A Framework of Autonomous and Self-adaptable Middleware Services to Support Mobile Agents in Dynamic Networks

Arndt Döhler, Christian Erfurth, and Wilhelm Rossak

Abstract. With the growing size of distributed systems and the higher number of available resources and services in networks, dynamic aspects become more and more important in systems engineering. We believe that there is a real need for decentralized, self-organizing structures to cope with the upcoming challenges. In this paper we describe a framework which provides a self-organizing process that allows to link otherwise autonomous elements in a flexible way. We also have a look at the underlying infrastructure that provides for the use of flexible overlay networks. These overlay networks are used to support mobile end-user platforms and form the basis for high-end services in the mobile agent middleware. One of these services, supporting autonomous and pro-active navigation of mobile agents in an unknown network, is discussed.

1 Introduction

In today’s world, the requirements for networking devices keep changing at a steady rate. More and more devices are enabled to connect to local or global networks. Many of them are mobile in a physical sense and are carried along, acting as personal assistants. This leads in turn to computer networks with a certain level of dynamics regarding structure and connections (their topology). These dynamic networks have to be connected, in most cases, to a more static backbone network to function properly. We believe, therefore, that it is necessary to investigate useful technologies to handle these systems in their dynamics and to find innovative solutions to integrate mobile devices in rigorously distributed information systems.

In the context of dynamically networked environments, mobile agents can be seen as one of the more promising new paradigms for the implementation of fully distributed software systems with a balanced peer-to-peer concept [1]. During the last years, such agent systems were developed mostly in research oriented environments, with a focus on a wide variety of aspects like security, migration, communication, etc. At Friedrich-Schiller-University Jena (FSU), we have focus on an efficient migration process for mobile agents, based on several protocols and migration strategies, as well as on robust infrastructure services.
and a supporting middleware for autonomous agent movement. As a result, the mobile agent system (MAS) Tracy [2,3] was developed – a Java2-based, generic middleware for mobile agent based systems.

So called agencies (Tracy agent platforms) are specialized execution environments for mobile agents. In our approach, every Java-enabled device in the Internet can carry such an agency and, thus, become a network node for mobile agents. Currently, we work on additional system components for the basic middleware layer to network mobile agencies fast, to improve scalability and flexibility, and to provide an information base for mobile agents that supports their pro-activity and adaptability. Especially interesting is the case where the network provides a dynamic environment [4], e.g. if mobile network nodes and services appear and disappear, and agents act as intelligent entities by determining their own path at run-time dynamically in the continuously changing landscape.

In this context, the general workflow of a mobile agent is as follow: On an agent’s journey, it visits only those agencies which provide a resource or service of interest. Furthermore, the agent tries to derive and use a fast path through the network, based on known infrastructure characteristics (as QoS, known services, etc.). Finally, an agent optimizes its transmissions between agencies with the help of several migration strategies, as described by Braun [5]. All information necessary for the agent’s navigation in the network and the related calculations are in our model provided by a so called routing service module.

The movement of a mobile agents is in this context based on a logical network view, i.e. mobile agents look at nodes with agencies only. The cooperation of normally autonomous and independent agencies is, therefore, essential to network agencies on this logical level.

The first part of this paper addresses the routing service which enables the movement and autonomy of mobile agents in such a logical network. The second part covers the issue of necessary agency interaction and describes a basic infrastructure service for dynamic network environments.

2 Service-based Navigation of Mobile Agents

In modern computer networks services can be regarded as dynamic components. To be able to use services, a mobile agent is in need of information about service location and reachability. To answer this need, we have developed a framework called ProNav which stands for pro-active navigation of mobile software entities1. Its most important feature is to locate services and information in the network and to offer this type of data to any mobile agent currently planning its itinerary. This is achieved by integrating the data that is locally acquired by each agent server into a so-called map that enables each agent to recognize and analyze its virtual environment. Even more, an agent is able to adapt to environmental changes without human intervention.

1 See also http://www.pronav.info
From an architectural perspective, ProNav extends any MAS by working as an intermediate layer in-between the actual agent system and the application layer that is formed by specialized mobile agents and their user and application interfaces (see Fig. 1).

Fig. 1. Architectural overview of ProNav and integration as middleware

In Figure 1, an architectural overview of the additionally introduced system components is presented. These components are integrated into the MAS Tracy using stationary agents. In general, such agents are not able to migrate but offer local services. Mobile agents are able to use local services by employing agent to agent communication within the local agency.

ProNav is divided into three major components: Map Module, Route Planner, and Migration Optimizer. In principle each component may be used independently by any mobile agent. However, only by integrating their services a mobile agent will achieve full autonomy and pro-activity for the itinerary planning task.

The Map Module is used by a mobile agent to locate services and to access information on network connection qualities. Connection qualities are especially important for the Route Planner and the Migration Optimizer to achieve optimization. The Route Planner calculates a “short” path through the network. The Migration Optimizer optimizes each single migration included in an agent’s itinerary from a more technological, in our case Tracy-specific, efficiency perspective. This module is mainly designed to reduce network load by selecting and transmitting only those code and data portions of the agent that are needed at the upcoming remote agencies. This is, if necessary, done by a concept called slicing [6]. Other options are to place code in advance in the network, to send...
data home to carry less “luggage”, to change the transmission protocol, etc. The Optimizer is not focus of this article [7].

2.1 A Map with a local focus

ProNav collects information to generate a “network map” offering information to mobile agents. To achieve this, we implemented a Map Module which consists of several network sensors and a map data structure. In addition to information on application-level services provided by the agencies, this module collects and distributes network status information.

The logical network of agencies needs to be subdivided using an infrastructure service (see second part of this article) to achieve scalable network maps. Basically, the map of an agency consists of a partial network graph. The vertices of such a partial graph are the visible agencies of the surrounding area such as all nodes in the local region for instance within the subnetwork and the neighbored remote regions. The edges of the graph represent the end-to-end view transport layer connections between the vertices. Each edge is characterized by the “full qualified domain name” of the remote agency and a couple of network parameters that reflect the current performance of the end-to-end connection. The Map Module uses network sensors with interfering measurement methods on top of Java to get the characteristics of a connection. There are sensors to measure availability, latency, transfer rate, and transmission time of a standard agent.

As an example, we describe the function of the latency sensor which measures round trip times of a minimal data packet. This means the sensor emulates a PING over a TCP connection. The sensor opens a connection to a special port of a remote agency. On this special port the remote sensor listens for measurement requests. After establishing the connection, the sensor starts the time measurement and sends a small packet. The answer of the remote agency is an acknowledgment, the measurement stops and the connection is canceled. After a definable duration a new measurement with the next agency will start.

We have made a set of evaluation measurements to get a feeling of the sensor’s quality. In Fig. 2 the measured values of the latency sensor are compared with values delivered by the PING of the OS in a wireless environment (IEEE 802.11b WLAN) and an Ethernet environment (IEEE 802.3 10BASE-T 10 mbit/s half duplex). The values of the sensor correspond to PING. Due to the application level implementation of sensors the values are a little bit above the PING values.

To flatten peaks measured by sensors, we use forecast modules which generate next expected values on basis of a small time series of measured values. The value of the forecast module which has delivered the best forecast in the last run is entered into the map.

The transfer rate sensor works in almost the same manner. However, for a measurement larger data packets are needed. To transmit useful data thereby these packets are used to exchange and propagate gathered map data (service offers, QoS) between agencies. As a result, every agency has complete information about connection qualities and service offers within its surrounding area. A map with a local focus is created. The map’s information will be summarized. This
compressed information will be propagated to other known regions. So within the local area, every agency has a network map with detailed information about its area and relevant information about known further away regions. Thus a mobile agent is able to locate remote services. To utilize such a service, the agent has to migrate to the remote area, access the local map over there, which is different from the current one, and finally migrate to the actual service location.

2.2 Route Planning

The Route Planner organizes an agent's trip through the network of agencies. The route planning process itself is basically the Traveling Salesman Problem (TSP) [8] which is a NP-complete type of problem. As a consequence, getting an optimal solution in practical application is ruled out. But there are heuristic algorithms (such as local search, genetic, simulated annealing, neural network algorithms etc.) that have been applied extensively for solving such problems [9]. The comparative performance of the algorithms depends on the problem and the given detailed circumstances.

The calculation of an itinerary is based on the map data. We calculate a kind of distance matrix simply by using the reciprocal values of measured transfer rate. This matrix has to be updated at regular time intervals to fit the environment's dynamical behavior. Then, a path finder algorithm is applied in order to get a distance matrix with shortest paths between places (without short cuts). In some experiments, we figured out that our distance matrix is not symmetrically in general. This is caused by oscillating transfer rates values and non symmetrical connections like DSL. For TSP, there are algorithms for asymmetrical (ATSP) [10] and for symmetrical matrices (STSP) [11].

In our case, local optimization algorithms are a good choice. Hence, our route planning process starts with a nearest neighbor search algorithm to generate an initial path through the net. This path is input for further optimizations with an adapted version of the iterated 3-Opt algorithm (I3Opt). In Figure 3, the

![Graph showing round trip times: Latency sensor vs. OS-PING](image-url)
result of the nearest neighbor algorithm is about 36% above optimum (optimum means minimum in this case) but is calculated within 0.7 ms (Pentium II 333 with Java). This route planning is done on a generated matrix of the problem space $tmat$ (triangulated random matrices) with 100 places [12]. Such a matrix is an asymmetrical one where an entry is the shortest path between two places.

To avoid unnecessary calculation, we compare the so far calculated migration time with the path improvement during the last steps. If the time benefit of the last 20 ms calculation is not greater than the path improvement the calculation stops. Thereby it takes also the calculation power into account.

The result of the calculation is an agent’s initial itinerary. During an agent’s journey it might be useful, or even necessary, to modify this itinerary (changed network status, new services, etc.). This can be done by the agent itself without any human-agent interaction.

### 2.3 A Sample Scenario

The following scenario describes the application of ProNav in a network of agencies. Thereby a mobile agent visits a set of agencies while migrating through the network to fulfill its task.

A user (the owner) hands over a task to an agent. Normally, such a task should not contain information on \textit{HOW} to fulfill. Hence, the agent has to organize the journey through the network by itself. Therefore, the agent searches for suitable services in the map provided by the local agency. This map contains information on services within the local region and some network characteristics. The search result is a set of agencies that should be visited. Now the agent may trigger the Route Planner to use the available map’s information on connection topology and qualities to identify a possible trip through the network. The result is a first
travel plan – the itinerary. Before the agent begins the trip, it might use the Migration Optimizer to optimize the trip from an efficiency perspective.

Fig. 4. Proactive navigation of a mobile agent in a dynamical environment

Now the agent “executes the itinerary” and starts the migration. During the trip the agent visits service points and communicates and cooperates with other agents. At any point in time, but at least when migrating to further away agencies (the map’s information is more blurred for further away agencies), the agent may fine-tune and re-adapt its itinerary. This is achieved by taking advantage of information now available. Finally, after its trip, the agent hands over the results to its principal. This might include a description of the visited agencies, a kind of travel report.

As already mentioned, in this part of our work the goal was to provide components that enable mobile agents to act in an autonomous fashion, without having to interact with the end-user to fulfill routine tasks. A user should be able to concentrate on WHAT the agent has to do and not on HOW it navigates in a basically unknown and potentially dynamic network. The basis for this service-based navigation is, however, an improved infrastructure that steps in to substitute for detailed end-user knowledge.

3 Infrastructure Concepts

3.1 The logical network view

In distributed systems nodes are able to communicate using operating system functionality to access the underlaying physical network. For direct communication between nodes these have to know the receiver’s address (or FQDN\(^2\)). In general a communication program has to get this address during configuration time or by direct user input. In dynamic network environments it is hardly possible to know potential all communication partners in advance. One way to adapt

\(^2\) fully qualified domain name
to changes in the network is the usage of logical overlay networks – a middleware concept to network nodes as dynamically as the topology keeps changing.

Logical networks are also deployed to structure large networks into separated communication channels. This comes more and more into play to handle the enormously growing quantity of (mobile) nodes and the information overload in world-wide networked activities. Large networks have turned into complex and steadily evolving systems that are no longer manageable by humans within the given time limits and quality parameters. Hence, to get good value from this resource, the task is handed over to a middleware.

3.2 A middleware for MAS

The middleware of a MAS is basically a peer-to-peer (P2P) system. However, compared with some of the better known file sharing systems [13–15] they have some different traits of character. The design of MAS is strictly focused on mobile agents and these mobile agents act very dynamically and autonomously. A MAS supports mobile agents by providing agencies and agent-specific services. The following example should pinpoint the difference.

Mobile agents, as described in section 2.3, use maps to navigate and to plan their route through the logical network. The quality of this approach depends directly on the actuality of the map’s information. Since the map service in a MAS does not know when an agent will access its map information, it must care about getting information continuously. Furthermore, to support a fast route planning for every potential mobile agent and its intentions, maps must contain every possible service, before any agent ever used the map service. In contrast to a MAS, in file sharing systems the search for each file (information unit) is triggered by a specific, separate search request. Information for this request is determined only based on its specific needs and not for every possible combination of requests, as in a MAS. With regards to timing, a file sharing system does not have any hard real-time constraints and, thus, does not exhibit the need to update its information base (many times) in advance to satisfy fast reaction times.

As a consequence, MAS services stress an independent and time-driven middlewares more than file sharing systems. However, in a global scenario the pure number of possible nodes and services makes it impossible for a MAS to scale within the targeted quality parameters. Assuming that mobile devices may also host an agency in a MAS, the problem of information actuality gets worse.

Hence, we designed a middleware concept to support large quantities of nodes and services and, at the same time, mobile agents on highly dynamic, mobile hosts. To achieve both of these counteracting objectives, we separate the network into two logical levels:

The upper level offers a scalable solution for joining, the lookup and leaving a service in a global context. As a technical solution to this problem, a distributed hash table (DHT)-based infrastructure is, in our opinion, first choice. A DHT stores value pairs that indicate the node’s address and the related service name. Such systems are well known in P2P research and their mechanisms and behavior
are widely understood [16]. Our research indicates that P-Grid [17] or TLS [18]
based DHTs do the best job.

In this paper, we will no longer discuss these issues in any further depths,
but focus the lower level that provides a highly dynamic, autonomous and self-
adapting environment to support MAS on mobile nodes. Each of the presented
services is a modular software component and implemented in JAVA 5.0.

3.3 QuickLink – A local logical high performance network

Network dynamics that stem from node’s joining and leaving the network occur
on every layer of the communication protocol stack. A single node’s protocol
stack will determine such dynamics on lower communication layers very fast.
However, the higher the communication protocol layer, the more inefficient
the communication protocols and the larger the possible network size. (Note: Already
on the IP layer a global Internet is possible). Logical networks are also called
application layer networks. They use application layer protocols for networking
and have global expansion. So they are basically less suitable to high network
dynamics.

Since network dynamics can be better detected in local network environ-
ments, we prefer local groups of nodes to control network dynamics. In our
basic infrastructure approach QuickLink, we group nodes in a network-aware
fashion where the term network relates to an IP-subnet. Nodes of a IP-subnet
are usually topologically close together and belongs to the same administration
domain. The local network is trustworthy, reliable and high-performance. For
that reasons QuickLink uses local limited unreliable UDP broadcasts intensively
for self-announcement and to maintain the network structure. These broadcasts
reach only the nodes in the IP-subnet. In the Internet broadcasts are avoided due
to their evocation of a high network load. In local environments this approach
is an efficient way to reach all nodes by submitting only one message.

QuickLink forms a logical network structure in a ring topology (see Fig. 5).
To maintain the structure every QuickLink node listens on a special port named
cycle port. Every cycle time period a QuickLink node sends its cycle broadcast
to cycle port. The cycle broadcast contains a list of IP-addresses and some addi-
tional information (see below) on all currently participating QuickLink nodes.
The entries of the list come in order of the known IP-addresses. All QuickLink
nodes receive the broadcasted list and compare it with their own list. In case of
a variance the old list will be replaced by the received newer list. So all nodes
can stay up to date, respectively will be updated at the same time by only one
message. Every node checks the sender address whether it is the next node in
the list of IP-addresses. Only the successor of the former sender will send its
broadcast after the cycle time period.

Before a node may join QuickLink it has to assert the presence of a exist-
ing QuickLink network by listening to the cycle port. In case of no received
broadcasts during the join time period the QuickLink node builds up its own
QuickLink list with itself as the only one member. As a consequence, this node
is predecessor and successor of itself and broadcasts every cycle time period to
the local network. In any other case the potential join candidate receives a cycle broadcast from an existing QuickLink network. It replaces its own list with the received one, inserts its own entry between the entries with the nearest IP-addresses and proofs now whether it is the new successor of the last sender. If it is the new successor, the node will wait for the next broadcast which must come from the actual node and his new successor, otherwise it does not have to wait. After that process, the new candidate joins the QuickLink network by sending an *join broadcast* with the complete new list to the *update port*, which differs from the cycle port. All nodes receive the update broadcast and replace their lists with newer list. This mechanism ensures that all nodes will be updated at the same time by only one message and the new node doesn’t unsettle the cycle mechanism. Note that QuickLink needs only one message and at most two cycle time-slots to join the network.

Leaving the QuickLink network happens, as in other dynamic systems, mostly by the unpredictable disappearance of the node. That means in worst case the missing node will detected after a full cycle. If the cycle broadcast of a node does not arrive, the successor will wait one cycle period before it will send its own cycle broadcast. In this message the successor marks the missed node as "*missed*" in the list and the second successor of the missed node pings the missed node to prove its reachability. If the missed note is reachable and was only in transient trouble, the second successor rejects the "*missed*" state, otherwise it rejects the missed note from the list. This mechanism takes care to eliminate occasionally disappeared messages and improves robustness of the system in general. So in worst case the leave of a node takes \( n + 2 \) cycle times. Note that information for a missed node, that come from the upper level of the infrastructure or the MAS, will be handled as described by the QuickLink node where the problem occurred. If desired nodes are not reachable, the node will update its list and broadcasts the new list using the join mechanism and the update port.

There is, in theory, no upper limit of nodes that limits the management capabilities of QuickLink. However, to achieve a high performance network, the number of nodes should not exceed about hundred nodes to keep the broadcast messages small and the cycle manageable. At a cycle time period of one second QuickLink can insert any node within one or two seconds and detect missed nodes in less than two minutes.

### 3.4 FitnessPerformer

The FitnessPerformer is a modular software component which measures recurrent computing power, several memory values and the period of running the underlying JAVA VM. These measured values represent the general performance of a node, its current utilization and its stability as a potential member of a network. Other services and MAS can use these values to get an overview of the capabilities of the own host. If FitnessPerformer runs together with QuickLink, some or all performance values can be coupled to the node entry in the QuickLink list. Note that measurements of QoS values of the network are currently not planned in this module, as we assume that logical links in local environments are

© GESTS-Oct.2005
equal and symmetric. Furthermore there are several net sensors in the ProNav framework. For further details we refer to our upcoming publications.

3.5 ServiceJuggler — a network-adaptive service manager

ServiceJuggler is also a modular software component which provides an advanced infrastructure service for MAS and other middleware components. On top of QuickLink it launches an intelligent service to manage application services. At this time we have to distinguish application services from infrastructure services. To keep the lists in QuickLink small and to get a good performance, only infrastructure services are coupled to the IP-address entries in the lists. We decode these services hard with only one bit per service to indicate whether it is present or unpresent. In our framework middleware services provides a basic service to network nodes together and to keep the network stable and appropriate. Routing Service, QuickLink, FitnessPerformer and ServiceJuggler are such services. They support a MAS as part of the infrastructure directly. However, mobile agents needs in general other services to fulfill their user tasks. Such service we call application services.

ServiceJuggler only manages application services. It uses QuickLink to get an overview over the present nodes and their current performance. With this information ServiceJuggler can start, migrate and stop a directory service for application services. In the case of a directory service, the period of running time of the node as an indicator of its stability, while memory size and utilization are the most interesting performance values. So the most stable node with sufficient memory capabilities is the most suitable one to run a ServiceJuggler directory service. If the directory service is started, the QuickLink module of the concerned...
node sets its ServiceJuggler flag on high and updates by the update mechanism all the other nodes. As a consequence, all nodes and their MAS know where the service directory currently executes. A MAS and other service providers may register their application services and MAS-specific middleware services like the Map Module can cache a local copy of the directory service entries to provide their own service. In case of a durable change of performance values, the directory service can be migrated to another node. Thus, ServiceJuggler can adapt to the network dynamics in the local logical network structure.

![Fig. 6. The logical network completed with the global P-Grid](image)

The node that runs the directory service is still a peer node of the system. However, it plays a specialized role in the whole local network region. This node should also be the connective link to the global network, the upper level of the network structure, because of its performance values. Because of the specific role of this node we call it region manager. It registers all known local application services in the global system under its own address (please see Fig. 6). This takes effect in the case of highly redundant occurrences of the region manager’s address in the global system. Assuming a very stable (stationary and high
available) node as region manager the updates in the global system may be reduced. Furthermore, since directory service is able to store search requests from and to the global system, the region manager benefits from the small-world-phenomenon [19] be establishing shortcuts and clustering often interacting regions. This effect improves the search effort of often used services dramatically.

4 Conclusion

In this paper we have shown mobile agent systems as one of the more promising alternatives to develop truly distributed systems in large and dynamic networking environments. For a mobile agent the opportunity for a proactive and autonomous planning of its itinerary is essential in this context. This feature is offered by ProNav as part of a generic infrastructure framework. Thus, an agent is enabled to fulfill the task to plan its itinerary itself and independently of its owners. ProNav also provides enough information and flexibility for regular updates and changes in the itinerary during its execution. This helps to react immediately and in an autonomous fashion to changes in the environment.

A robust and high-performance infrastructure organization is important for ProNav to function as described. However, logical networks exhibit a high level of internal dynamics: Nodes and services are added or deleted, quality and performance of a node changes over time, etc. Therefore, the concepts of QuickLink and ServiceJuggler were introduced and enhanced with the performance functionalities of FitnessPerformer to better react to the dynamics of the system and to provide a suitable basis for the ProNav module.

The introduced approaches are in general not limited to MASs. They can be used to enable the self-adaption of any autonomous distributed system that supports a minimum of communication and autonomy.

References


©GESTS-Oct.2005
122 A Framework of Autonomous and Self-Adaptable