of a domain information agent is modeled as a value between -128 and +127. This priority is defined by the administrator before the domain information agent is started. It cannot be changed during the agent's life-time, at the earliest when the agent is restarted. The priority value should result from the reliability and long-liveness of this agent server. The higher the value is the more important is the role that this agent server may assume within the network (see also Fig. 1). The default value of an agent server equals 0.

With the concept of priorities, the launching process of a domain information agent changes slightly. When a new domain information agent receives the UDP packages containing the URL and priority information of found domain manager nodes (remember that several domain managers might exist in a single subnetwork), it now compares the priorities of theses nodes with its own. If its own priority is higher, the new node becomes domain manager. In the other case, it tries to register with one of these nodes starting with the one with highest priority.

If a new node becomes domain manager, a process of changing roles is started: A mobile agent is started to visit all domain manager nodes and notifies each to release its role and to fall back to the role of a domain node. Each node is informed about the new domain manager node, so that no new registration process is necessary.

When a domain manager node is shutting down, it selects the next domain manager from the list of all known domain nodes according to their priority. To inform the new domain manager and all connected domain nodes about the new situation, the same process starts as mentioned above.

To prevent that two agent servers starting at the same time become domain manager nodes, the priorities and the agent servers' names can be used. When receiving the UDP package containing URL and priority of another domain manager node, the new node can determine which node is going to take the role of a domain manager node by comparing the priorities of both nodes. If both priorities are equal, the first node according to lexical sorting of their URLs is selected.

5 Recovery from Failure Situations

The handling of failure situations is a very important issue in distributed systems. Typical challenges of mobile agent

systems are especially a consequence of the wide area network character of agent networks. In a mobile agent network it might happen very often that servers go down or are unavailable. As a consequence, agents must search for an other host offering the same services. For the management of a logical agent server network it is, therefore, very important to handle those situations appropriately. The technique to find out that neighboring servers are not available anymore is implemented by the use of mobile agents that are sent to the other server. Those so-called *ping agents* are sent from nodes to managers and from managers to other managers. Currently we do not distinguish between server and network failures and handle both situations identically as a server failure. The following failure situations are detected:

1. **Failure of a domain manager**. Each domain node periodically sends out ping agents to its domain manager. If the migration process fails for any reason, it is assumed that the domain manager is not available any more and that a new domain manager must be defined. The first node that has noticed this situation becomes a domain manager for a limited time, not considering any priority information. All other nodes within this domain will notice this failure situation within a pre-defined period of time and will find this temporary manager node to register with. This period of time equals the maximal time between sending two ping agents (see below). The temporary manager node collects all priority information and can be certain to select the node with the highest priority as the next domain manager. The exchange of roles is performed in the same way as described above in Sec. 3.1. Without the temporary manager node it might happen that exchanging roles between nodes takes place as often as nodes are in the domain.

2. **Failure of a neighboring domain manager**. If a domain manager detects that a neighboring domain manager is not alive anymore, it removes it from the list of known managers. Depending on the the strategy of finding other neighbors it might immediately request new neighboring managers from the master node. If the failed manager has been the only connection between two separate subnetworks it is possible that they become disconnected. Therefore it is important to specify a good minimal degree of connection for the network.

3. **Failure of a node**. If a domain manager detects that a node is not alive anymore, it is removed from the list of known domain nodes. If the node is restarted, it can register with the manager again.

If any of these failure situations can be attributed to a network failure instead of a node failure, it might happen that the alleged crashed node will be available again after a short period of time. With our approach we can achieve the expected behavior, that is to restore the old state automatically. For example, if the connection between all nodes and the manager node is broken (but the manager node is still alive), a new domain manager will be elected as described above. The disconnected manager notices stepwise that each node is not available any more and removes it from the list of known domain nodes. After the last node has been removed, the domain manager checks if there is another domain manager reachable by using an UDP multicast. If it finds another domain manager the process of comparing priorities and possibly exchanging roles starts. Otherwise, it stays as a domain manager, but tries to find another domain manager until a new domain node has registered.

A very important parameter which influences the quality of service of the logical agent network is the time between two ping agents (ping interval). If it is too long, the network will react sluggish and might hand out outdated information about the network. If the ping interval is too short, network traffic will increase.

If the master node concept is used to connect to other domain managers, a master node failure would cause severe problems to maintain the inter-domain structure of the network. Currently, we have not implemented any approach to handle this failure situation.

6 Usage

After we have illustrated how the network structure is constructed, we will now explain how the information is provided to mobile agents. It is again the domain information agent (DIA) that provides information about known domain nodes and domain managers. A mobile agent can gain this information either by exchanging messages with the DIA, or by observing the blackboard. A description of all types of messages, including the content of the reply message can be found in Tab. 1. All capitalized words are String constants defined in class `DomainInformationAgent`. The following example (Fig. 2) shows an agent that sends a message to the DIA to retrieve all domain nodes. The answer message contains a list of URLs in the message body, where entries are separated using three " `%` " characters. To parse this list of URLs, class `StringArray` in package `de.unijena.tracy.util` can be used. See the Tracy JavaDoc documentation for more information.

The second way to obtain information about the network structure it to observe specific entries on the blackboard. The blackboard is a hierarchically structured container for information to provide information that should be seen by all agents on a single agent server. The DIA publishes information about known domain managers and domain nodes on the blackboard in directories `System/domainmanager/managers` resp. `System/domainmanager/nodes`. Both directories contain sub-directories for each host.

```
import de.unijena.tracy.util.*;
import de.unijena.tracy.agent.*;
import de.unijena.tracy.agentsystem.*;
import de.unijena.tracy.comm.*;
import de.unijena.tracy.domainservice.*;

public class MessageExampleAgent extends MobileAgent{
  public void startAgent() {
    try{
      sendMessage(DomainInformationAgent.DOMAIN_AGENT_NAME,
                  DomainInformationAgent.GET_NODES,
                  null);
    } catch (MessageQueueException e){
      // DomainInformationAgent does not exist
    }
  }
   public void handleMessage(Message msg){
    if (msg.getTyp().equals(DomainInformationAgent.GET_ANSWER)){
      if (msg.getContent() != null){
        String[] servers =
                 StringArray.stringToStringArray(msg.getContent());
      }
    }
  }
  public void systemFailure(){
  }
}
```

Figure 2: An example of an agent that asks the domain information agent for the names of all domain nodes.

Request subject	Reply subject	Reply parameter	Description
GET_NODES	GET_ANSWER	List of URLs	Get a list of all nodes.
GET_MANAGERS	GET_ANSWER	List of URLs	Get a list of all managers.
IS_DM	IS_DM_ANSWER	DM_TRUE/DM_FALSE	Ask for the role.

Table 1: Messages to obtain information from the domain information agent.

Each host directory contains entries with the agent server name. For example, if agent server with name `swiss.uni-jena.de/piz-gloria` is a domain node, the blackboard contains an entry with name `System/domainmanager/nodes/swiss.uni-jena.de/piz-gloria`. Using the blackboard has the advantage to be able to observe single directories or entries and become immediately informed in case of any modification. Doing this, it is for example very easy to notice that new nodes have enrolled on this domain manager. See [5] for more information on how to access and observe the blackboard.

7 Quality of our Approach

The quality of an approach for managing logical agent server networks is influenced by the quality of the resulting network and the performance of the whole service in failure situations. The first issue will be a topic of further work, where we want to evaluate the resulting network quality with regard to different connection strategies and distance metrics. This research is mostly done on an application level, for example to evaluate the time for an agent to route its way through the network considering the information provided by the logical network.

For the moment, we have evaluated our approach with regard to the performance of the pure management services. Our results show on the one hand that our approach causes no measurable overhead for the whole network traffic and on the other hand that the network can react very fast in failure situations and can achieve a stable state within a very short period of time.

All our experiments were conducted in a cluster of 8 computer systems running Linux on a processor with 800 MHz and in a network with 100 MBit/s bandwidth. The UDP time-out is set to 100 ms. The ping interval is set to 1000 ms.

Our results are summarized in Tab. 2. Each experiment was repeated 10 times and given results are mean values. Times do not include start-up times of the agent servers. In experiment 1 a new agent server registers at an existing domain manager. The time consists of sending the UDP multicast message to find the domain manager, checking transmission protocols, and register with the domain manager by using a mobile agent. In experiment 2 a new agent server is started and there is no other domain manager in the subnetwork. Thus, it becomes a new domain manager after it has waited for three UDP time-out periods (which in sum are 300 ms). No mobile agent is used in this experiment. In experiment 3 the current domain manager is shut down and delegates the role of the domain manager to a known agent server. The mobile agent is used to inform all nodes of the new domain manager and to delegate the role to the new domain manager. In experiment 4 the current domain manager is manually stopped to indicate a failure situation. The measured time is needed for the other seven agent servers to notice this situation (after at last 1000 ms) and to vote for becoming new domain manager, including the whole process of changing roles using a mobile agent. During this process 24 migrations are performed in parallel to connect nodes to the temporarily domain manager, followed by 8 migrations to change roles. Finally, in experiment 5 a new domain manager registers at two other domain managers, located in other LANs in Jena, using the master approach. The master is located at Weimar university (network bandwidth 34 Mbit/s). The mobile agent is used to search for the neighboring domain managers and to register over there.

8 Using Domain Manager Functionalities to Propagate Services

A possible application of the domain manager concept is to use it as a basis to design an information service for agents. *Information service* in this context means to support agents to search actively for those services they need to fulfill their user-defined task. It is the agent that locates available services, maps them to its needs, and autonomously charts a best possible route through the network to reach them.

In general, services are provided in the network by an agency or an agent within an agency. They are distributed over the nodes in the available network. The idea is to improve the autonomy of agents in a way that is transparent to the end user. The proposed information service would make it possible for the end user to simply state *what* he or she wants the agent to do, instead of *how*. This means that the end user could avoid to program a dedicated route into the agent's code, a route that is based on a most likely incomplete and possibly outdated human perception of the network. The human owner of a mobile agent could leave it open where fitting information and processing capability is collected and utilized. The agent and its supporting infrastructure will take care of the rest.

Of course this means that each agent, and the network

Experiment	Description	Time [ms]
1	Register a single node with an existing domain manager	40
2	Start a new domain manager	442
3	Shut-down a domain manager	558
4	7 server vote to become new domain manager	1542
5	Register a new domain manager at two neighboring managers	655

Table 2: Inspected scenarios for the performance evaluation. Given times are mean value of 10 experiments each.

of agencies they use, must provide a couple of new capabilities: Advertise and describe available services; match services to the user's orders; locate the nodes where those services are available; find a possibly optimal route through the network to travel to all indicated nodes/agencies and actually trigger service execution before returning home with the desired result. We are currently designing and implementing support for the tasks that tackle advertising of services and location of the respective nodes. Matching is simply based on a fixed set of keywords and, thus, kept very simple. Routing is open to a variety of solutions, including graph-based algorithms or methods from artificial intelligence. In this short overview we will focus on the most basic problems, that is to advertise and locate services in a most likely dynamic network of inter-linked domains.

Domain nodes already offer various services to mobile agents, either themselves or through agents they currently host. These services are well known to all agents that currently reside at the domain node. However, to make these services known to all agents in the domain, each domain node has to publish a service list. This can in our environment be done by a relatively simple extension of the domain manager concept: Each node's service list is transmitted to the domain manager node using the already necessary exchange of mobile agents during the registration process. Additional information about services available on the domain node is simply attached to the registration agent. Ping agents are used to hold this information up to date. (Remember: Ping agents are used to check if neighboring servers are still reachable. Such agents are exchanged between the domain manager and the domain nodes on a regular basis - see section 5).

The set of transmitted service lists can be seen as a service map of the local domain that is placed at the respective domain manager node. Agents are now able to use this map to match services, chart a route, etc. To avoid the migration to the service map, that is the domain manager node, this map can be mirrored at every domain node, again by simply using ping agents. If there are connections to other domains available - this is already possible on the basis of links between domain manager nodes - a summary of a domain specific service map can be propagated to other domains.

We improve the service map a little bit more by collecting additional technical information, e. g. characteristics like bandwidth, latency etc., and adding them to the service map. This would help the agent to plan the trip more exactly by taking into account also quality and reliability of available connections between nodes. To achieve this task a net sensing module quantifies the line characteristics in certain intervals by performing measurement experiments. The extracted data is collected and stored on the local domain node as well as sent to other domain nodes. This can be done very effectively during such an experiment by using the packages that have to be sent over the network for measurement purposes anyway. The actuality of the line characteristics depends on the frequency of the measurement experiments. The more dynamic the network is, the higher the frequency should be. In addition, forecast modules may be used to calculate the next expected value for extracted characteristics.

At this point a complete map with services and line characteristics of the local domain is available. This map is provided to mobile agents on every node in the domain that hosts an agency. By now such an agent is not only able to chart a service-oriented route that allows it to support its task, but is also able to estimate migration times and determine an optimized migration strategy.

As a possible alternative to the specified map-building mechanism the whole process of creating the enhanced service map could be based solely on the infrastructure of net sensing modules. This would avoid the (mis-)use of register or ping agents, as proposed so far. As already done with line data, the service map could be transmitted to other domain nodes using the bandwidth experiment packages. This would minimize any coupling between the domain manager and the service propagation subsystems and provide a quick and reliable basic infrastructure for service propagation. Therefore, currently the implementation of net sensing modules is our highest priority. For more detailed information regarding the use of the Tracy domain service in this context we refer to [10].

9 Conclusions and Further Work

In this paper we introduced the concept of a logical agent server network, which is a necessary service for mobile agents to roam a network automatically. We have presented our approach which is multi-agent based. On each agent server a stationary domain information agent is responsible to provide information about neighboring agent servers to mobile agents. The stationary agent uses several kinds of mobile agents to communicate to other domain information agents on remote agent servers. We introduced the

priorities concept to model different kinds of agent server nodes and to influence to process of finding domain managers. We have implemented our approach successfully in the Tracy toolkit and we presented results of first performance measurements.

Currently, we are developing a network performance measurement tool which uses our logical agent server network to locate agent servers to test latency and bandwidth. This information is provided to mobile agents, which use them to determine a fast route through the network.

In a next step we will evaluate the quality of the evolved network. Therefore, we will take an application level view and compare different connection strategies for domain managers and their influence on the performance of mobile agents traversing the network.

References

[1] Joachim Baumann. *Mobile Agents: Control Algorithms*, volume 1658 of *Lecture Notes in Computer Science*. Springer-Verlag, 2000.

[2] Christoph Bäumer, Markus Breugst, Sang Choy, and Thomas Magedanz. Grasshopper — A universal agent platform based on OMG MASIF and FIPA standards. In Ahmed Karmouch and Roger Impey, editors, *Mobile Agents for Telecommunication Applications, Proceedings of the First International Workshop (MATA 1999), Ottawa (Canada), October 1999*, pages 1–18. World Scientific Pub., 1999.

[3] Fabio Bellifimine, Giovanni Caire, Agostino Poggi, and Giovanni Rimassa. Jade – A White Paper. *EXP in search of innovation*, 3(3):6–19, 2003.

[4] Peter Braun, Jan Eismann, and Wilhelm R. Rossak. Various Performance Experiments with the Mobile Agent System Tracy. Technical Report 11/01, Friedrich-Schiller-Universität Jena, Institut für Informatik, 2001.

[5] Peter Braun, Christian Erfurth, and Wilhelm R. Rossak. An Introduction to the Tracy Mobile Agent System. Technical Report Math/Inf/00/24, Friedrich-Schiller-Universität Jena, Institut für Informatik, September 2000.

[6] Peter Braun, Christian Erfurth, and Wilhelm R. Rossak. Performance Evaluation of Various Migration Strategies for Mobile Agents. In Ulrich Killat and Winfried Lamersdorf, editors, *Fachtagung Kommunikation in verteilten Systemen (KiVS 2001), Hamburg (Germany), February 2001*, Informatik aktuell, pages 315–324. Springer-Verlag, 2001.

[7] Peter Braun, Ingo Müller, Sven Geisenhainer, Volkmar Schau, and Wilhelm R. Rossak. A service-oriented software architecture for mobile agent toolkits. In *11th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2004) May 2004, Brno (Czech Republic)*, pages 550–556. IEEE Computer Society Press, 2004.

[8] Peter Braun and Wilhelm R. Rossak. *Mobile Agents – Basic Concept, Mobility Models, and the Tracy Toolkit*. Morgan Kaufmann Publishers, December 2004.

[9] www.edonkey2000.com.

[10] Christian Erfurth, Arndt Döhler, and Wilhelm R. Rossak. A First Look at the Performance of Autonomous Mobile Agents in Dynamic Networks. In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9, Big Island, Hawaii (USA), January 2004*. IEEE Computer Society, 2004.

[11] www.gnutella.com.

[12] Object Management Group. The Common Object Request Broker Architecture, Rev. 2.2, February 1998.

[13] Gene Kan. Gnutella. In Oram [21], pages 94–122.

[14] www.kazaa.com.

[15] Danny B. Lange and Mitsuru Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison-Wesley, 1998.

[16] Adam Langley. Freenet. In Oram [21], pages 123–132.

[17] Mitsubishi Electric ITA. *Mobile Agent Computing – A White Paper*, 1998.

[18] Paul Mockapetris. Domain names - concepts and facilities, 1987. Internet Engineering Task Force, RFC 1034.

[19] Paul Mockapetris. Domain names - implementation and specification, 1987. Internet Engineering Task Force, RFC 1035.

[20] Andrea Omicini, Franco Zambonelli, Matthias Klusch, and Robert Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer-Verlag, 2001.

[21] Andy Oram, editor. *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O'Reilly, 2001.

[22] Giovanni Vigna, editor. *Mobile Agents and Securtiy*, volume 1419 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.