An Infrastructure-based Approach to Support Dynamic Networks with Mobile Agents

Arndt Döhler, Christian Erfurth, and Wilhelm Rossak

Computer Science Dept., Friedrich-Schiller-University Jena, 07743 Jena, Germany,
{arndt.doehler, erfurth, rossak}@informatik.uni-jena.de,
WWW home page: http://swt.informatik.uni-jena.de/

Abstract. With the growing size of distributed systems and the higher number of available resources and services in networks dynamical aspects become more and more important in systems engineering. We believe that there is a real need for decentral, self-organizing structures to cope with the upcoming challenges. In this paper we describe a framework which provides a self-organizing infrastructure that allows to link otherwise autonomous elements in a flexible way and adapts dynamically to changes in the underlying network. This framework is implemented as an extension of the mobile agent system Tracy, which is also a product of our university. The Tracy Domain Management module is part of the framework and provides the basis for segmenting the infrastructure. Another module we are going to discuss in this paper facilitates autonomous and proactive routing of mobile agents. Agents form the application layer of the system. Routing is triggered by the needs an agent inherits from its owner and then matched to the resources and services available in the network in an iterative fashion. We describe concepts, design issues and first results of our work with Tracy and the use of these additional Tracy modules.
Key words: Distributed systems, self-organization, rule-based behavior, proactive navigation of mobile agents, mobile agent systems.

1 Introduction

In the context of networked environments, mobile agents can be seen as a new paradigm for the implementation of fully distributed software systems with a balanced peer-to-peer concept [1]. During the last years at Friedrich-Schiller-University Jena (FSU), we have developed our own mobile agent system (MAS) Tracy [2, 3]. Tracy is a Java2-based middleware that supports the efficient migration of mobile agents over several protocols and migration strategies. So called agencies (Tracy agent servers) are the specialized execution environments for mobile agents. In our approach, every Java-enabled device in the Internet can be such a network node. Currently, we work on additional system components on top of the basic middle-ware layer to network mobile agencies by a self-organizing mechanism, to improve scalability and flexibility, and to provide an information base for mobile agents that supports their proactivity and adaptability. Especially interesting is the case where the

network provides a dynamical environment [4], e. g. if mobile network nodes and services appear and disappear, and where agents act as intelligent entities by determining their own path at run-time dynamically in the continuously changing landscape.
The movement of mobile agents is based on a logical network view, i. e. mobile agents discern agencies only. The cooperation of normally autonomous and independent agencies is essential to network agencies on such a logical level. The first part of this paper covers that issue and describes a self-organizing network of agencies.
The second part of the paper addresses the routing service which improves the movement of mobile agents in such networks and supports their autonomy. On an agent's journey, it visits only those agencies which provide a resource or service of interest. Furthermore, the agent tries to use a fast path through a network based on known infrastructure characteristics (as QoS). Finally, an agent optimizes its transmissions between agencies with the help of several migration strategies described by Braun [5]. All information necessary for the agent's navigation in the network and the related calculations are provided by the routing service module.

2 Concepts of the basic infrastructure

2.1 A logical network

A node with an agency is the basic element of our infrastructure. All networked agencies form a logical or virtual application-level network. Every agency offers services managed by local stationary agents. Mobile user-task agents (application-agents) can use these services by local message exchange with the stationary agents. To use remote services on other agencies, a mobile agent must migrate to the desired agencies for local communication with the stationary agents. This approach is typical for a strictly defined MAS and has been described e. g. in Braun [5].
In this context an autonomous decision of a mobile agent is based on a couple of basic capabilities each agency must exhibit: Knowledge regarding the existence of other agencies and theirs offered services is essential, the propagation of this information through the network is desirable, and the infrastructure must be enabled to handle network changes.
The problem is, that in the worst case every agency would have to hold information regarding every other known agency and, thus, a fully intermeshed virtual network comes into being. Since fully intermeshed networks aren't a scalable solution in industrial size networks, we decided to separate the network into manageable and interrelated chunks.

2.2 Topology – The Domain Concept

The basic idea of our approach is to split the whole MAS network into *domains* (see Fig.1), which are limited to IP-subnetworks. All agencies within a domain register at a single agency called *domain manager*. In

our approach all agencies have basically fully equal rights and basic capabilities since the *DomainInformationAgent*, the domain management component of Tracy, is present on each agency. So we have a peer-to-peer system. By launching an agency as a domain manager it takes on a specific role and offers the relevant domain management services. From the network management view this role-based behavior can be seen as a client-server behavior, where the domain manager plays the role of the server.

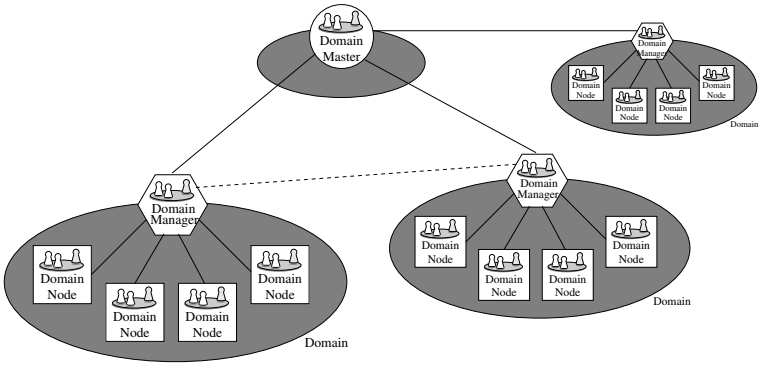

Fig. 1. Domain Concept: A structured network of agencies.

The domain manager is responsible to manage all other agencies in a domain called *domain nodes* and to hold connections to other domains. Every domain node has its unique domain manager, and the domain manager knows all domain nodes that are currently active in its domain. If an agency starts or stops, it has to register respectively check out with the domain manager node.

The domain manager holds the complete information of its domain nodes and of itself. It propagates this information to all domain nodes inside its domain, but not beyond. Thus, all agencies within the domain know each other, and form again a fully intermeshed network of a limited size. In practical applications we have learned to expect not more than 60 agencies per domain.

To re-integrate the whole logical agent system network, domains have to be linked together. For that reason domain managers contact a unique *domain master*. The master is a specialized domain manager which manages only other domain managers and interconnects them with each other. In future, we plan some more domain masters to prevent the single point of failure problem. On this level, only summarized information are exchanged.

Since it is possible to launch more than one domain, this approach is capable of handling very large networks in a piece by piece fashion, while it allows for scalability at the same time. Only inside a domain information and resources are fully intermeshed. In-between domains the mesh

is broken. This makes it, of course, necessary for a mobile agent to move into a domain before it can access its resources.

2.3 Valency of a node – Priority Concept

A Tracy domain is a self-organized basic cell of the infrastructure. When a Tracy agency is launched, it checks the presence of other agencies by sending a UDP-multicast first. If a domain manager answers, the agency must register with the domain manager by sending a mobile agent. In the case of absence of a domain manager the agency becomes domain manager itself. If several agencies were launched simultaneously or a domain manager breaks down, agencies compete to become the domain manager according to the first-come-first-serve principle.
To influence the role allocation according to importance of an agency, a priority value can be assigned to every agency [6]. The priority is modeled as a byte value and ranges between -128 and $+127$. It should correspond to the performance, the quality of the network connection, and the reliability of a node. Currently the priority value has to be fixed before the agency starts. After the launch it can't be changed.
With the concept of priorities, the launching process of a domain information agent changes slightly. When a domain manager receives registration messages from other nodes, it now compares their priorities with its own value. If its own priority is lower than one of a new node, that node becomes the new domain manager.
The drawback of this solution is the fixed assignment of priority values. Furthermore, the programmer has to know the absolute valency of his device or the valency in ratio to the other agencies before its agency starts. This leads to an arbitrary or appraised allocation of the priority value.

2.4 Valency of a node – Dynamical Priority Assignment

Our new approach is, therefore, to dynamically assign proper priority values during the runtime of an agency. Launching an agency happens as described before, but after registration with the domain manager respectively become domain manager itself an agency performs performance measurements (computing power, memory size and others) by some sensors. The performance measurements reflect the performance of the node and form an abstract view on the local capabilities of the system environment of the agency.
The Map Module which will be discussed in section 3.1 provides information on known services and on network connection qualities. Together with the performance measurement results as an information base (see Fig. 2) it is now possible to calculate a proper priority value to support an automated and useful choice of a domain's manager.
Performance measurements can be regularly repeated and the time interval can be dynamically adapted to the current network situation with the help of small statistics on the last measured values. If there are changes in the network accessibility (e. g. by a more badly signal-noise-ratio of

a 802.11 connection) or the usage rate of a node (e. g. by a higher utilization by other applications) an agency will take notice of it and the priority can be changed.
If the transfer rate decreases or the RTT to a node increases, or a node's computing power decreases and the memory utilization increases, the node is less suitable to manage a domain because this leads to additional utilization in computing power, memory and network load. So the priority of such a node has to decrease (and vice versa) according to the relative changes of the values.
If a service is launched on a node, it must be checked if it is an administrative service or an application service. In the first case the priority has to be decreased because of direct, additional agency utilization. In the second case the service may be performed by an application outside of the agency. From the performance perspective this application does not directly increase the load the agency has to handle. Therefore, the additional utilization of the host platform should be measured by the sensors. Thus, the start of an application service means mostly that the host platform is a stable and reliable computer with an excellent network connection. This more logical hint can't be measured or performed otherwise but ought to be observed over time. If the reliability assumption is affirmed, priority can be increased.

2.5 Dynamical Priority Assignment – Scenario

A typical scenario is shown in Fig. 2 and describes the usefulness of dynamical priority assignment. On the left side the priorities of the domain nodes are predefined, static and without any relation to the logical network's real situation. From the bottom upwards there are three layers which corresponds to three logical views. A *network quality view* comes from the Map Module which is fed by several network sensors. A *view of the logical agency network* and the roles of the known agencies comes from the DomainInformationAgent itself. Several node performance and utilization sensors feed the DomainInformationAgent directly. Note, that the sensors are not shown in the Figure. The most abstract logical view forms the top layer of this Figure: *the service view*.
After calculating the priority each node sends the value to the domain manager. The most prioritized node becomes the new domain manager, shown on the right side of the Figure.

2.6 Discussion

The dynamical priority assignment represents an abstract closed control loop. The priorities are the control variables and the role allocation of the domain manager is the controlled system. An inherent design problem of closed control loops is to prevent unstable and instable states, which may caused by too strong feedbacks of the controlled system or by disturbances. In our case an instable state means a recurrent shift of the domain manager role. By the role changing itself, the priority of the new domain manager will decrease due to the utilization by its new domain

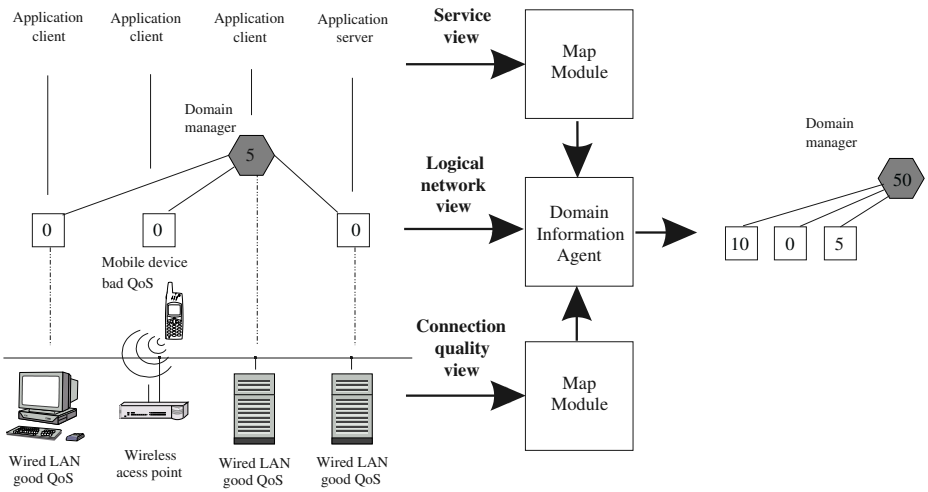

Fig. 2. Dynamical Priority Assignment

management service. To prevent unstable states, we stabilize the system by introducing a programmable threshold value that must be reached by a new domain manager candidate. Currently we program a fixed threshold value of 20%. Additionally we use small statistics on older priority values to get a time-dependent change of the priority which stabilize the control loop as well. A drawback of this solution is the relatively slow change of the domain manager role in situations with high network dynamics.

3 Proactive Navigation

In modern computer networks services can be regarded as dynamical 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*. 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. These mechanisms utilize the domain concept, as discussed above, as a basic feature and extend it with additional functionality.

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. 3).

In Figure 3, an architectural overview of the additionally introduced system components is presented. These components are integrated into

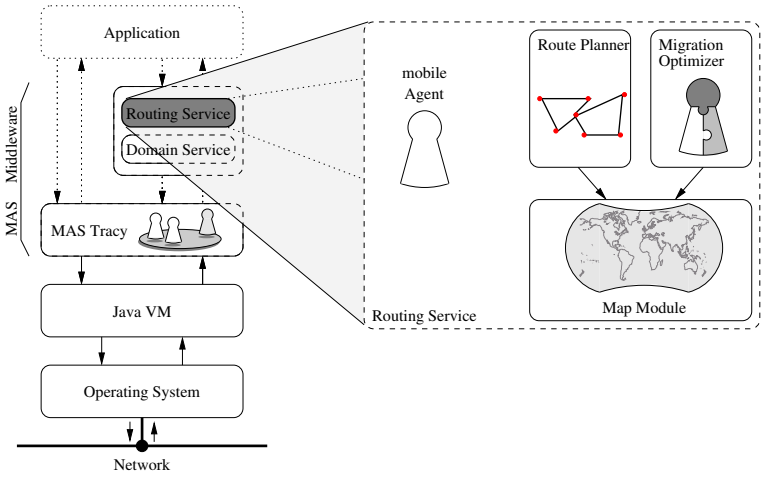

Fig. 3. Architectural overview of *ProNav* and integration as middle-ware.

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: the *Map Module*, the *Route Planner*, and the *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 [7]. 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 paper [8].

3.1 A Logical Network Map

Building on the Domain Service, *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 the Domain Service described above to achieve scalable network maps. Basically, a 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 domain including the domain manager and the neighbored remote domains each represented by its domain manager. 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. 4 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.

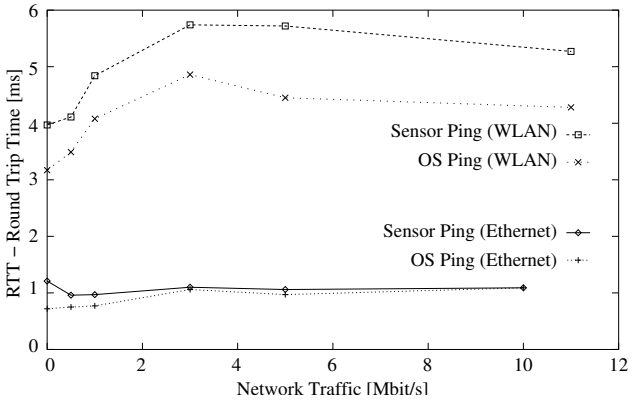

Fig. 4. Round trip times: Latency sensor vs. OS-PING

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 domain. A Domain Map is created. The domain manager summarizes its Domain Map information and propagates this compressed information to other known domains. So within a domain, every agency has a network map with detailed information about the local domain and relevant information about known remote domains (via the domain master). Thus a mobile agent is able to locate services within a remote domain. To utilize such a service, the agent has to migrate to the domain manager of the remote domain, access the local Domain Map, which is different from the current one, and finally migrate to the actual service location.

3.2 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) [9] 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 [10]. 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) [11] and for symmetrical matrices (STSP) [12].
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 5, the 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 [13]. Such a matrix is an asymmetrical one where an entry is the shortest path between two places.

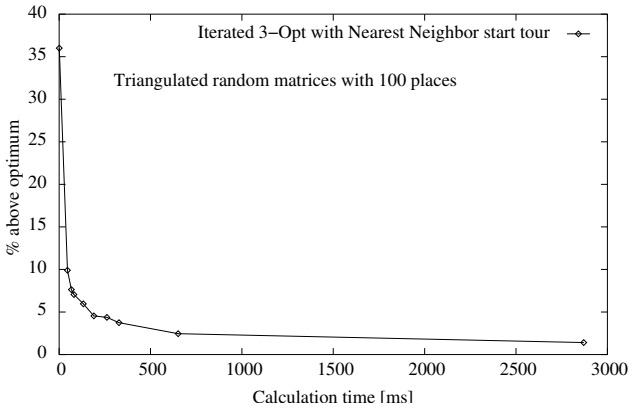

Fig. 5. Route planning with tmat100.

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.

3.3 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 *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 domain 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. 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 in different

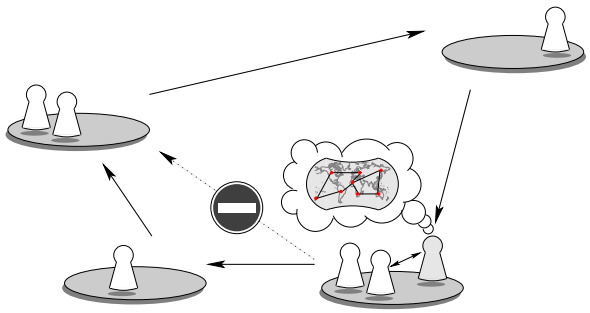

Fig. 6. Proactive navigation of a mobile agent in a dynamical environment.

domains. 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 indicated, we want to provide an infrastructure that enables agents to be more autonomous. A user should concentrate on *WHAT* the agent has to do and not on *HOW*.

4 Conclusion

We see 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. The agent's programmer and user do no longer have to plan the itinerary for the agent. The agent is enabled to fulfill this task itself and independently of its owners. *ProNav* also provides enough information and flexibility to abandon the notion of a fixed route through the network and allows 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 basic robust infrastructure organization is important for *ProNav* to function as described. However, logical or virtual networks exhibit a high level of internal dynamics: Agencies and services are added, deleted, or modified, connection quality changes over time, etc. Therefore, the Domain Concept was introduced and enhanced with new priority functionalities to better react to the dynamics of the system and to provide a basis for the *ProNav* module.
The introduced approaches, the Domain Concept and *ProNav*, are in general not limited to MASs. They can be used to enable the self-organization of any autonomous distributed system that supports a minimum of communication and autonomy.
As a next step we plan to model the dynamic behavior of the system formally. We also plan to use control loop approaches, well known in

electrical engineering, to analyze the effects of dynamical priority assignment. We also look at other infrastructures for distributed systems to go beyond the current MAS-based implementation.

References

1. Vigna, G.: Mobile Code Technologies, Paradigms, and Applications. PhD thesis, Politecnico di Milano (1998)
2. Friedrich-Schiller-Universität Jena, Software Engineering Group: Tracy – The Mobile Agent System. URL: http://tracy.informatik.uni-jena.de (2004)
3. Braun, P., Erfurth, C., Rossak, W.: An Introduction to the Tracy Mobile Agent System. Technical Report Math/Inf/00/24, Friedrich-Schiller-Universität Jena, Institut für Informatik (2000)
4. Erfurth, C., Rossak, W.: Characterization and Management of Dynamical Behaviour in a System With Mobile Agents. In Unger, H., Böhme, T., Mikler, A., eds.: Innovative Internet Computing System - Second International Workshop, IICS 2002, Kühlungsborn (Germany), June 2002. Volume 2346 of Lecture Notes in Computer Science., Kühlungsborn (Germany), Springer Verlag (2002) 109–119
5. Braun, P.: The Migration Process of Mobile Agents - Implementation, Classification, and Optimization. PhD thesis, Friedrich-Schiller-Universität Jena, Institut für Informatik (2003)
6. Braun, P., Eismann, J., Rossak, W.: A Multi-Agent Approach To Manage a Network of Mobile Agent Servers. Technical Report 12/01, Friedrich-Schiller-Universität Jena, Institut für Informatik (2001)
7. Fensch, C.: Class Splitting as a Method to Reduce Network Traffic in a Mobile Agent System. Master's thesis, Friedrich-Schiller-Universität Jena, Institut für Informatik (2001)
8. Schaaf, M.: Entwicklung und Implementierung einer Komponente zur Migrationsoptimierung für Mobile Agenten. Master's thesis, Friedrich-Schiller-Universität Jena, Institut für Informatik (2003)
9. Lin, S.: Computer Solutions of the Traveling Salesman Problem. Bell System Technical Journal **44** (1965) 2245–2269
10. Johnson, D.S., McGeoch, L.A.: The Traveling Salesman Problem: A Case Study in Local Optimization. In E.H.L.Aarts, J.K.Lenstra, eds.: Local Search in Combinatorical Optimization. John Wiley and Sons, Ltd. (1997) 215–310
11. Johnson, D.S., Gutin, G., McGeoch, L.A., Yeo, A., Zhang, W., Zverovitch, A.: Experimental analysis of heuristics for the atsp. [14] 445–487
12. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the stsp. [14] 369–444
13. Cirasella, J., Johnson, D.S., McGeoch, L.A., Zhang, W.: The Asymmetric Traveling Salesman Problem: Algorithms, Instance Generators, and Tests. In: Proceedings of ALENEX. (2001)
14. Gutin, G., Punnen, A.P., eds.: The Traveling Salesman Problem and its Variations. Kluwer Academic Publishers (2002)